.

Aegis
A Project Change Supervisor

Reference Manual

Peter Miller
pmiller@opensource.org.au

.

This document describes Aegis version 4.25
and was prepared 23 July 2016.

NAME

aegis − project change supervisor

aegis (ee.j.iz) *n.*, a protection, a defense.

DESCRIPTION

Aegis is a CASE tool with a difference. In the spirit of the UNIX Operating System, *Aegis* is a small component designed to work with other programs.

Many CASE systems attempt to provide everything, from bubble charts to source control to compilers. Users are trapped with the components supplied by the CASE system, and if you don't like one of the components (it may be too limited, for instance), then that is just tough.

In contrast, UNIX provides many components of a CASE system − compilers, editors, dependency maintenance tools (such as make), source control tools (such as RCS). You may substitute the tool of your choice if you don't like the ones supplied with the system − gcc, jove, cake, to name just a few. *Aegis* adds to this list with software configuration management, and true to UNIX philosophy, *Aegis* does not dictate the choice of any of the other tools (although it may stretch them to their limits).

Enough hype, what is it that *Aegis* does? Just what is software configuration management? This question is sufficiently broad as to require a book in answer. In essence, *Aegis* is a project change supervisor. It provides a framework within which a team of developers may work on many changes to a program independently, and *Aegis* coordinates integrating these changes back into the master source of the program, with as little disruption as possible. Resolution of contention for source files, a major headache for any project with more than one developer, is one of *Aegis'* major functions.

It should be noted that *Aegis* is a developer's tool, in the same sense as make or RCS are developer's tools. It is not a manager's tool − it does not provide progress tracking or manage work allocation.

BENEFITS

So why should you use *Aegis*?

Aegis uses a particular model of the development of software projects. This model has a master source (or baseline) of a project, and a team of developers creating changes to be made to this baseline. When a change is complete, it is integrated with the baseline, to become the new baseline. Each change must be atomic and self-contained, no change is allowed to cause the baseline to cease to work. "Working" is defined as passing it's own tests. The tests are considered part of the baseline. *Aegis* provides support for the developer so that an entire copy of the baseline need not be taken to change a few files, only those files which are to be changed need to be copied.

In order to ensure that changes are unable to cause the baseline to cease to work, *Aegis* mandates that changes be accompanied by at least one test, and that all such tests be known to complete successfully. These steadily accumulated tests form an ever increasing regression test suite for all later changes. There is also a mandatory review stage for each change to the baseline. While these requirements may be relaxed per-change or even per-project, doing so potentially compromises the "working" definition of the baseline.

The win in using *Aegis* is that there are *O(n)* interactions between developers and the baseline. Contrast this with a master source which is being edited directly by the developers − there are *O(n!)* interactions between developers − this makes adding "just one more" developer a potential disaster.

Another win is that the project baseline always works. Always having a working baseline means that a version is always available for demonstrations, or those "pre-release snapshots" we are always forced to provide.

The above advantages are all very well − for management types. Why should Joe Average Programmer use *Aegis*? Recall that RCS provides file locking, but only for one file at a time. *Aegis* provides the file locking, atomically, for the set of files in the change. Recall also that RCS locks the file the instant you start

editing it. This makes popular files a project bottleneck. *Aegis* allows concurrent editing, and a resolution mechanism just before the change must be integrated, meaning fewer delays for J.A.Programmer.

Aegis also has strong support for geographically distributed development. It supports both push and pull models, and many distribution topologies. Aegis' normal development process is used to validate received change sets before committing them.

ARCHIVE SITE

The latest version of *Aegis* is available by HTTP from:

```
URL:    http://miller.emu.id.au/pmiller/
File:   aegis.html              # the Aegis page
File:   aegis.4.25.README       # Description, from tar file
File:   aegis.4.25.lsm          # Description, in LSM format
File:   aegis.4.25.ae           # the complete source, aedist format
File:   aegis.4.25.spec         # RedHat package specification
File:   aegis.4.25.tar.gz       # the complete source
```

This directory also contains a few other pieces of software written by me. Some are referred to in the *Aegis* documentation. Please have a look if you are interested.

Mirrors

See http://miller.emu.id.au/pmiller/ for a list of mirror sites.

Aegis is also carried by `metalab.unc.edu` in its Linux archives. You will be able to find Aegis on any of its mirrors.

```
URL:    ftp://metalab.unc.edu/pub/Linux/devel/vc/
File:   aegis.4.25.README       # Description, from tar file
File:   aegis.4.25.lsm          # Description, in LSM format
File:   aegis.4.25.spec         # RedHat package specification
File:   aegis.4.25.ae           # the complete source, aedist format
File:   aegis.4.25.tar.gz       # the complete source
```

This site is extensively mirrored around the world, so look for a copy near you (you will get much better response).

MAILING LIST

A mailing list has been created so that users of *Aegis* may exchange ideas about how to use *Aegis*. Discussion may include, but is not limited to: bugs, enhancements, and applications. The list is not moderated.

The address of the mailing list is

```
aegis-users@auug.org.au
```

Please **do not** attempt to subscribe by sending email to this address. It is for content only.

How To Subscribe

To subscribe to this mailing list, visit the Aegis-users mailing list page (`http://www.auug.org.au/-mailman/listinfo/aegis-users`) and go through the **subscribe** dialogue.

Archive

The mailing list is archived at eGroups. The URL is `http://www.egroups.com/list/aegis-users/info.html`

No Files By EMail

The software which handles this mailing list **cannot** send you a copy of *Aegis*. Please use FTP or ftp-by-email, instead.

BUILDING

Instructions on how to build and test *Aegis* are to be found in the *BUILDING* file included in this distribution.

SOME HISTORY

The idea for *Aegis* did not come full-blown into my head in the shower, as some of my programs do, but rather from working in a software shop which used a simplistic form of something similar. That system was held together by chewing-gum and string, it was written in a disgusting variant of Basic, and by golly the damn thing worked (mostly). *Aegis* is nothing like it, owes none of its code to that system, and is far more versatile. It turns out that the system used is nothing new, and is described in many SCM textbooks; it is the result of systematically resolving development issues for large-ish teams.

Since that company decided to close down our section (the company was under attack by a hostile takeover bid) we all moved on simultaneously (all 60 of us), sometimes working together, and sometimes not, but always keeping in touch. With suggestions and conversations with some of them early in 1990, the manual entries for *Aegis* took shape, and formed most of the design document for *Aegis*.

Since getting the first glimmerings of a functional *Aegis* late in 1990 it is increasingly obvious that I never want to be without it ever again. All of my sources that I modify are instantly placed under *Aegis*, as is anything I distribute. All code I write for myself, and all new code I write for my employer, goes under *Aegis*. Why? Because it has fewer bugs!

Example: one of the sources I carry with me from job to job is "cook", my dependency maintenance tool. Cook had existed for 3 years before *Aegis* appeared on the scene, and I used it daily. When I placed cook under *Aegis*, I found 6 bugs! Since then I have found a few more. Not only are there now fewer bugs, but they never come back, because the regression test suite always grows.

Branching

In 1997 the full branching support was released (it took nearly 18 months to retro-fit. The underlying data structures for projects and change sets need to be merged. While I noticed back in 1990 that they were very similar, it wasn't until branch support design was well underways that they should have been the same data structure from the beginning.

Geographically Distributed Development

In 1999 a conversation on the `aegis-users` mailing list resulted in the creation of *aedist*, a program which packages and unpackages Aegis changes so they can be sent by e-mail, or WWW or whatever. With 20:20 hindsight, this could have been done way back in 1991, because the basic idea builds on Aegis change process model.

Windows NT

Aegis depends on the underlying security provided by the operating system (rather than re-invent yet another security mechanism). However, in order to do this, Aegis uses the POSIX *seteuid* system call, which has no direct equivalent on Windows NT. This makes porting difficult. Single-user ports are possible (*e.g.* using Cygwin), but are not usually what folks want.

Compounding this is the fact that many sites want to develop their software for both Unix and Windows NT simultaneously. This means that the security of the repository needs to be guaranteed to be handled in the same way by both operating systems, otherwise one can act as a "back door" into the repository. Many sites do not have the same users and permissions (sourced from the same network register of users) on both Unix and Windows NT, making the mapping almost impossible even if the security models did actually correspond.

Most sites using Aegis and Windows NT together do so by running Aegis on the Unix systems, but building and testing on the NT systems. The work areas and repository are accessed via Samba or NFS.

LICENSE

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

It should be in the *LICENSE* file included in this distribution. The full test of the OpenSSL exception should be in the *LICENSE.openssl* file included in this distribution.

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

RELEASE NOTES

For excruciating detail, and also acknowledgments of those who generously sent me feedback, please see the *etc/CHANGES.** files included in this distribution.

Upgrading

In general, all the machines on your network need to be running the same release of Aegis. While the database format is backwards compatible, it is rarely forwards compatible in the face of new capabilities.

Version 4.26 (N-MMM-NNNN)

• Added OpenSSL license exception (LICENSE.openssl).

Version 4.25 (8-Mar-2008)

Version 4.24 (09-Mar-2008)

• Numerous portability improvements.

• Numerous improvements to the history reconstruction code.

• Numerous improvements and bug fixes to the distributed development code. See *aedist*(1), *aeget*(1), *aetar*(1) and *aepatch*(1) for more information.

• The site specific architecture information has been split into a separate file, maked with an entire-source-hide attribute, so that new Aegis-under-Aegis projects don't have such bad architecture problems.

• The license has been changed to GPLv3.

• A bug has been fixed the the change::pconf_get method which sometimes caused segfaults.

• A bug has been fixed in aeclone which caused aecp -delta to segfault when: a change set whith a removed file was cloned, and the clone change set subsequently integrated.

• The aediff command is now smarter about files which may have moved.

• A bug has been fixed in the aeannotate command, it no longer segfaults for some file histories.

• A Vietnamese error mesage translation has been added.

• There is a new Portugese (Brazillian) message catalogue translation.

• It is now possible to develop begin undo and new change undo in a single command.

• The *ael*(1) command now understands are much wider range of ways to specify changes.

• A segfault has been fixed in the use of --delta and --delta-data options.

• This change set fix a problem in the *aesvt*(1) checkout command that can fail to extract from a gzip compressed archive.

• A bug has been fixed in the *aechown*(1) command, to stop a segfault when printing some error messages.

• There is a new *aebisect*(1) command which helps to find project regressions not handled by the test suite.

• Many commands now cope with renames in more situations.

• A bug has been fixed in *aeclone*(1) which caused *aecp −delta* to segfault when a change set whith a removed file was cloned, and the clone change set subsequently integrated.

• The *aede-policy*(1) manual page has been updated to document the aede-policy-line-length file attribute.

• A vietnamese translation has been added.

• The *aereport*(1) command now understands more ways to specify changes.

• There is a new ${path_reduce} substitution which may be used to remove redundant elements from path lists, such as used by the $PATH environment variable.

• When the development directory style required actions to be perfomed on the development directory, the obsolete "creating symbolic links to baseline" message was produced. This was confusing. A more generic message is now used, which is intended to be less confusing.

• The aecp(1) -delta command now follows the whiteout preference when copying a "removed" file. • The test suite now runs much faster.

• A bug has been fixed in the *aedbu*(1) command. It no longer complains about permissions when the develop_begin_undo_command has been set.

• A bug has been fixed in the *aeimport*(1) command. It no longer uses the Attic portion of filenames when populating the history directory tree.

• The change details listing now prints comments in a wide column when the comments are lengthy.

• There is a new $Active_Directory substitution, used to obtain the development directory, or the integration directory, depending on the change state. This is rather like the default behaviour of the *aecd*(1) command.

• There is a new ${project version} substitution.

• The commands run by aeipass are now accompanied by more file name information, so that you can know which source file corresponds to which UUID history file, if there is a failure in the history commands.

• The *aeca*(1) command now checks for and discards duplicate architecture names. This fixes a bug with unsatifiable architecture dependencies.

• The *aeb*(1) command has been improved, it no longer keeps running the project_file_comand over and over again.

• A bug has been fixed in the *aenf*(1) command; it now preserves existing file contents if new files already exist in the development directory.

• A bug has been fixed in the "aet -regression" command, it no longer reports free()ing a non-existent string.

• A bug has been fixed in the *aed*(1) command, it no longer reports a bug when a cross branch merege is attempted for a file independently created in both branches.

Version 4.23

Version 4.22.2 (18-Oct-2007)

This is an update for the 4.22 stable release, it is meant to help Aegis users while the next release cycle ends.

• [1684820] Fixed a bug in aeclone that caused aecp −delta segfaults.

• The symlink farm now handle derived files registered within Aegis more like normal derived files.

• [1697199] The change_pconf_get function no longer looks for historical versions of files, if it can help it. This makes many things go faster and solved the problem of configuration fields redefinition. While this change does not make Aegis more time safe, it cures one of the symptoms.

• Fixed test 222 to work with recent releases of subversion.

• The *aepconf*(5) man page has been improved.

• [Debian 435422] The reference manual was wrongly referring to −Page-Headings instead of −Page-Header. The documentation has been updated to match the source code.

• [1704108] The *aecp*(1) −delta command now follows the whiteout preference when copying a "removed"file.

• [1704100] A bug has been fixed that caused *aecp*(1) −delta X to copy in a change also a file with the old name of a file *aemv*(1)ed before delta X.

• The generated Makefile now installs *aelock*(1) with the correct permissions.

• [1701701] A bug has been fixed in the *aetar*(1) command, it no longer creates tarballs that cause BSD and other tar to complain like this: tar: End of archive volume 1 reached tar: Unexpected EOF on archive file

• The configure script now handle correctly the datadir substitution.

Version 4.22.1 (14-Apr-2007)

• Test t0247a-walt.sh has been fixed, it was not exporting AEGIS_TEST_DIR. This make *aeintegratq*(1) leaving stuff in the home directory of the user.

• Some minor fix that prevented Aegis to build on RPM based distributions has been fixed.

• The t0011a.sh test script failed when *lex*(1) was missing, since it is not required to build Aegis the test script has been modified to pass even when *lex*(1) is missing.

• *aedist*(1) now handle certain renamed files correctly when receiving branches or entire-source.

• The t0011a.sh test script failed when *lex*(1) was missing, since it is not required to build Aegis the test script has been modified to pass even when *lex*(1) is missing.

• *aedist*(1) now handle certain renamed files correctly when receiving branches or entire-source.

• [1691122] Newer versions of the autoconf tools introduced a new @datarootdir@, and complained loudly if it wasn't used. Aegis configure does not trigger anymore those warnings.

• The test suite does not use anymore *diff*(1) −u because not all systems have gnu diff, so the use of gnu diff's −u option is nor portable, and will give false negatives on some systems.

• The test suite does not use anymore *diff*(1) −u because not all systems have gnu diff, so the use of gnu diff's −u option is nor portable, and will give false negatives on some systems.

• *aedist*(1) −rec now save the UUID as the user defined original-UUID if the UUID is already present in the repository. This is especially useful when receiving changes in the same repository.

• *aeclone*(1) now preserve the the UUID of the original change as the original-UUID user defined attribute of the new change. It also copy any other used-defined attribute.

• Test 89 has been disable on HP-UX-10 because that system has a "vendor specific" (i.e. broken) *cpio*(1) archive format.

• Test 95 has been improved to be less sensitive to *libmagic*(3) differences.

• Test 207 has been changed to be less sensitive to *sort*(1) differences.

• The project_specific setenv:* variables are now exported only once.

• [1674882] The following bug as been fixed: if a file is created and renamed within a single branch, and that branch is integrated, then the file is not included in the output of 'aedist −send −es' from subsequent branches.

• A bug has been fixed in the *aedbu*(1) command. It no longer complains about permissions when the *develop_begin_undo_command* has been set.

• The *aedist*(1) −rec command now better handles file renamed (not aemved) to match the local repository state.

• A bug has been fixed that caused the *change_pconf_get* function terminate *aegis*(1) with a fatal error if applied to a branch without a config file (e.g. if the trunk does not contain any closed branch).

• The *aenpr(1) −keep* command now set the administrator recursively.

• A bug has been fixed in the ${project-specific} substitution, it now works correctly with the *aesub*(1) −bl command.

• *aedist(1) −received* has been modified to set the user defined attribute foreign-copyright to true when receiving a remote change set. This in order to avoid *aede-policy*(1) complain about incorrect copyright notice at *aede*(1) time.0

• A bug has been fixed that caused an *aemv*(1) followed by an *aenf*(1) to generate two different files with the same UUID.

• Avoid the "multiple permission set" error on quit.

• A bug has been fixed in the UUID generating code; it was running out of file descriptors.

• A bug has been fixed in the *aet*(1) −regression command, it no longer reports free()ing a non-existent string.

• A bug has been fixed in the *aed*(1) command, it no longer reports a bug when a cross branch merge is attempted for a file independently created in both branches.

• A bug has been fixed which caused *aeipass*(1) to assign UUID to files at branch integration pass time. This can happen if the files was created and integrated with an old Aegis release, lacking support for file's UUID. This bug make it possible to have the history for a file split into two part, one accessible via the file_name, the other accessible using the UUID.

• A segfault in *aeannotate*(1) has been fixed.

• A bug has been fixed related to the use of the *unchanged_file_integrate_pass_policy=remove* policy described in *aepconf*(5). In this case aeipass failed to reset the locked_by field from the project fstate file, this prevented subsequent changes to modify the removed file.

• A bug has been fixed in the handling of the symlink farm, for development directory styles which use them for derived files. Derived files in the baseline directory which were formerly source files, but then aerm-ed, are now included in the development directory when copy/link styles are used.

• A bug has been fixed in the *aenf*(1) command; it now preserves existing file contents if new files already exist in the file development directory.

• The ./configure script has been improved to stop with a fatal error if the bzip2 library is not available.

Version 4.22 (29-Mar-2006)

• A bug has been fixed in the *aeclean*(1) command, it now correctly resets the change build and test times.

• A bug has been fixed in writing of tar and cpio data, in cases where there was one byte too much padding.

• A bug has been fixed in the *aeintegratq*(1) command, it no longer ignores change number zero.

• A bug has been fixed in the *aepromptcmd*(1) comand, it now understands that when the build command is "exit 0" then no build is required.

• The *aede*(1) comand now runs the *review_pass_notify_command* (instead of the *develop_end_notify_command*) for projects configured to skip the *being reviewed* state.

• A bug has been fixed in the *aeannotate*(1) command, it no longer uses the wrong timestamp when creating histories for completed branches.

• A bug in the *aed*(1) command has been fixed, it no longer reports a bug when trying to merge a file that has been renamed.

• A bug has been fixed in the *aet*(1) command, it now correctly handles multiple architectures being reported for batch test results.

• A bug has been fixed in the *aet −regression* command, the *batch_test_command* now correctly handles multiple architectures in the results.

• The notification scripts distributed with Aegis have been fixed, they now correctly substitute recipients' email addresses.

• A bug has been fixed in the *aediff*(1) command, the *−change* option is now ablew to cope with degenerate forms of the delta name in cases like *aediff −change D001* and similar.

• A bug has been fixed in the *aenc*(1) command, it now takes more notice of project testing default settings.

• A bug has been fixed in the *aeget*(1) interface, the adjective for the alternate listing link at the bottom of the Integration Histogram pages has been inverted.

• A bug has been fixed in the *aeget*(1) command, is is now always possible to see the error produced by a script when the *noerror* modifier is specified.

• A bug has been fixed in the *aeget*(1) web interface, it now provides the correct links to the more and less detailed file history pages.

• The *aeget*(1) web interface no longer emits broken links to removed source files.

• A bug has been fixed in the *aenbr*(1) command, the *protect_database* project attribute is now correctly inherited from the parent branch.

• A bug has been fixed the the RSS feed, where HTML special characters were not rendered correctly.

• A bug has been fixed in the *aeipass*(1) command, it no longer fails if the *history_create_command* was not set, it uses the *history_put_command* instead, as it is supposed to.

• A bug has been fixed in the *aedist −send* command, it no longer attempts to include the source of removed files.

• A bug has been fixed in the *aedist*(1) command, it no longer segfaults when compiled with DEBUG defined.

• A bug has been fixed in the *aedist −replay* command, it no longer downloads change sets more than once.

• A bug has been fixed in the *aedist −send* command, it no longer obtains the wrong version of the project files when building patches for files which have been renamed.

• A bug has been fixed in the *aedist*(1) command, no longer attempts to include the source of removed files.

• A bug has been fixed in the *aedist −pending* command, it now resolves project aliases.

• A bug has been fixed in the *aedist*(1) command, it no longer segfaults on IRIX.

• A bug was fixed which caused the development_directory of a branch to be recorded as an absolute path in the Aegis meta-data, rather than relative to the home of the project. This problem make it difficult to move a project to a different location in the filesystem.

• There is a new open source project example on the web site, which allows tarballs to be unpacked and turned into an Aegis project in less than 30 minutes.

• There is a new *aefinish*(1) command which may be used to read the state of a change set and then run all of the Aegis commands necessary to to end development. See *aefinish*(1) for more information.

• The *aexml*(1) command now understands ".bz" output file suffix, in addition to the ".gz" suffix it already understood. The man page has been updated to cover the −output opion.

• The *aerevml −send* command is now able to produce bzip2 compressed output.

• The restrictions on project alias names have been eased. It is now possible to have any alias name you like, so long as it doesn't contain any shell special characters.

• It is now possible to set change attributes from the command line, without going via an editor. See *aeca*(1) for more information.

• The *aetar −send* command is now able to produce bzip2 compressed output.

• There is an new *aetar −exclude* command line option, allowing you to exclude files from the tarball being unpacked and used to for the change set. This is typically necessary when a tarball includes derived files (e.g. the ./configure script in most open source projects).

• There is a new *aetar −exclude-auto-tools* option, which can be used to exclude derived files commonly found in open source projects using the GNU Autoconf and GNU Automake tools.

• There is a new *aede-policy*(1) command which may be invoked by *develop_end_policy_command* to enforce additional local policies. See *aede-policy*(1) for more information.

• When symlinking files (source or derived) into the development directory, the last-modified time of the link is set to the last-modified time of the file being linked to, when the underlying filesystem supports it.

• The aefa(1) command now accepts *name=value* attribute assignments on the command line.

• The *aet*(1) command now understands *name=value* pairs on the command line, and passes them unchanged to the test command. The −force option implies a `force=1` variable setting.

• The *aepatch −send* command is now able to produce bzip2 compressed output.

• The *aesvt*(1) command now uses the *bzip2*(1) algorithm by default. There is a *aesvt −compression-algorithm=gzip* option for forwards compatibility.

• There is a new *ae-repo-ci*(1) command which may be used in an *integrate_pass_notify_command* to do a parallel check-in of a change set into a second parallel repository. It understands CVS and SVN at the moment; it is easy to extend to understand more repository types. The old *ae-cvs-ci*(1) script now invokes

the *ae-repo-ci*(1) command.

• The build step of the development process can now be made optional. Configuring a *build_command* of "exit 0" will tell Aegis your project does not need to be built.

• The *aedist −replay* command now adds a compatibility modifier to all of the downloads URLs, so that the change set received will be compatible with the version of aedist at the receiving end.

• The *aedist −send* command now accepts a *−no-mime-header* option, to make it easier to validate the *aedist*(1) output against the real *cpio*(1) command.

• The *aedist −send* command is now able to produce bzip2 compressed output.

• There is a new *entire-source-hide* file attribute which may be used to omit site-specific files from *aedist −send* change sets.

• The *aetar −remove-path-prefix* option now also accepts a numeric argument.

• The *aeannotate*(1) command now understands the *−change* and *−delta* options.

• The *aedb*(1) command has been enhanced to check that directory permissions above the development directory will be traversable by the integrator and the reviewers.

• The *aecpu*(1) comand now understands the *−read-only* option to mean uncopy all of the insulation files.

• There is a new *aelock*(1) command, which may be used to take read-only locks. This can be useful for backups, and other activities outside Aegis' scope which require a constant project state to operate correctly.

• The *aedist* command can now perform file merges with better results.

• The *aedist −receive* command now looks to see if the executing user has project admin priviledges, and if so does not cancel testing exemptions.

• The *aedist −receive* command now applies patches using the *patch*(1) command, rather than doing it less well itself.

• The *aedist −replay* command now attempts to use the same change number as on the remote system. A bug has been fixed in the way it looked for change numbers.

• There is a new *unchanged_file_integrate_pass_policy* field in the project configuration file, which controls what to do when a change set contains an unchanged file at integrate pass time.

• It is now possible for developers to edit a change description when a change is in the awaiting development state, if the project has *developers_may_create_changes* enabled.

• The *aed*(1) command is now optional. Configuring a *diff_command* of "exit 0" will tell Aegis your project does not need to be differenced.

• The *aeget*(1) interface now places HTML anchors in description text where it recognizes them.

• There is a new *aeget:inventory:hide* change attribute, which may be used to prevent strictly local change sets from being advertised in the *aeget*(1) change set inventory.

• The *aeget*(1) web interface file listings pages now link the edit numbers to file versions. When history is available there are also links to the previous verion, and the arrow is linked to a diff page.

• The *aeget*(1) presentation of file history has been improved to highlight renaming of files.

• The *aeget*(1) web interface now has a recursive option on its project integration history pages.

• The *aebuffy*(1) command is now able to run the *tkaer*(1) command from more states, and it now accepts 'q' to quit. The display of changes with double quotes (") in their brief description has been improved.

• A build problem with libcurl not being present has been fixed.

• A bug has been fixed which caused errors when Aegis was compiled with g++ 4.1

• A build problem has been fixed on Solaris.

• A build problem related to *bison*(1) using *libintl*(3) has been fixed.

• The `./configure` script has been improved to correctly detect installation of the OSSP UUID library.

• A build problem on HP/UX has been fixed.

• A build problem on MacOS X has been fixed.

• A build problem has been fixed where libraries required by the `./configure` script are located under `/usr/local/lib` or some other non-standard place.

Version 4.21 (10-Nov-2005)

You must have the *Gnome libxml2* library (`http://xmlsoft.org/`) installed in order to build Aegis. Please install the xml2 library version 1.8.17 or later. You do not have to install the rest of Gnome, the library can be used on its own. If you are using a package based install, you will need the *libxml2-devel* or *libxml2-dev* package in addition to the *libxml2* package.

Ideally, you would also install the *libmagic* package, used to determine file types, just as *file*(1) does. (This is not to be confused with the `libmagic6` image manipluation library. If you are using a package based install, you will need the *libmagic-devel* or *libmagic-dev* package in addition to the *libmagic* package.

• A bug has been fixed in the *aecp −independent −output* option, which resulted in an error when Aegis tried to chmod nothing.

• The auto file promote feature previously available in *aed*(1) has been added to the *aeb*(1), *aecp*(1), *aerm*(1) and *aenf*(1) commands.

• The *aedist −pending* and *aedist −missing* commands now print the number of changes in the remote inventory.

• A bug was fixed in the *aecp* command which caused a segfault sometimes when the user tries to copy a removed file.

• The *aedist −replay* command now accepts a *−maximum* option, which includes change sets not yet completed in the local change set inventory when considering what to download.

• There is a new *develop_end_policy_command* field in the project configuration file. It can be used to add addition constrains to change sets before they can complete *aede*(1) successfully.

• The *aedist −receive* command now annotates remote change sets (typically, change sets downloaded via the *aedist −replay* command) with their origin URL.

• A bug has been fixed in the *aebuffy* command where it would display incorrectly when the brief_description of a change contained double quotes.

• It is now possible to attach a comment to all commands which involve a change state transition, e.g. *aenc*, *aede*, etc. This is done using the **−reason** command line option, just as you are able to do for review fail, etc.

• A bug has been fixed in aenc, where it did not correctly copy user defined attributes.

• There is a new *aelcf*(1) command to efficiently generate lists of change source files for use by your build tool.

• There is a new *aelpf*(1) command to efficiently generate lists of project source files for use by your build tool.

• There is a new cache of state information attached to each delta, the project file state at the time of the delta. This has the potential to accelerate aecp −delta, and all other project_file_roll_forward-based operations. Large projects may want to turn this off, because each delta will produce another large project file state cache.

• There is support for generating RSS feeds from Aegis. See the Aegis project pages on the Aegis web interface for an example. See *aepconf*(5) and *aeget*(1) for more information.

• The ${change delta_uuid} substitution now allows access to the delta_uuid in the *being integrated* state.

• The "wrong file" error message from aedist has been improved, to say what was expected.

• There is a new optional *$filename* substitution for the *history_put_command*, so that you can attach the

current name of the check-in to the history file meta-data. There is a new optional *$uuid* substitution for history_put_command, so you can attach that as mete-data, too.

• There is a new history tool bundled with Aegis. See *aesvt*(1) for more information.

• There is a new *default_regression_test_exempt* project attribute.

• The *aedist −receive* delta selection mechanism has been improved: previously the edit-origin-UUID attribute was considered in favour of the original-UUID attribute, with this change it is used the change set, bounded to the edit-origin-UUID or to original-UUID, more recently integrated. This should reduce the frequency of logical conflicts.

• There is a new *aerevml*(1) command, which can be used to send change sets in the RevML format. See *aerevml*(1) for more information. The *aeget*(1) web interface is also able to serve change sets in this format.

• A problem has been fixed which caused Aegis to fail on the hppa port of Debian.

• The *aetar −receive* program now uses the archive name as the brief description.

• A bug was fixed in *aedist −send* which caused segfaults when processing some files.

• A bug was fixed which caused *aedist −send* to produce an archive that can not be *aedist −receive* because of an operation impossible to replicate in a change set.

• There is an implementation of Robert Collins' subunit testing framework available. See *aesubunit*(1) for more information.

• A bug was fixed in *aedist* that caused an error when receiving a branch's archive generated with the *aedist −send −entire-source* option.

• A bug has been fixed in *aedist −receive* that caused a segfault in the rename handling code.

• The *aedist −missing* listing (and the *aedist −replay* behaviour) now check for branch UUIDs as well, just in case someone fetched a branch as a change set and applied it. However, *aeget* does not report these UUIDs, because that would be too confusing.

• It is now possible to specify any sufficiently unique leading prefix of a UUID rather than the full 36 characters.

• There is a new *${History_Path}* substitution available. It gives you the path name of the history file corresponding to the given filenames.

• A bug in *aedist −receive* which caused incorrect delta selection has been fixed.

• There is a new *aedist −pending* option which can print the list of local change sets missing from a remote repository.

• The *aedist −receive* command is now able to use the edit-origin-UUID attribute to copy modified files from the right origin.

• A bug has been fixed in *aedist −send* where some types of incomplete changes would fail an assert.

• There is a new *aexver*(1) command which can be used to view historical versions of files in an Aegis repository. See *aexver*(1) for more information.

• A bug as been fixed which caused *aemv*(1) to incorrectly rename a file to an existing directory

• It is now possible to specify user-defined user attributes in the `~/.aegisrc` file.

• The *aenf*(1) command now gives a warning if you specify the "config" file without the "−config" option. This is the old name for the project configuration file, the new name is "aegis.conf".

• The *aefind*(1) command now understands `{+}` to mean the resolved file name, and `{−}` as the unresolved file name.

• There was a bug where Aegis would exit with a fatal error if one of the directories on the AEGIS_PATH was read-only. Such directories are now ignored.

• The *aetar*(1) command has been improved to process modified and created files in a batched way; this improves the speed.

• Additional explanatory text has been added to the message printed when error message translation files can't be found.

• The *aenf*(1) command now understands the −*keep* and −*no-keep* options, to explicitly control the creation of new files in the development directory.

• A bug has been fixed in *aemv*(1) which failed to check the new name against the filename charset, *etc*.

Version 4.20 (28-Jan-2005)

Please Note: Users are advised to check the history command settings in their project configuration files. With the advent of file UUIDs, the history mechanism now decouples source file names from history file names. In particular, the assumption that the history file basename is the same as the source file basename is no longer true. Correct settings may be found in the *lib/config.example/* directory of the source distribution.

• The defaulting rules for the change number (if none was specified on the command line) have been altered. the current directory now takes precedence over the "only one" rule. This seems to meet user expectations better.

• A bug has been fixed in the *aecvsserver*(1) command which caused to to fail when accessed by some clients.

• A bug has been fixed which caused many of the programs to leave temporary files behind.

• A bug has been fixed in the *aedist* −*send* −*entire-source* command where it would hang for some cases. (Actually, it would dump core after using up all available swap space on an infinite recursion).

• A bug has been fixed in the *aedist* command (and other places) where the open of the project configuration file could fail, due to not properly reconstructing in historical circumstances.

• A bug has been fixed in the integration build which was removing files it should not, for *during_build_only = true* work area styles.

• The *aeb* command now complains *much* less about "directory not empty" when using the link farm.

• A bug has been fixed in the *aetar* −*send* −*entire-source* command where some files were missing when asking for a complete set of historical sources.

• A bug has been fixed in the *aedist* −*send* −*entire-source* where some files were zero length when asking for a complete set of historical sources.

• A bug in *aedist*(1) has been fixed, it was forcing regression test on the receiving side even if the change set does not require it and *default_test_exemption* was set to true. It was annoying especially if the test suite take a long time to run completely.

• Some bugs have been fixed in *aediff*(1) which caused it to mis-parse the command line in some cases, and it was also barfing on the expected exit status 1 when an actual difference was found.

• A bug has been fixed in the *aecpu*(1), *aemtu*(1), *aemvu*(1), *aenfu*(1), *aentu*(1) and *aermu*(1) commands. They were not repairing the symlinks (*etc*) required by the *development_directory_style* settings.

• A bug has been fixed in the *Change_Files* listing; it was not showing the locked-by information.

• A bug has been fixed in the code which updates the development directory symlinks. It was failing to make all the directories required.

• A bug has been fixed in the *aedist* −*send* −*entrie-source* command, where it would segfault in some cases.

• A bug has been fixed in reading plain *diff*(1) format patches. This was particularly obvious because *aeannotate*(1) uses this form of diff by default.

• A bug has been fixed in *aeget*(1) where it was showing removed source files as available for download.

• A bug has been fixed in *aeget*(1) where it produced invalid output if the SCRIPT_NAME environment

was not set.

• A bug has been fixed in *aeget*(1) where it would sometimes ignore modifiers. This was particularly noticable in the download pages.

• A bug has been fixed in the *aeimport*(1) command. It was using the old work area style configuration file parameters, instead of the new *development_directory_style* settings.

• A memory leak has been fixed in the symbol table code.

• A bug has been fixed in the project_file_find_by_uuid function. In some cases it would SEGFAULT, particularly once the memory leak in the symbol table code was fixed.

• Several build problems have been fixed.

• The *aeintegratq*(1) command has a new −loop *options, which causes it to keep processing changes that become available while it is running.*

• The *aet*(1) command has a new −sugest-limit option which runs as many regression tests as possible (from most relevant to least relevant) but stops after the given number of minutes. This is a way for running the most relevant tests in a limited time. For example, this option could be used if a project has so many integrations in a day that it can only afford 20 minutes of integration testing for each one.

• The *aed*(1) man page has been updated to better describe the behaviour around the merge command.

• The *aetar −send* command now accepts an −include-build option that also add build files, registered with *aegis −new-file −build*, to the ouput archive. A −not-include-build option is also accepted.

• The *aetar −receive* command now avoids copying build files from the baseline because this operation is forbidden and the error stops the processing.

• There is a new *${Change_Attribute}* substitution, which is replaced by the values of the change attributes named.

• The history recapitulation code (project_file_roll_forward) now indexes by UUID rather than by file name (with backwards compatibility for UUID-less repositories). The user visable result is that file history reports and listings now accurately track renames.

• The *aet −nopersevere* option now also stops for *no result* as well as *fail*.

• The aedist −receive command now understands file UUIDs. This means that it will operate on the correct file even when one or the other repository has renamed the file.

• The *aedist −receive* command has been enhanced to perform file merges if necessary.

• There is a new *aedist −replay* option. When used in with an *aeget*(1) server, it can be used to synchronize two repositories. The *aedist −missing* option may be used to show what would be downloaded.

• The *aefa*(1) command, with the −edit option, now shows you the content type, rather than adding it silently.

• There is a new *aediff −command* option, allowing you to specify the command you want to use to display the difference. For example, you could use *tkdiff*(1) or *mgdiff*(1) to display the change graphically.

• The *aediff*(1) command now adds labels when it is producing a context or unified diff output.

• There is a new optional *review_policy_command* field in the project confioguration file. This allows for customised review policies for each project, including multiple reviewers and specific reviewers for portions of the sources.

• There is a new ${Change_Reviewer_List} substitution, which is replaced by a space separated list of reviewers of the current change, since the last develop end. This is of particular use to the *review_policy_command* field of the project configuartion file.

• There is a new ${Change_Developer_List} substitution, which is replaced by a space separated list of all the developers of the current change.

• There is a new ${quoted_email_address} substitution, which replaces it arguments with the email

addresses of the names users. See *aesub*(5) for more information.

• The notification scripts have been updated to use the new ${quoted-email-address} substitution.

• The remaining *aegis.cgi*(1) functions have been reproduced in *aeget*(1). The *aegis.cgi*(1) script is now deprecated.

• When the UUID of a change is cleared it (because some operation on the change set invalidates it) is saved in a change attribute named *original-UUID*.

• The *aedist −receive* command is now able to use the original-UUID attribute of the incoming change set to select the delta to merge with.

• The "path unrelated" error message has been updated to make it more informative.

• All attribute names (project, change and file) are now case-insensitive.

• The *aedist −receive* command has been enhanced to allow you to select the branch of the delta to merge with.

• Several classes within the source have been refactored.

Version 4.19 (30-Sep-2004)

Please Note: Users are advised to check the history command settings in their project configuration files. With the advent of file UUIDs, the history mechanism now decouples source file names from history file names. In particular, the assumption that the history file basename is the same as the source file basename is no longer true. Correct settings may be found in the *lib/config.example/* directory of the source distribution.

• There is a new *development_directory_style* field of the project configuration file. This allows CVS-style and Arch-style work areas, in addition to the BCS-style and viewpath work areas already supported. These new work area styles permit many existing projects to use Aegis with no change to their build systems. The `libsndfile` and `OpenLDAP` projects, for example, have been imported and built without modification. See *aepconf*(5) and the *Dependency Maintenance Tool* chapter of the User Guide for more information.

• There is a new *aediff*(1) command, which may be used to obtain a *diff*(1) listing of a file for different deltas.

• There is a new *aepromptcmd*(1) command, used with bash's PROMPT_COMMAND environment variable. It can be used to obtain a colored prompt, simulating the process described in Kent Beck's book *Test Driven Development*.

• There is a new *signed_off_by* field of the project configuration file. Set it to true if you want "`Signed-off-by`" lines appended to change set descriptions as the changes pass through the Aegis process. The *aede*(1) and *aerpass*(1) commands now understand two new **−signed-off-by** and **−no-signed-off-by** options, to override the project setting. The *aedist −send* and *aepatch −send* commands also understand the new **−signed-off-by** option, to add the "`Signed-off-by`" line to the outgoing change set description. Conforming to: http://www.ussg.iu.edu/hypermail/linux/kernel/0405.2/1301.html and http://www.osdl.org/newsroom/press_releases/2004/2004_05_24_dco.html

• The *aet*(1) command has been enhanced to allow integrators to run specific tests.

• The *aesub*(1) command can now read the text to be substituted from a file or standard input.

• It is now possible to use the project-specific attributes to specify environment variables to be set for commands executed by Aegis. This can be used to set a predictable PATH, for example.

• It is now possible to customize the *aeget*(1) web interface using project specific attributes.

• The *ael*(1) command and the *aeget*(1) web interface now have file inventory pages, for the project file inventory and the change file inventory.

• There is a new "change set inventory" listing available via the *ael*(1) command and the *aeget*(1) web interface, which lists changes and their corresponding UUIDs, and links to an aedist download for each change. The idea is that the *aeget*(1) pages may be used to automate downloading change set your repository does not yet have.

• There are two new history commands in the project configuration file, the history_transaction_begin and history_transaction_end fields. It is not an error if these fields are absent. If you need a transaction key, use the $version substitution.

• The *aedist*(1) command now runs all tests required for the change set, and honors test exemptions.

• The *aedist*(1) command now sleeps for a second to ensure that the last-time-modified of derived files will be strictly later than source files, and that the *aeb*(1) timestamp will also be strictly later then the last-time-modified for the source files.

• The tkdiff(1) man page has been updated to say how to use mgdiff(1) instead of tkdiff(1).

• All commands which accept the **−change** option may now be given a change set UUID. You can discover a change's UUID using the *ael cd* (list change details) or *ael inventory* listings.

• The *aed*(1) command now restores source file from backups (,B) when a merge fails. Previously this was not the case and subsquent aed invocations failed because the source file was missing.

• The *aetar −send* command now has an **−add-path-prefix** option, so that you can add a path prefix to all of the files in a tarball. The *aeget*(1) CGI interface now adds a path prefix to generated tarballs by default.

• Whenever you edit file attributes, there is a `Content-Type` attribute added automagically if none was there already. The idea is that this could be used by scripts to differentiate file types.

• The *aepatch*(1) command now uses *diff −u* by default.

• A number of build problems on different systems have been fixed.

• A number of minor problems with tests on different systems have been fixed.

• A bug has been fixed in the aepatch command; it was not parsing simple diff patches correctly.

• The example history commands have been updated to work better with the new UUID code.

• A bug has been fixed in aecp −delta, where it would fetch the wrong version of a file in some cases.

• A bug has been fixed in the handling of the executable bit.

• A bug has been fixed in *aede*(1), where is did not permit branches to end when they had a removed file (without a UUID) which has been subsequently recreated (with a UUID).

• A bug has been fixed in the *aeget*(1) command for file contents. It was giving a "multiple permissions set (bug)" error message.

• A bug in the *aedist −receive* command, where it was not accurately manipulating the incoming change set UUID.

• A bug has been fixed in *aed*(1) which caused it to SEGFAULT.

• A bug has been fixed in the *aede*(1) command, where it failed to copy the UUID when it promoted a file from "create" to "modify" automatically.

• A bug has been fixed in the `$date` substitution, it was not advancing properly when used in progress messages.

• A bug has been fixed in the command line processing of the *aefa*(1) command.

• A bug has been fixed in the *aegis −review-begin* command; it was not operating correctly when the change was in *awaiting_review* but the project was in *goto_being_reviewed*.

• A bug has been fixed in the `$basename` substitution; it now functions exactly like *basename*(1) command.

• A bug has been fixed in the *aet −bl* command; it erroneously stated that the `$Search_Path_Executable` substitution was mandatory, when it actually optional.

Version 4.18 (10-Jun-2004)

• A number of build problems have been fixed, particularly concerning GCC 3.3 and later.

• The *aemv*(1) command has been enhanced to accept more than two file names. You are now able to move serveral files in the one command.

• The *aedist −receive* command has been enhanced to process move operations in a batched way. This improved performance when receiving a change that renames many files.

• The `./configure` script has been changed to take note of the `--sysconfdir` option, used to specify the location of the /etc directory.

• A bug has been fixed in the *aepatch*(1) command. It would SEGFAULT when a non-source file was patched.

• A bug has been fixed in the *aemeasure*(1) command. It would SEGFAULT when no files were named on the command line.

• The Russian error message catalogue has been updated.

• A subtle bug in the change file out-of-date tests have been fixed. It did not adequately address the transition case for projects containing files with and without UUIDs.

• The `./configure --sysconfdir` option is now honored. It is *very* important to set it to */etc* when you configure Aegis.

Version 4.17 (3-Jun-2004)

• Each new change set is now assigned a Universally Unique Identifier (UUID) to allow it to be tracked across geographically distributed development. The *aedist*(1) and *aepatch*(1) commands now send the change set UUIDs, and preserve them on receipt.

• Each file now has a Universally Unique Identifier (UUID) which allows tracking files across renames, even on geographically separate sites. (The *aedist*(1) and *aepatch*(1) commands send the file UUIDs, the next release will take advantage of them on receipt.)

• The history filename used to remember file history is now based in the file UUID, if the file has a UUID. This simplifies continuity of history across renames (this fucntion always been present in Aegis, but harder to access).

• As a consequence of the UUID being used to generate history file names, there is no longer the restriction that new files may not be named after the directory portion of a deleted file (or vice versa).

• There is a new *aecvsserver*(1) command, which presents Aegis projects and change sets as CVS modules. All of the core CVS functions are supported. This code needs to be exersized and tested by users.

• It is now possible to specify arbitrary attribute names and value for each source file. The *aefa*(1) command may be used to edit file attributes. The *aedist*(1) and *aepatch*(1) send these attributes; a future release will take advantage of the information on receipt.

• It is now possible to attach arbitrary attribute names and values to change sets. For example, you can use this to remember the bugzilla tracking number for a change.

• The *aepatch*(1) command now includes change set meta-data as a compressed BASE64 encoded block at the top of the patch, after the human-readable text but before the files. This means that *aepatch*(1) can be as effective as *aedist*(1) is transmitting chaneg sets. Patches without meta-data still work as before.

• There is a new report script which writes change logs in Debian format.

• The *aeget*(1) web interface has been improved. The *aepatch*(1) download now accepts `compat=`*N* modifier, and there is a new Project Staff page.

• There is a new *ae-cvs-ci*(1) support script which may be used as an *integrate_pass_notify_command* to commit change sets to CVS in parallel.

• There is more documentation in the User Guide about using GNU Diff, particularly using diff −U to provide whole-file listings with "change bars" on the left hand side.

• The files view of the *aeget*(1) web interface now accepts options to control the page contents. The

simplest view allows recursive fetch of project sources using wget or similar, with no extraneous links to confuse the results. Previous behavior is preserved by the aeget-generated links.

• You now receive a warning when you are seeing the short version of the error messages. These are terse and often quite cryptic. the long form of the error messages is to be preferred.

• The behaviour of the *aedeu*(1) command has changed slightly. When changes are in the *being reviewed* state, and Aegis has been configured to use the *awaiting review* state, the *aedeu*(1) command will now report an error. This is so that reviewers don't waste their time reviewing changes which have already been returned to the *being developed* state. Think of the change as "belonging" to the reviewer while in the *being reviewed* state.

• The *aedist −send* command has a new *−compatibility* option, use to indicate the version of the **receiving** aedist program. This, in turn, selects the features which may be added to or omitted form the generated `.ae` file.

• There is a new `config` file usage, and a corresponding *aenf −configure* command line option. It is now possible to move project configuration files. It is now possible to remove project configuration files, provided there is at least one left. The *aeimport*(1) command now avoids files which have the same name as the default project configuration file ("aegis.conf" or "config") and will use something else.

• The *aeipass*(1) command now adds a symlink from the delta directory to the baseline once it has been integrated. This helps lots of (idiotic) compilers which insist on burying absolute paths into executables.

• It is now possible to assign to some project configuration file array fields more than once. This can be useful where the configuration file is split into several pieces on several branches.

• The source language has been changed from C to C++. Future releases will take advantage of this.

• Several bugs have been fixed in the *aeget*(1) web interface where it would display "−42" instead of "0" for changes and branches numbered zero.

• A bug has been fixed in the *aed*(1) command when merging files which have been renamed. It now recognises they need merging.

• A bug has been fixed in the *aenf*(1) command. It now correctly ignores difference files when given a directory name.

• A bug has been fixed in *aedist*(1) where one of the *aegis −new-file* commands was missing a *−no-template* argument. Under some circumstances, this resulted in change sets which could not be *aedist −receive*d.

• A bug has been fixed in the way invalid sequences of multi-byte characters are handled by the internationalization code. This potentially affected all reports, listings and error messages. The symptom was that aeannotate listings could sometimes have a blank source code column.

• A bug has been fixed in the *aepatch*(1) command. It was creating empty patches for some changes in completed project branches. This also affected *aedist −send* and *aecp −delta* and *aecp −rescind* in some cases. It was caused by a subtle flaw in the non-detailed case for the *project_file_roll_forward* function.

• A bug has been fixed in the handling of the MANPATH enviromnet variable by the *profile* and *cshrc* scripts.

• A bug has been fixed in the *aedist −receive* and *aepatch −receive* commands has been fixed. There were cases where these commands could access off the end of an array and SEGFAULT.

• A bug has been fixed in the *aede*(1) command when it received pre-config-usage change sets. It used to try to remove the last project configuration file, which is a fatal error, and made it impossible to receive the change set.

• A bug has been fixed which caused the *aetar*(1) command to hang (actually, any thing which consulted LDAP or NIS) because the reserved symbol "send" was being overloaded. The reserved symbol "clone" was also being overloaded. Both have been fixed.

• A bug has been fixed which caused the *aedist*(1) command to reprt the wrong error when the input file did

not exist.

• A bug has been fixed in the *aenbru*(1) command which made project aliases disappear.

• A bug has been fixed in the *aede* command. It would fail with new build files already in the baseline.

Version 4.16 (14-Jan-2004)

• There is a new *aecp −keep* option, causing *aecp*(1) **not** to overwrite file contents in the development directory.

• The *aedist −receive* option now understands changing the type of a file.

• It is now possible to specify a URL to the **−file** option on the command line of *aedist*(1), *aepatch*(1) and *aetar*(1). The data will be downloaded and applied.

• More work has been done towards making the code compilable by a C++ compiler.

• The project list (see *ael*(1), *aeget*(1), etc) is now sorted in a slightly more natural way, as are the version statistics at the end of an *aeannotate*(1) listing.

• A bug has been fixed in *aede*(1) for branches, where Aegis would complain about build source files (created by the *aenf −build* command) being out-of-date. This, of course, was difficult or impossible to fix, and unnecessary because the next build would fix them.

• A bug has been fixed in the *aecp −independent* command, where it did not preserve the execute bit, nor honour the user's umask.

• The missing *aemt* and *aemtu* alias ve been added to the profile.

• More detail has been added to *aepconf*(5) detailing how to create the project configuration file for the first time.

• A bug has been fixed in *aedist*(1) and *aepatch*(1) which would cause an assertion failure (or segfault) when you tried to *aedist −send −delta −es* files which did not exist at that delta.

• A bug has been fixed in *aedist*(1) and *aepatch*(1) which caused an assert failure (or segfault) when you tried to send a file which had been created and removed in a branch, and after the branch was integrated only a remove record exists in the parent branch.

• The problem with test 134 failing has been fixed.

• A bug has been fixed in *aeipass*(1) which prevented changing a file's usage from being as straight-forward as it should have been.

• The source RPM (and the *spec* file) now has Build Prequisites specified.

• The **−Change** option now accepts more than just a change number. It now accepts many forms similar to those used by the *${version}* substitution, allowing its output to be used directly as command line input; forms such as `−c 1.2.C34` and `−c=5.6.D78` are now understood to imply a **−branch** option as well as either **−change** or **−change-from-delta**, respectively. In addition, you may prepend a project name, to imply the **−project** option as well; form such as `−c aegis.4.15.C28` are understood.

• The *aemeasure*(1) program now also generates Halstead metrics.

• A bug has been fixed in the symbolic link handling code. In some cases it would report "multiple user permissions (bug)" and not complete correctly.

• A bug has been fixed in the test of *aedist*(1) for moved files. There was nothing wrong with *aedist*(1), the test itself was broken.

Version 4.15 (17-Nov-2003)

• A bug has been fixed in “ael cf”. It used to fail an assertion when there were no files in the change.

• A bug has been fixed which caused aeipass to segfault when adjusting file modification time stamps in some circumstances.

• A bug has been fixed in the cross branch merging code. It would sometime erroneously complain about files no longer being in the baseline.

• A bug has been fixed which caused *aedist*(1) and *aeannotate*(1) to segfault. It was caused by the roll forward history mechanism ignoring some branches in some cases.

• A bugs has been fixed in the *aenrv −Descend_Project_Tree* option, which was free()ing a project twice, sometimes causing segfaults.

• The *aeget* CGI interface is now able to retrieve historical versions of files.

• The *aeget* CGI interface now has support for file metrics.

• The *aeget* CGI interface has been enhanced to provide more information about project files and change files: activity, conflicts, history.

• The *aeipass*(1) command now sets the AEGIS_INTEGRATION_DIRECTORY environment variable before running the *integrate_pass_notify_command*, so that you can add a symlink for compilers which insist on placing absolute paths into debugging information in object files.

• The *aeget* CGI interface has been enhanced to provide more information about project files and change files − activity, conflicts, history.

• The *aeget* CGI interface now reports more project information.

• The *aeget* CGI interface now has download links in many of its menus, allowing more and better downloads than the old *aegis.cgi*(1) script.

• The way *aenf*(1) and *aent*(1) work have been made more generous. It is now possible to *aerm*(1) a file and then *aenf*(1) or *aent*(1) the same file in the same change. This is useful for changing the type of a file. Previously this has to be done as two consecutive changes.

• The *aecp −independent* command has been enhanced to allow you to extract versions of built files (created with *aenf −build* and maintained at *aeipass*(1) time).

• Documentation has been added to *aer*(5) for the try/catch mechanism.

• There was a disagreement between the *aereport*(1), *aeannotate*(1), *aedist*(1), *aefind*(1), *aeimport*(1), *aels*(1), *aepatch*(1), *aerect*(1), *aetar*(1) and *aexml*(1) man pages and the commands themselves about the existence of the **−version** option. The commands now behave as documented.

• There is a new *Project_Branch_Dates* report, which may be used to see when branches of a project were begun and completed.

Version 4.12 (29-Sep-2003)

• A bug has been fixed in *aedist*(1) where it handled moved files incorrectly.

• There is a new experimental *aeget*(1) program. It is a potentially faster, potentially more capable replacement for the *aegis.cgi*(1) script. At the moment it isn't, it's experimental.

• A bug has been fixed in *aedist*(1) where it would sometimes segfault when sending transparent files.

• Command completion now works for the *aemt*(1) and *aemtu*(1) commands.

• A bug has been fixed where the symbolic link farm could point to the wrong place when change files are transparent.

• Change file notification commands have been added for the *aemt*(1) and *aemtu*(1) commands. See *aepconf*(5) for more information.

• A bug has been fixed in *aefind*(1) command where it could report files which had been removed.

• A bug has been fixed in the *aecp*(1) command where it would scramble the *aet −reg* exemption.

• A bug has been fixed in the *aede*(1) command. The problem manifested as an *aet −reg* command which terminated early.

• There is a new *aexml*(1) command. You can now obtain various pieces of the Aegis database as XML. See *aexml*(1) for more information.

• The the *new_file_command*, *copy_file_command* and *remove_file_command* fields of the project *config* file are now defaulted correctly.

• Theer is a new *$change_files* substitution. See *aesub*(5) for more information.

• The project config file has a new *architecture_discriminator_command* field. Now you can use an arbitrary command (rather than *uname*(2) information) to determine the architecture. See *aepconf*(5) for more information.

• The Russian message translation has been updated.

• The German message translation has been updated.

• The *ael*(1) command now has a new *incomplete* listing. It lists changes between *awaiting review* and *being integrated*. inclusive.

• The *ael*(1) command now accepts arguments for the listings. The *default-change*, *default-project*, *outstanding-changes* and *user-changes* lists now accept a user name argument.

• The *aemt*(1) command now understand the **−UNCHanged** option, so that files which are in the branch, but unchanged from the deeper branch, may be made transparent.

• A bug has been fixed in the *wecp*(1) command where the **−OverWrite** option did not honor the presence/absence of the **−ReadOnly** flag.

• There is a new *aeedit* script. See *aeedit*(1) for more information.

• A bug has been fixed in the file history mechanism (as used by the *−delta* options, *aeannotate*(1), *aedist*(1), *aepatch*(1), etc) which did not correctly understand transparent files.

• The *aeclean*(1) command now touches all of the source sfiles. It also accepts a **−NoTouch** option.

• There is a new *$change_files* substitution. See *aesub*(5) for more information.

• The *aeclean* command now touches the source files as well. Use the now **−no-touch option if you don't want this.**

• There have been several improvements to the output of the *aegis.cgi* script and the web site.

• For Aegis developers: all of the K&R insulation has been removed; you now need an ANSI C compiler to build Aegis. Some preparation has also been done to get the source ready for a C++ compiler.

Version 4.11 (29-Jan-2003)

• For Aegis developers: the developer build now uses *sudo*(8) to simplify and automate the tricky bit. The regular distribution build is unchanged.

• A bug has been fixed where the '*aet −reg*' command could not find any tests to run, cause by inconsistencies in the view path handling for project file searches.

• A partial Romainian translation has been added.

• A Spanish localization has been added. It needs work by a human.

• The French localization has been improved.

• The *aedist*(1) command now preserves the executable bit on files.

• There is a new *−descend-project-tree* option for the *aena*(1), *aera*(1), *aend*(1), *aerd*(1), *aeni*(1), *aeri*(1), *aenrv*(1), *aerrv*(1) and *aepa*(1) commands, to apply the action to all descendant branches of the project.

• A bug has been fixed in *tkaer*(1) which stopped it working on some systems.

• The *aeintegratq*(1) command now copes better with changes leaving the *awaiting integration* state.

• A bug has been fixed in the *aeimport*(1) command which misunderstood RCS branches.

• A bug has been fixed where there *aenf*(1) command would use the new *config* file about to be created, which was almost always wrong.

• There is a new *${substr}* substitution. See *aesub*(5) for more information.

• The *aeclone*(1) command now understands transparent files.

• The *aecpu*(1) command now restores test exemptions in some cases.

• There is a new *aemeasure*(1) command, which procudes simple file metrics for use with Aegis.

• There is a new *project ancestors* report.

• Trunk version number no longer have a leading dot.

• Command line completion now works for *zsh*(1).

• The *aetar*(1) command now preserves the executable bit on files.

• A bug has been fixed which caused *aetar*(1) to hang.

• The *aereport*(1) and *aesub*(1) commands now gave the same email address for users.

• The *aeannotate*(1) command now olny prints caption columns if their value changes. This highlights the differences, and is less distracting.

Version 4.10 (24-Dec-2002)

• There is a new *aemt*(1) command, used to make branch files "transparent". This is like an *aecpu*(1) command for branches, but done through the agency of a change set.

Note: The behaviour of the view path in the presence of transparent files is complete, however full support for *aecp −delta* and reports is not. Support will be present in the next release. File transparency information stored by this release will be able to be used by *aecp −delta* and reports in the next release.

• There is a new *aemtu*(1) command, to undo the effects of the *aemt*(1) command.

• It is now possible to use the *aeclone*(1) command on changes in the *awaiting development* state.

• The problematics directory permissions check has been removed from the *aeintegratq*(1) command.

• A bug has been fixed in *aecp*(1) when retrieving deltas before files were removed.

• There are new *${split}* and *${unsplit}* substitutions for manipulating search paths (*etc)*. See *aesub*(5) for more information.

• A bug has been fixed where test time stamps were not updated for batch tests which covered multiple architectures.

• The *aedist*(1) program now includes a change number, which will be used on receipt if possible. Note that this produces .ae files which are not backwards compatible; the −nopatch option will suppress inclusion of the change number in the archive.

• A German translation of Recursive Make Considered Harmful has been added, courtesy of CM Magazin.

• A bug with *aeimport*(1) and removed files has been fixed.

• A problem has been fixed with the transition case when a project changed from `develop_end_action = goto_being_reviewed` to `goto_awaiting_review` while having changes in the *being reviewed* state.

• A problem with long command lines has been fixed in the *aedist −receive*, *aepatch −receive* and *aetar −receive* commands.

• A problem with *aeimport*(1) and binary files has been fixed.

Version 4.9 (23-Oct-2002)

• The *aepatch*(1) and *aetar*(1) commands now accept −add-path-prefix and −remove-path-prefix options, for manipulating the filenames when unpacking an creating a change set. The *aepatch*(1) documentation has been significantly improved.

• There is a new *aecp −rescind* option, which may be used to rescind (roll back) a completed change. See *aecp*(1) for more information.

• The Debian */etc/mailname* file is now understood by the *${user email}* substitution.

• There is a new *project_gantt* report, which produces comma-separated-value (CSV) output, for extracting data to import into Ms. Project. Unfortunately, Mr. Project does not yet know how to import CSV files.

• It is now possible to provide a comment to the *aerpass*(1) command, just as you always could to the *aerfail*(1) command.

• The *aet*(1) program now has a −progress option, to tell you where it is up to. See *aet*(1) for more information.

• The Russian error messages have been updated.

• The *aeimport*(1) program now understands the CVS Attic directory.

• There are new *perl*, *PLural_Forms*, *capitalize*, *downcase* and *upcase* substitutions. See *aer*(5) for more information.

• A work-around for the aeimport/delta bug has been added, for projects which were imported with the buggy aeimport.

• Aegis developers will need to upgrade to GNU Autoconf 2.53 or later, as the GNU Autoconf files have been updated to work with that version. This does not affect normal users.

• Many typos have been fixed in the documentation, and some improvements have been made.

• Some build problems have been fixed.

• Numerous improvements have been made to the web interface.

Version 4.8 (19-Aug-2002)

• A bugs has been fixed in the *aetar −receive* command, where it incorrectly complained about shorty input files.

• Numerous changes have been made to the web interface. They now use cascading style sheets, have more navigation links, and inclde tarball downloads.

• Several build issues have been resolved.

• A bug has been fixed in the *aeimport*(1) command. The symptom was that the *aecp −delta* command misbehaved. The probelm was that the first delta needed a timestamp *prior to* the first change set taken from the import sets.

• A bug has been fixed in the *aepatch −send* command, where it would add `Index` lines for files with no differences.

• A bug has been fixed in the `protect_development_directory = true;` handling, where it would cause a "multiple user permissions setting" error message.

Version 4.7 (6-Aug-2002)

• The *aefind*(1) command now has −resolve as the default. To get the previous behaviour, use the −NoResolve *option.*

• In the *aeca −e* and *aepa −e* commands, it is now possible to quote strings with at-signs (@) instead of double quotes. This type of string allows newlines within the string. See *aegis*(5) for more information.

• For the benefit of Aegis developers, there is now HTML documentation genaretd by Doxygen (if you have Doxygen installed). When developing an Aegis change, in your development directory, point your browser at `doxygen-html/index.html`. The `common/str.h` file is an example of the style desired, should you wish to contribute to the effort to get all of the header files suitably annotated. Also, the removal of the K&R C support has started, see the files in `common/*.[ch]` for examples. Also <varargs.h> is not longer used anywhere.

• The *aedist*(1) command has two new options, −patch and −nopatch, which may be used to control how and when *aedist* uses patches. See *aepatch*(1) for more information.

• A bug has been fixed inthe *strncasecmp* function. This only affected you if your system did not have a native version of this function.

• The *aeca*(1) command now accepts a −fix-architecture option. This option may be used to correct the architecture list of a change automatically.

• The *aedist −receive* command now runs the *aeca −fixarch* command when a change set arrives which modified the project *config* file. This should fix many of the "architecture not in project configuration file" problems when seeding new projects.

• Some deficiencies on the “How to Become a Developer ” instructions have been addressed. The native Aegis build (but not the Makefile.in) now builds the "tags" and "TAGS" files so that it easier to navigate the sources.

• There is a new *aetar*(1) command. It may be used to send and receive tarballs as Aegis change sets. See *aetar*(1) for more information.

• Missing documentation on the *aepconf*(5) man page about the fine grained file change notification commands has been added.

• Some changes have been made to the Aegis web interface, with more back links. Also uses *html2diff*(1) if available.

• It is now possible for reviewers to use the *aet*(1) command to run tests against the changes they are

reviewing.

• The command completion for the *aet*(1) command now works better; it now completes project test names as well as change test names.

• The *aepatch*(1) and *aedist*(1) commands now cope with a wider range of input vagueries, including some weird things done by MTAs and more content transfer encoding synonyms.

Version 4.6 (11-Jul-2002)

/* vim: set ts=8 sw=4 et : */

• The *aeipass*(1) command now sleeps, rather than issue the rather alarming “warning: file modification times extend into the future” message. There is a new project *config* file field, *build_time_adjust*, which controls this behaviour, but it is strongly recommended that you leave it on the default setting.

• There is a new *${base_relative}* substitiontion, almost the inverse of *${source}*. See *aesub*(5) for more information.

• A bug has been fixed with the *aeca* and *aepa −edit* option. It was caused by the change in the previous release which added editor user preferences.

• A few build problems have been fixed.

• A bugs has been fixed in the *tkaepa* script. It would sometimes fail the "OK" button.

• A bug has been fixed in the "user changes" list. It was not explicitly passing the project name when it accessed the list of user owned changes.

Version 4.5 (26-Jun-2002)

• It is now possible to set pager and editor preferences in your *.aegisrc* file. See *aeuconf*(5) for more information.

• A bug in *aepatch −receive* has been fixed, where it would sometimes misapply a patch. The search used to determine the patch position (when it needs to be offset) has been improved.

• The *aedist*(1) and *aepatch*(1) commands now accept −delta and −delta-date options.

• The *integrate_q.sh* shell script has been replaced by the *aeintegratq*(1) Perl script. It can now lots more useful things. See *aeintegratq*(1) for more information.

• A bug has been fixed in the date parsing code (used by the −delta-date option). There was the potential to mis-calculate dates after February 2000.

• A bug has been fixed in *aepatch −receive*, where it sometimes complain of "no uudecode data in file", for files which did not require uudecoding.

• There are more change-specific substitutions available. See *${change ...}* within *aesub*(5) for more information.

• The *aepatch*(1) command now understands ordinary diff listings, in addition to the context and unified differences it already understood.

• There is a new *aeannotate*(1) command, used to produce annotated source file listings. See *aeannotate*(1) for more information.

Version 4.4 (12-May-2002)

• It is now possible to specify system wide user preferences. See *aeuconf*(5) for more information.

• The *aepatch*(1) command now understands the quoted-printable content transfer encoding.

• The *aepatch*(1) is more robust when receiving patches that want to use a change number that has already been used.

• The Dutch error message translations have been updated.

• There was a problem with the way the install directory for *aegis.cgi* was being determined. The *aegis.cgi* script is now installed into `$bindir` by default. There is a *aegis.cgi.i* helper script to find your web server's *cgi-bin* directory and copy *aegis.cgi* there, but this is not done automatically. See *aegis.cgi*(1) for more information.

• Another change has been made to cope with still more Bison changes.

• A French error message translation has been contributed.

• A problem with *aedist −receive* has been fixed, where the new *configuration_directory* could interact with the order of file creation.

• A big has been fixed in the uuencode output, which could occasionally miss the "begin" line.

• A bug has been fiexed in the context diff parsing, where it would get the last hunk wrong if it was a hunk which deleted lines, due to incorrect end-of-file handling. This affected both *aepatch −receive* and *aedist −receive*, because *aedist*(1) now includes patches for better merge behaviour.

• Numerous issues concenting the new GNU Gettext versions have been addressed.

• A number of Solaris build problems have been fixed, and one genuine bug buried in the warning messages (change completion time was wrong for changes not yet completed).

• More information about the "mod times extend into the future" warning issued by *aeipass*(1) has been added to the man page.

• Some improvements have been made to the web pages.

Version 4.3 (16-Apr-2002)

• The notification shell scripts all now use sendmail consistently. Autoconf support for locating sendmail is not yet present.

• A problem which caused a core dump on Cygwin has been fixed.

• The *aede*(1) command now gives a more informative error message when files in a branch require merging.

• There is now an interconnection between the *aeib*(1) and the *aeb*(1) command. When you specify a minimum integrate begin, you also get a minimum integrate build.

• A bug has been fixed which caused *aenf*(1) to dump core if you used the file name accept pattern.

• The executability or otherwise of each source file is now remembered. If any of the execute bits are set at *aede*(1) time, the file is remembed as executable. When an executable file is copied into an integration directory or development directory, all of the execute bits (minus the project umask) are set.

• A bug has been fixed in the "*aecp −ind*" command, where it would give a "there is no development directory" error when you tried to extract a file version from history of a completed branch.

• Many of the web pages have been updated to provide a more consistent and intuitive interface. It is also possible to get patches, via the *aepatch* command.

• Interrupts are now ignored during database writes. This should alleviate some of the problems induced by Ctrl-C. (It would be nice to find the actual cause.)

• The *aedist*(1) command has been enhanved to include a patch fragment for modified files, as well as the whole source files. On receipt, if the patch applies cleanly the whole source is ignored. If the file does not exist at the receiving end, or the patch does not apply cleanly, the whole source file is included. The incremental cost is very low, because all of the patch pieces appear in the source file, and thus compress

exceptionally well. The net result is to greatly reduce merge costs on recipt of `.ae` files. However, this change to *aedist*(1) is only backwards compatiple. Previous versions of *aedist*(1) will give a fatal error if they see a `.ae` file generated by this version of *aedist*(1).

• File name resolution is now more robust in the face of permission problems.

• Some error message translations have been improved.

• A small bug has been fixed in the history labeling.

• You can now use shell (#) and C++ (//) comments in your project *config* file, if you prefer them to C comments.

• A bug has been fixed in the maintenance of the symlink farm. It would often fail to make all of the necessary symlinks.

• There is a new project attribute, *protect_development_directory*, which when true causes the development directory to be read-only in states between *awaiting_review* and *being_integrated*.

• A problem has been fixed where some reports would fail is users had made their *.aegisrc* files unreadable.

• A number of small build problems have been fixed.

• Command completion has been added for the *aeb*(1), *ae_c*(1), *aeca*(1), *aecd*(1), *aechown*(1), *aeclean*(1), *aecp*(1), *aecpu*(1), *aedb*(1), *aedbu*(1), *aede*(1), *aedeu*(1), *aedn*(1), *aeib*(1), *aeibu*(1), *aeipass*(1), *aeifail*(1), *aena*(1), *aencu*(1), *aend*(1), *aenf*(1), *aenfu*(1), *aeni*(1), *aenrv*(1), *aentu*(1), *ae_p*(1), *aepa*(1), *aera*(1), *aerb*(1), *aerbu*(1), *aerd*(1), *aerfail*(1), *aeri*(1), *aerm*(1), *aermu*(1), *aerpass*(1), *aerpu*(1), *aerrv*(1) and *aet*(1) commands. More will be added in the future.

• It is now possible to specify a directory to contain project *config* file fragments. These fragments are then read in as if catenated as a single project *config* file. See *aepfonf*(5) for more information.

Version 4.2 (26-Feb-2002)

// vim: set ts=8 sw=4 et :

• There is a new "−No-Page-Headers" option which may be used to suppress page and column headers in listings and reports.

• There is a new "aecp −delta-from-change" option, allowing the specification of a delta number by specifying the number of a completed change.

• The "aecp −ind −delta" command now omits files which did not exist at the given delta.

• There is a new *history_label_command* which may be used to label your history files at each integration. See *aepconf*(5) for more information.

• The code which guesses which change you are working on, based on your current directory, has been enhanced to cover far more cases. It can recognize the integration directory, too.

• There is a new *Change_Log* report, which generates reports in the style of common Internet change logs.

• The web interface is now able to show you file differences between deltas.

• A bug has been fixed in the "*aecp −delta*" command (for all delta variants). The problem occurred when you wanted to copy a version of the file before the file has been modified by the branch (but it wasonly a problem for files modified later in the branch, files never modified by the branch were OK). As a side-effect of the bug fix, "*aecp −delta*" now goes significantly faster (*N* times faster, where *N* is the number of files you are copying).

• Build problems caused by new Bison releases have been fixed.

• A number of oversights in handling the new *awaiting review* state have been corrected.

• The *${expr}* substitution has been enhanced to include modulo, logical not and the six relative operators. All using the usual C syntax and precedences. See *aesub*(5) for more information.

• There is a new *${switch}* substitution, see *aesub*(5) for more information.

• A Russian localization of the error messages has been contributed.

• A bug has been fixed in the “aecp −output” code, which sometimes incorrectly created directories.

• A bug has been fixed in the symbolic link maintenance code. It now repairs links which point to a file which is too deep in the ancestor tree, and has been subsequently replaced. It now uses a single pass, rather than two passes.

• The *change_file_command* field of the project *config* file is now available at a finer granularity. There are 8 new commands (the *copy_file_command*, *copy_file_undo_command*, *new_file_command*, *new_file_undo_command*, *new_test_command*, *new_test_undo_command*, *remove_file_command* and *remove_file_undo_command* fields) which may be individually configured. They default to the previous behaviour, for backwards compatibility. See *aepconf*(5) for more information.

• A bug has been fixed in the *aepatch*(1) command, which prevented it form constructing patches for changes on completed branches.

• The *aeipass*(1) command now issues an error message if the build changes a source file. (Previously it erroneously reported that the history tool had done the damage.)

• A bug has been fixed in “aecpu −unchanged” in the case where the change had no files. (It tried to uncopy a file called the empty thring.)

• The missing *aemvu*(1) man page has been added.

Version 4.1 (6-Dec-2001)

Note: You will need to upgrade all of your Aegis machines simultaneously for this release. It introduces database changes which older Aegis release will not be able to cope with.

• A bug has been fixed in *aed*(1), which tried to access a nonexistent files under some circumstances.

• A bug has been fixed in *aede*. When two changes created the same file, the second change received a misleading message from *aede*.

• There is a new German message translation.

• There is a new *tkaepa*(1) command, giving GUI access to the *aepa*(1) command.

• The *aeclone*(1) command now runs the *change_file_command* and *project_file_command* from the project *config* file. This is in order to be more consistent withthe *aecp*(1) command.

• The "time safe" property described by Damon Poole mostly applies to Aegis’ operation. One last area related to future times and the −delta options. There is now a warning in the instance where non-time-safe behavior may occur.

• The *history_put_command* and *history_create_command* field of the project *config* file are strongly recommended to be identical. It is now possible to only specify the first one, and the second will default to it.

• A bug has been fixed in the *aeib*(1) command, when the *link_integration_directory* field in the project *config* file is false.

• There is a new *awaiting review* state, and new *aerb*(1) and *aerbu*(1) commands to go with it. It is now possible to configure your project to have changes enter the *awaiting review* state after *aede*(1), rather than the *being reviewed* state. It is also possible to skip the review states altogether and immediatelt enter the *awaiting integration* state.

• There is a new *modeP field for the specification of architectures in the project config file. The means that you can designate some architectures as mandatory and some as optional. See aepconf(5) for more information.*

• The *aenbr*(1) command now populates the new branch’s baseline with symlinks if the project *config* file is set so that they would remain after an integration build. This is more consistent with the *aedb*(1) behaviour in the same situation.

• There have been a number of changes to the web pages, accompanying the move to SourceForge, along with some corrections.

• There is a new *aels*(1) command, which may be used to list directories, annotated with Aegis’ file

attribute information.

• The *aeclean*(1) command now accepts the **−Keep** option, so that it reports what it would do, rather than actually do anything.

• A problem with the CGI interface, which reported a bug to the user, has been fixed.

Version 3.29 (31-Oct-2001)

• The *aeimport*(1) command can now import CVS repositories which contain binary files.

• There is a new *${Read_File_Simple}* substritution. It is like *${Read_File}*, but is does not substitute into the file contents.

• The *aecp −independent* command now accepts a *−output* option.

• There is a new *${environment}* substitution, allowing you to access environment variables within substitutions. See *aesub*(5) for more information.

• There is a new *${project-specific}* substitution, allowing you to define project specific value to be inserted into various commands. See *aesub*(5) and *aepconf*(5) for more information.

• The *aefind*(1) command now works with completed change. It searches the baseline.

• A problem with using the ${source} substitution within the *integrate_pass_notify_command* has been fixed. It was getting the path wrong.

• The batch test command is only ever invoked of there are tests to run. (This fixes a problem where it would simethimes run with no arguments.)

• The web reports now bahave themselves when the names of non-longer-here user appear.

• A number of errors and typos have been fixed in the documentation.

Version 3.28 (21-Aug-2001)

• There is a new *aepatch*(1) program, which may be used to send an receive changes using the classic open source patch format. See *aepatch*(1) for more information.

• The general output mechanism (for listings and reports) has been rewritten to be significantly faster.

• Numerous small things have been improved in and around the `./configure` script and the `Makefile`.

• The web interface has been improved. It should result in better save file locations being suggested for `.ae` files. cgi vs downloads

• Aegis now takes a baseline read lock during tests, so that the baseline doesn't move out from under your tests, causing mysterious failures.

• A bug has been fixed in the `subst` function of the report generator. It was free()in a string twice.

• There is a new *${developer email}* substitution, for inserting users' preferred email addresses into commands. Useful for the state transition notification commands.

• There is now more text in the *aepconf*(5) man page, explaining how each of the pattern fields are applied to file names. It is now explicit when patterns applied to whole file names, and when they only applied to path name elements.

• A segfault has been fixed in the removed file whiteout code.

• The *aesub $source* substitution now works in combination with the **−BaseLine** option.

• The *aegis.spec* file now mentions the executables again.

Version 3.27 (26-Jun-2001)

• A bug has been fixed in the *aesub*(1) `$delta` substitution. It now works correctly for completed changes.

• A bug has been fixed in *aermu*(1), when used in combination with the symlink farm. It no longer complains about "multiple user permissions set".

• A serious bug has been fixed in the locking code. The bug meant that only one build per project could happen at a time. (There was never any risk of repository or Aegis database damage.)

• A bug has been fixed in the *aedist*(1) command. It failed to correctly recognise files produced using the *aedist −send −no-ascii-armour* option.

• The *aecpu −unch* command now deals more gracefully with files which have been removed from the project in the mean time.

• There is a new *change file history* listing, similar to the *file history* report. It is much faster, much more informatuve, and less selective.

Version 3.26 (21-Jun-2001)

• Some optimizations have been done to the input parsing. Depending on your architecture, this will or won't be noticable.

• The locking has been changed so that *aeipass*(1) takes precedence over new development builds, so that there is a guarantee that *aeipass*(1) will succeed in finite time. Current development builds will run to completion, but new development builds will block until the *aeipass*(1) gets the basline lock and subsequently completes.

• The "file format error" bug in *aedist*(1) has been fixed.

• There is a new *project activity* report, which is useful to project leaders to see what has been happing in the project, sorted by time and then by user name.

• Aegis can now transparently cope with binary files, even if the history tool cannot. It does this by using a MIME encoding for binary files. (This can be configured away, if your history tool correctly handles binary files.) See *aepconf*(5) and the *User Guide* for more information.

• There is a fix for the "file unrelated" error commonly seen on Solaris and BSD when combined with an automounter, in come cases. It relies on the *bash*(1) behaviour which sets the `$PWD` environment variable. (GNU libc does this internally to the *getcwd*(3) function, not all libcs do.)

• The *aer*(1) report generator now has access to the project *config* file fields, through a new `config` field in the report generator's concept of the project state.

• There is a new *aer*(1) `$comdir` substitution, which gives access to the shared state directory, configured at build time.

• The *aebuffy*(1) now accepts a project name on the command line.

• There is a new *build_covers_all_architectures* field in the project *config* file, so that you can tell Aegis that the build tool builds all architectures wimultaneouasly. See *aepconf*(5) for more information.

• The *tkaer*(1) command now has a comment editor, so that you can edit your review fail comments from within the GUI.

• A bug has been fixed which was caused *aenbru*(1) to delete one directory level too deep when the branch was removed.

• There is a new `getuid()` function in the report generator.

• This change fixed a bug in *aede*(1) where it would not allow a branch, created with *aeimport*(1), with new files which had subsequently been modified to end development, when those files had never existed in the baseline.

• It is now possible for project administrators to nominate the developer in the *tkaenc*(1) dialog. You are presented with a pick list.

• There is a new *aesub*(1) `$history_directory` substitution. This may be used in scripts which

access the history tool's files directly.

• There is a new *change_file_undo_command* field of the project *config* file. It is similar to the *change_file_command* field (it defaults to it if unset), but is executed by all of the "undo" file commands.

• The *aede*(1) command no longer cancels your build and test time stamps. This means that you don't need to re-build if you don't change anything, after *aedeu*(1) or *aerfail*(1).

Version 3.25 (3-Apr-2001)

• It is now possible to remove users who's accounts have been removed (the affects the *aera*(1), *aerd*(1), *aeri*(1) and *aerrv*(1) commands).

• There is a new −*description-only* option to the *aeca*(1) command. This is useful for editing only the descrioption, and also for use within scripts.

• The −*file* option has now been generalized to accept "−" to mean the standard input. This is useful in scripts.

• There is a new *aebuffy*(1) command, which may be used to see what changes a user has outstanding. It needs X11 (Tk/Tcl) to work. Named after the *xbuffy*(1) command.

• The *tkaer*(1) command now presents you with a "detail" button, so that you may see the change details when pwerforming a review.

• The restriction that placed the function name at the start of the command line (*e.g.* the "−cp" of *aecp*) has been relaxed. This may now appear anywhere on the command line.

• The Bourne / BASH shell aliases have been improved, so that they now preserve quoting of special characters and white space. This dates from the earliest days of Aegis. It's wonderful to have it fixed at last.

• There is a new *aemvu*(1) command, which may be used to undo the effects of an *aemv*(1) command. This should prove less confusing than the previous method.

• A big has been fixed in the *aemv*(1) command. It failed to accept the −base-relative option, even though it was documented to do so.

• A bug has been fixed in the quote_tcl() report function. Is fixes the problem with getting the dollars sign into descriptions when using the *tkaenc*(1) command.

• The SCCS section of the *User Guide* and example configurations have been updated and confirmed to work correctly, however I've only tested this with GNU CSSC.

• A bug in the file name handling has been fixed. This was most obvious around the *aecpu*(1) command when you had *create_symlinks_before_build* turned on and you were using an automounter, but it occured at other times as well.

• The *aeimport*(1) command now understands the SCCS format. If the comments in GNU CSSC are accurate, this also means you can import BitKeeper repositories, however I am unable to confirm this.

Version 3.24 (10-Mar-2001)

• There is a new *aeimport*(1) command, which may be used to import CVS archives into Aegis.

• The cross branch merge has been improved so that it uses an earlier version number than it was using, resulting in a more sensable merge.

• A bug has been fixed in the *${quote}* substitution which incorrectly quoted the exclamation mark (!). Unfortunately, quoting isn't at all simple, because you can't exclusively use single quotes *or* double quotes *or* backslash.

• There is now a *${change description}* substitution, allowing you access to the brief description of a change in a substitution. (The suggested RCS history command have been changed to use it.)

• A Dutch localization of the error messages has been contributed.

• Project administrators can also use the *aeibu*(1) command. Handy for abandoned integrations which inconvenience everyone else.

• There is a new project *config* file field, called `build_covers_all_architectures`, which allows you to tell Aegis that your build process can cover all architectures simultaneously.

• The *${quote}* substitution has been fixed to correctly quote more characters. It now prefers the single quote (but is is npt possible to use this exclusively).

• The web site now uses PDF files for documentation, rather than gzipped PostScript. This was for lots of reasons, including the fact theat many folks couldn't work out how to print them, and also IE decompressed them "for free" but left the .gz suffix.

• The report generator, *aereport*(1), can now access fields of the *.aegisrc* file. This is important for accessing the preferred email address in various reports. • The "*aecp −delta*" command now adds removed files to the change as removed files instead of adding them as copied-but-empty files. This should make reproducing projects more accurate, but you need to use *aermu*(1) to get rid of them ,rather than *aecpu*(1).

• The *aedist* program now adds a "Content-Disposition" header to the files it generates. This means MIME programs will unpack it into a correctly named file more often.

Numerous build problems have been fixed, both for Unix and for Cygwin (Windows). There have benn some test script improvements, too.

Contributions have started to roll in using the "aedist" format. This is very encouraging. The instructions for how to do this are contained in the "Howto", in the *Developer* section.

Version 3.23 (29-Oct-2000)

• A bug has been fixed which caused the report generator *change_number* function to give garbage answers for change number zero (fortunalely, not very common).

• There is a new *mtime* function in the report generator.

• There is a new *aecomp* utility, which may be used to compare two active changes, using *tkdiff*.

• A bug in "aesub ${dd}" which reported the wrong directory when applied to branches, has been fixed.

• The project *config* file now contains two new fields, *create_symlinks_before_integration_build* and *remove_symlinks_after_integration_build*, which may be used to better control the behavior of the symlink farm at integration time. (Default behavior is backwards compatible.)

• A new utility called *tkaer* has been contributed. It is for reviewing, and shows you lists of files. When you click on one, it launches *tkdiff*(1) to examine it. You're going to like this one, folks!

• The *aedist −receive* command now preserves the testing exemptions, if possible.

• A problem with very very large test runs and the −no-persevere option has been fixed.

• The *aenf*(1) and *aent*(1) commands now accept **−template** and **−no-template** options, to control the use of new file templates.

• A nasty Catch-22 in the *aedist*(1) command has been fixed, involving the (unnecessary) use of new file templates, when the actual template files don't yet exist in the −receive development directory.

Version 3.22 (13-May-2000)

• **Please Note:** Some code has been added to Aegis to assist in diagnosing problems when restoring projects from backups. If you see a message “aegis: *project-path*: has been tampered with (fatal)” this means there are problems with the project file ownerships. The project owner needs to be >= AEGIS_UID (defaults to 100), and the project group needs to be >= AEGIS_GID (defaults to 10). Use chown −R and/or chgrp −R to fix these problems.

• The *aesub*(1) command now accepts the ${arch} substitution in combination with the −baseline option.

• A bugs has been fixed in the *aedist −receive* command, when one of the files was also locked for review.

• A bug in *aeclone*(1) has been fixed, where it dropped file move information.

• The *aeib*(1) command now correctly validates that youare actually allowed to do this integration. This may win the prize for the *oldest* Aegis bug.

• There is a new ${search_path_executable} substitution. See *aesub*(5) and *aet*(1) for more information.

• Line wrapping in reports works properly again for lines with no white space. The previous release broke it when the wide output generalization was added.

• The *aet −nopersever* option works again. The previous release broke it when the batch test support has implemented.

• A problem with the *aeb*(1) command which made it difficult to use with th symbolic link farm (in some cases) has been fixed.

• A new report is available from the web interface, showing a change-of-state histogram over time for all state transitions (not just the integrate pass transitions).

• A problem with the *aenf*(1) command which made it difficult to use with th symbolic link farm (in some cases) has been fixed.

• The *aeipass*(1) command now preserve file mod times across history updates, if the history tool gratuitously changes them.

• The Solaris and IRIX build problems (wputc, *et al*) has been fixed.

• Numerous documentation patches were received and have been applied.

Version 3.21 (12-Mar-2000)

• A couple of minor bugs have been fixed in *aedist*, especially the problem with sending an baseline image while a change is being reviewed.

• A couple of bugs have been fixed in the tkae* commands, in partucular they no longer leave temporary files lying around.

• Lots of stuff has been added to the HOWTO: a cheat sheet, how to change a project’s owner, how to use distributed development, how to become a developer.

• The problem which caused ‘aesub ${copyright_years}’ to contain duplicates has been fixed.

• There have been Y2K fixes: the date parsing for the −delta-date option has been fixed, and the web page data has also been fixed.

• The *aet*(1) command can now run more than one test at once, if configured appropriately. This is of most use on systems with more than one CPU.

• The −UNFormatted option no longer truncates column values.

• The *aesub*(1) command now accepts the −baseline option, so that you can get project-specific substitution in shell scripts.

• A bug has been fixed in *tkaenc*(1) which gave incorrect testing ssttings. It now also tracks the project testing exemptions.

• A bug in *aenf(1) has been fixed which allowed multiple instances of the same file to be created.*

• A bug has been fixed which caused ‘aesub ${search_path}’ to fail in some cases.

• A bug has been fixed in *aenf*(1) which allowed you to create the same file multiple times, corrupting Aegis' database and causing *aede*(1) to report mysterious errors. Use *aenfu*(1) multiple times to untangle things.

• Information has been added to the section 5 manual pages, detailing how to access state information from within the report generator. This should make writing report scripts a little easier.

• A bug has been fixed which caused Aegis to misbehave when launched by some versions of *cron*(8) or *at*(1).

Version 3.20 (19-Oct-1999)

• The *aeib* command is now more robust about "foreign" files in the baseline (*e.g.* root-owned core files).

• A bug has been fixed in the *${administrator_list}* substitution.

• A bug has been fixed in the *aedist −delta* option, which caused it to dump core.

• There is now a section in the *History Tool* chapter of the User Guide describing how to add checksums to your history files, in order to detect file corruptions. It is a general technique which applies to most history tools (including RCS).

• A bug has been fixed which caused *aeclone* to misbehave badly when dealing with removed files.

• There is now an embryonic "How To" document for Aegis. Please feel free to contribute subjects.

• You can now say "**−BRanch −**" as a synonym for the "**−TRunk** option, for those commands which accept it.

• The report generator now copes with more types of empty lists.

• A bug has been fixed which caused a core dump instead of a useful error message if you tried to create an alias with an illegal name.

• A bug has been removed which left undeletable branch aliases if a branch was removed.

• A bug has been fixed in *aenbru* which failed to remove the branch development directory .

• The *aenf*(1) command now behaves better when you do horrible things like turn the files you created into directories without telling Aegis first.

• A couple of small bugs have been fixed in the *aenpa*(1) command, both in error situations.

• A bug with the **−interactive** option has been fixed. It will actually ask you, now.

Version 3.19 (4-Aug-1999)

• You can now run a command to generate new file templates if you want, rather than using a simple string substitution. See *aenf*(1), *aent*(1) and *aepconf*(5) for more information. The existing functionality is still there.

• There is a new ${SUBSTitute} substitution, which provides regular expression substitutions. This is useful in new file templates.

• A bug has been fixed which allowed *aede* of a branch when there were some kinds of outstanding changes.

• The automatic change number guessing has been improved slightly, and will cope with some more variation in the *development_directory_template* field.

• There are two new commands, *aenpa*(1) and *aerpa*(1) for creating and removing project aliases. This means that you can give project branches more meaningful names.

• There is a new *aesub*(1) command. It substitutes its arguments and prints them, rather like the *echo*(1) command. This is useful when you need access to the Aegis substitutions in a script.

• The command line option "−−" is now understood. It means "the rest of the arguments on the command line are filenames or strings". Because this makes the options on the command line more "order sensitive" than usual, use with care.

• There is a new *tkaenc*(1) command, allowing you to create new change via a Tcl/Tk GUI. (And a

problem with TCL special characters in description text has been fixed.)

• The *aenf*(1) command now does the right thing with directories named on the command line. In particular, you can now use “aenf .” to import whole directory trees.

• There is a new *State-File-Name* list type, useful when writing cookbooks or makefiles to keep a web page in sync with a change.

• There is a new ${capitalize} substitution, useful for putting in new file templates.

• A bug has been fixed which caused *aeclean* to delete the development directory of changes with no files.

Version 3.18 (8-Jul-1999)

• A bug has been fixed which caused *aecp −delta* to dump core in some cases.

• A bug has been fixed which caused the create-symlinks-before-build functionality to create symlinks to deleted files.

• Still more typos and minor errors have been corrected in the documentation.

• The *aerp*(1) man page has been moved to *aerpass*(1). Similarly for *aerfail*(1), *aeipass*(1) and *aeifail*(1). This should make things easier for users to find the man pages.

Version 3.17 (22-Jun-1999)

• Another *aedist* bug has been fixed − unfortunately it was introduced while trying to fix the last one.

• A Cygwin 20.1 portability bug has been fixed.

• There is a new *${dirname_relative}* substitution. This is useful in new file templates, and also some configured commands.

Version 3.16 (15-Jun-1999)

• There is a new *tkaeca* command. It is a GUI interface to the *aeca*(1) command, using Tcl/Tk.

• There are two new reports available: the *Project-Branches* and *Project-Active-Branches* reports may be used to query about branches within a project.

• A bug has been fixed in the *aedist −receive* duplicate suppression code. It was complaining about user permissions.

• A bug has been fixed in *aeb*(1), which did strange things if you tried to build an unbuildable change.

• There is a new *−No-WhiteOut* option for the *aerm*(1) and *aemv*(1) commands, letting you suppress the “whiteout” files, along with some explanation in the man page about why they are there. See *aerm*(1) for more information.

• The default value of the “*maximum_filename_length*” field of the project *config* file has been raised from 14 to 255. If your project depends on the old default value, you will need to set it explicitly.

• The *aedist −receive* command now accepts a *−directory* option, so you can specifiy the location of the development directory.

Version 3.15 (2-May-1999)

• The "*aedist −receive*" command now accepts a **−delta** option, allowing a received change set to be applied to an historical version.

• There is now some information about managing super-projects and sub-projects in the *Branching* chapter of the User Guide.

• The *aenpr*(1) command now accepts a *−keep* option, so that you can re-attach projects moved after using the *aermpr −keep* command. See *aenpr*(1) for more information.

• The *aenpr*(1) command now accepts *−edit* and *−file* options, allowing you to specify project attributes when creating the project. See *aenpr*(1) for more information.

• If the project *developers_may_create_changes* attribute is true, the *aencu*(1) command now allows developers to destroy changes they created.

• There is a new *add_path_suffix* substitution, for manipulating search paths. See *aesub*(5) for more information.

• There are 3 new substitutions: ${bindir}, ${datadir} and ${libdir}. These are replaced by the *./configure* options of the same name (or the values calculated, if none were given to *./configure*). The old ${lib} substitution is deprecated in favour of the new ${datadir} substitution. See *aesub*(5) for more information.

• Some changes have been made which increases portability, particularly the Linux libc5 *vs* libc6 differences.

• Some changes have been made which increases portability, particularly for Windows NT. This isn't to say Aegis works under Windows NT yet, but it helps the porting efforts. Don't forget to run the mkpasswd and mkgroup utilities included in the Cygwin system.

Version 3.12 (26-Mar-1999)

• The way the Apache configuration files are scanned for and read has been changed, to adapt to recent Apache changes. The *./configure* script will now find it more often.

• The "*aedist −receive*" command has been enhanced to be more robust about change sets without headers (some browsers *generously* strip them all off.

• A bug has been fixed in the "*aedist −receive*" command which sometimes caused decompression failures. An unfortunate interaction with the Windows NT support caused CRLF sequences in the compressed data to be mangled in some cases.

• The wrong include file was being used for zlib. This has been fixed, so it should build more easily now.

• The way MANPATH is handled on Linux has been improved in the *chsrc* and *profile* commands. It will not over-ride */etc/man.config* now.

• The *aegis.cgi* script has been made more robust in coping with *aedist* errors.

• The *symlink_exception* field of the project *config* file now accepts filename patterns, not simply literal filenames.

• There was a problem compiling with gcc 2.8, involving the <stdarg.h> header. This has been fixed.

Version 3.11 (17-Mar-1999)

• The *aet*(1) command now accepts a `-force` option, forcing tests to be run, even if Aegis doesn't think they need to.

• The Aegis CGI interface has been enhanced so that you can download changes from the generated web pages listing the changes, using the *aedist* command.

• The *aedist -send* command now accepts a `-no-ascii-armor` option, which leaves off the MIME base 64 encoding. Useful for binary distributions and web servers.

• There is a new *trojan_horse_suspect* field in the project *config* file. This is used by *aedist -receive* to check for files which could be abused to carry Trojan horse attacks.

• The *aedist -receive* command now accepts a `-trojan` option which treats the incoming change set as suspect, and a `-no-trojan` option which treats the incoming change set as benign.

• The *aedist -receive* command now quotes filenames (if necessary) when executing commands, thus defending against filenames which contain semicolons.

• The *aenbru*(1) command has been implemented at last. At last! You no longer need to use the *aedbu* work-around.

• The *aedbu*(1) command now gives an error if you attempt to apply it to a branch.

• The *aermpr*(1) command may now be applied to a project with active branches, and will remove the branches as well (provided there are no active changes on any of the branches).

• The `dos_filename_required` and `windows_filename_required` fields of the project *config* file have been enhanced to reject the brain-dead Windows special filenames such as "aux" *et al*.

• The `${user}` and `${project}` substitutions have been enhanced to provide additional information when given an additional argument. Useful for file templates. See *aesub*(5) for more information.

• Several portability enhacements, notably the Windows filename incompatibility has been fixed, and also the Linux *stdlib.h* problem.

Version 3.10 (6-Mar-1999)

• As of this release you must have **zlib** installed before you can build Aegis.

• There is a new *reuse_change_numbers* project attribute, letting you control whether *aenc* fills in holes in the change number sequence. Defaults to true if not set. See *aepattr*(5) for more information.

• There is a new *integrate_begin_exceptions* field in the project *config* file. This permits the user to specify file to be omitted when the integration directory copy/link is performed.

• The *aet*(1) command has been changed so that it does not exit with an error if you have a test exemption but no tests. This is no longer an error.

• There is a new *aedist*(1) command, which may be used to send and receive Aegis change sets via e-mail and the web.

• The *aeclone*(1), *aenbr*(1), *aenc*(1) commands now accept a **`-output`** option, a file to contain the automatically generated change number. This greatly assists in writing scripts. See the man pages for more information.

• The *aent*(1) command now accepts a **`-output`** option, a file to contain the automatically generated file name. See the man pages for more information.

• There is a new *compres_database* field in the project attributes, allowing the Aegis database to be stored in a compressed form (using the GNU Zip algorthm). Unless you have an exceptionally large project, coupled with fast CPUs and high network latency, there is probably very little benefit in using this feature. (The database is usually less than 5% of the size of the repository.) On slow networks, however, this can sometimes improve the preformance of file-related commands.

Version 3.9 (7-Feb-1999)

• A bug in the merge command has been fixed. It no longer deletes all of your change source files if one of the merge commands fails.

• There is a new *tkaegis* command, using Tk/Tcl to give Aegis a GUI. Contributed by Graham Wheeler <gram@cdsec.com>. Please report *tkaegis* bugs and suggestions to Graham.

• The integrate pass command has been enhanced to cope with RCS and SCCS expanding keywords in source files (modifying the repository) on check-in. This can be ignored, or a warning can be issued, or it can be a fatal error (this is the default). See *aeipass*(1) for more information.

• The worked example in the User Guide has (finally!) been updated to use the new branch numbering. Numerous spelling errors have been corrected.

• The developer section of the worked example chapter now also includes discussion of some common questions raised by folks evaluating Aegis. It covers insulating development directories from the baseline, partial check-in and collaboration.

• The *aesub*(5) man page now brings attention to the fact that the ${Copyright_Years} substitution contains spaces. You often need to quote it.

• The man pages which mention filename limitations, now also note that where underlying file-system has stricter filename length limitations than the *filename_maximum_length* field in the project *config* file, the file-system wins. Mention of this is now also present in *aedb*(1), *etc*; Linux UMSDOS is highlighted as problematic.

• Aegis can now collect code metrics. See *aeb*(1) and *aeipass*(1) for more information.

• There are three new report functions available: *quote_url*, *quote_html* and *unquote_url*. These are all for use when creating Aegis reports for the CGI interface. See *aer*(5) for more information.

• There are several new substitutions available. These include `subst`, `trim_extension`, `trim_directory`, and `trim_filename`. See *aesub*(5) for more information.

• The integrate_q.sh script now works correctly for branches.

• Numerous configure, make and install problems have been fixed for a variety of portability targets.

• The RPM spec file has been corrected to use appropriate file attributes.

Version 3.8 (1-Oct-1998)

• Some users were unable to build the previous release, due to inconsistent wide character support by the various UNIX vendors. This has now been fixed.

• There are two new substitutions, *trim_directory* and *trim_extension*, which are useful for constructing file templates. These can be very useful in constructing skeletons of C++ classes.

• Some changes have been made to pathname handling to better cope with automounters. See *aegis*(1) for more information (see discussion of the AEGIS_AUTOMOUNT_POINTS environment variable). This assumes that paths below the automounter's mount directory are echoes of paths without it (*e.g.* `/home` is the trigger, and `/tmp_mnt/home` is where the NFS mount is performed, with `/home` appearing to be a symlink).

Version 3.7 (22-Sep-1998)

• The *aeifail*(1) and *aerfail*(1) commands now have a new −reason option, to specify the failure reason on the command line, rather than in a file.

• Some file operations are now faster. Mostly, this applies to operations which mention many files, and to projects with large numbers of files. Smaller projects may not notice any improvement.

• There is a new **−delta-date** option to the *aecp*(1) command, allowing deltas to be extracted by date. This change also had the side-effect of making extraction by delta number more accurate on branches.

• There is a new **−base-relative** option to most of the file manipulation commands, *aecp*(1), *aenf*(1), *etc*. This option may be used to specify that relative filenames are relative to the base of the source tree, rather than the current directory. There is also a related user preference, see *aeuconf*(5) for more information.

• There is a new “aeclean” command. It can be used to clean your development directories of non-source files. See *aeclean*(1) for more information.

• The *aeb*(1) command now passes through arguments of the form *name=value*, on the assumption that these are variable assignments for th ebuild tool. Previously, they were “resolved” as if they were file names.

• A serious bug in the error and interrupt handling has been fixed. This bug would sometimes case Aegis to hang, and eventually run out of stack, when the user attempted to interrupt Aegis using ^C.

Version 3.6 (5-Jul-1998)

• The *diff3_command* field of the project *config* file has been replaced by a *merge_command* field. It works exactly the same way, but Aegis moves the files around first, so that the output replaces the change source file. This results in fewer “lost” merges. Those of you who have been hacking the *diff3_command* to move the files around will need to take the moves **out** when you rename the *diff3_command* field to be the new *merge_command* field.

• The columnizing functions used by the report generator and the listings has been enhanced to understand international character sets. This allows native character sets to be used in comments and descriptions, without getting gibberish (C escapes) in the output.

• There is a new *shell_safe_filenames* field in the project *config* file. This field controls whether filenames are required to be free of shell special characters. This field defaults to true if not set, so if you are using any “interesting” filenames, you may need to explicitly set this field to false. (You still can’t use spaces or international characters in filenames.)

• There is a new ${quote} substitution for insulating shell special characters in filenames in the commands in the project *config* file.

• A number of bugs relating to environment variables have been corrected; this will make the aereport and aefind commands behave more consistently, with respect to the aegis command.

• A bug has been fixed which caused the final newline of new test files to be omitted.

• A bug which prevented the “aeb −minimum” option from working in any non-trivial case has been fixed.

Version 3.5 (28-May-1998)

• A bug was fixed in the lock waiting code. Aegis will now correctly wait for locks when there are several users blocking on the same lock.

Version 3.4 (22-May-1998)

• There is a new “aegis −clone” command, used to replicate changes across branches. See *aeclone*(1) for more information.

• There is a new “−No-Wait” command line option, which asks for a fatal error if a lock cannot be obtained immediately; this applies to all commands which takes locks. See *aeuconf(5) for more information.*

Version 3.3 (4-Apr-1998)

This release is a bug fix release, and mostly install and portability bug, at that.

• The problem with *errno* defines messing up *glue.c* has been fixed.

• Numerous fixes to the wide character support, to cope with the vagueries of wide character support on many platforms.

• The problem with the LINES and COLS environment variables messing up testing have been fixed. Some tests gave false negatives because of this.

• There is a new *aeb −minimum* option, for use with symbolic links, only, which has a minimal set of source file links, rather than everything in the baseline.

Version 3.2 (22-Mar-1998)

• There are some additional reports available via the web interface. They are mainly to extract error causes and trends from the project history statistics.

• There have been a number of minor bug fixes concerning the handling of old 2.3 projects. This should ease transition for users with existing 2.3 projects.

• A bug in aecp −delta has been fixed, where Aegis was trying to find change state files one branch level too high.

• There is now a re-try performed when a stale NFS file handle error is detected. This should make it easier for some sites which are heavily networked.

• There have been some improvements to the way Control-C is handled. It should be more responsive when waiting for locks.

• Project administrators may now end development of a branch. Since branches can endure for months or years, the orginal branch creator may have moved on. This copes with this situation.

Version 3.1 (15-Jan-1998)

Version 3.0 was not used by many sites. It was available as beta software for about a year, in numerous incarnations. Version 3.1 is the first completely stable version since adding full branching support.

Version 3.0

Version 3.0 is fully backwards compatible with earlier versions, however once a project has been used under 3.0, it will not be possible to revert, e.g. to version 2.3, without restoring the project's "info" directory from backup. While this was generally true of previous releases, any additional state information was usually undo-able with *vi*(1). This time the process is much more involved because the project state files and the change state files have been combined as a necessary step in implementing branches.

Version 3.0 Major New Features

• Aegis now has a feature known in the literature as long transactions, also known as branches. This allows appropriately created changes to be treated as if they were projects, and thus to have changes made to them. This allows a hierachy of changes within changes, to any desired depth. See the *Branching* chapter of the *Aegis User Guide* for more information.

• The project state files have been merged with the change state files. This is part of the implementation of branching. If you have written your own reports, you may need to alter them slightly. For example, in version 2.3 and earlier, reports accessed the project state file using

```
auto p, ps;
p = project[project_name()];
ps = p.state;
```

Because the project state has been moved into a change state, the *state* field above now points at a change state description, and most of the old project information is contained in the *branch* field within it. Reports access this information as

```
auto p, ps;
p = project[project_name()];
ps = p.state.branch;
```

Except for files, which were already present in the *cstate*, so access to the project file list need not change. See the new *aecstate*(5) for more information.

• The new project command now creates branches to match the version number specified. See *aenpr*(1) for more information.

• The error messages of Aegis have been internationalized. This affects how you build Aegis, and the environment Aegis runs in. See the *BUILDING* file for more information. The *cshrc* and *profile* shipped with this release set the LANG environment variable to "en" (for English) if you have not set it; otherwise ther error messages would be terse and uninformative.

• The *aet (1)* command can now suggest tests to be run. This is done by correlating the source files and test files from each change. See *aet*(1) for more information.

• There is now an *aereport*(1) command. The separates out the report functionality from the main body of the Aegis code, allowing the report generator to be used in places where more trust is required.

• There is an intranet Web interface, which is installed automaticly when the install script discovers a web server. This interface allows browsing of much of the Aegis meta-data, of all publicly accessible projects.

• There is now an *aefind*(1) command. This is very similar to the UNIX *find*(1) command, except that it finds in the unified directory stack of a change and its project. The introduction of full branch support can sometimes mean that finding a file may require looking in more than two directories; the *aefind*(1) command makes this simple again.

Version 3.0 Minor New Features

• There is now a −No_Pager option, to prevent listings and help from being redirected to a pager. There is also a user preference to more thoroughly disable paging, and a −PAGer option to override it. See *aegis*(1) and *aeuconf*(5) for more information.

• There is now a −No_PErsevere option to *aet*(1), allowing you to request that *aet*(1) stop after the first test failure. There is also a user preference to set this permanently if desired, and a −PErsever option to override it. See *aet*(1) and *aeuconf*(5) for more information.

• The copyright years attribute has been moved from being a project attribute to a change attribute. This is consistent with a number of other fields which have transparently moved from the project state files into the

change state files, as a result of branching support. See *aeca*(1) and *aecattr*(5) for more information.

• There is a new *Search_Path* substitution, to support builds on branches. See *aesub*(5) for more information. As a side effect, you can also use it in the *test_command* field of the project *config* file, and thus have a search path to look down for data files for your tests.

• Test times are now remembered, so that tests are only run if they need to be. This allows you to keep working on a test, and Aegis only runs those that have not yet passed.

• Aegis now uses “fingerprints” to tell if files have changed, rather than simply relying on file modification time stamps. While this makes Aegis more robust, there is one caveat: it is recommended that 3.0 be installed when there are no changes in the ’being reviewed’ or ’awaiting integration’ states, in any project.

• There is now a log file prefernces control, allowing users to set their preferred logging behaviour. See *aeuconf*(5) for more information.

• It is now possible to specify the filename for new tests on the command line. See *aent*(1) for more information.

• It is not possible to specify a pattern for test filesnames. See *aepconf*(5) for more information.

• There is now a **−MAXimum** option to the *aeib* command, allowing you to keep obsolete derived files at integrate begin. This can avoid long integration build times for large projects.

Version 3.0 Bug Fixes

• Architecture names are now checked a ‘develop end’ time, to ensure there are no unknown variants. This fixes the mysterious “you must build again” problem.

• The *aecp*(1) and *aed*(1) commands now take a baseline read lock, to be more symetric with the *aeb*(1) command which has always done so. The *aeipass*(1) command takes the complementary baseline write lock, ensuring the the baseline remains constant for the duration of builds, file copies, differences and merges. The manual entries for these commands have all been improved to document this behaviour. See *aeb*(1), *aecp*(1), *aed*(1) and *aeipass*(1) for more information.

• There are now some reminder scripts in the library, which can be run from *crontab*(1). These are installed into the */remind* directory. These scripts can be used to remind users of changes in various states, such as those being developed or being reviewed.

• All of the commands which accept the *−Edit* option now also accept a *−Edit_BackGround* option, allowing edit commands to be piped in from the standard input.

• The *aecp*(1) command now accepts a *−INDependent* option, allowing files to be copied independent of any change (similar to the *−INDependent* option of the command.) See *aecp*(1) for more information.

• The *aecp*(1) command now accepts a *−Read_Only* option, allowing files to be copied into a change specifically to insulate it from baseline changes. Such files must be uncopied before development may end. See *aecp*(1) for more information.

• The *aenrls*(1) command is now used *only* to convert pre-3.0 projects into post-3.0 projects. This is because the full branching support in 3.0 makes it more useful to create a new release of a project by ending development on the branch of the previous release and starting development of a new branch numbered for the new release. See the *Branching* chapter of the *User Guide* for more information.

Version 2.3

• The merging behaviour of the *aed*(1) command has changed. If any files require merging, it only merges. In this way, merged files are not lost in the rest of the output. Also, there are now command line options and user preferences so that you can select to only merge or only difference. See *aed*(1) and *aeuconf*(5) for more nformation.

• It is now possible to assign symbolic names to project deltas. This means that you may now recreate earlier project baselines by name.

• All commands which accept a **−Edit** option now check for most errors before commencing the edit. This avoids wasted edits in many error cases.

• Fuzzy file name matches are now used to improve the error messages from aecp, aerm, etc.

• Version number separators in project names are preserved across new releases. Particularly, you can use a minus ('−') between the name and the major version number.

• A new "copyright_years" project attribute has been added. This is a list of years maintained at integrate begin time, to automate the insertion of list of copyright years into copyright messages and documentation. There is a new ${Copyright_Years} substitution and the copyright years are also listed in the "aegis −list version" listing. See *aesub*(5) and *ael*(1) for more information.

• It is now possible to specify patterns for acceptable and unacceptable filenames in the project configuration file. See *aepconf*(5) for more information.

• Four more functions have been added to the report language: *length*, *split*, *substr* and *wrap*. See *aer*(5) for more information.

• The tests distributed with are now more stable on very fast hosts. See the environment variables section of *aeb*(1) for more information.

• The *lib/config.example* directory of the distribution now contains files with example portions of the project *config* file. May thanks to David R Shue <shue@ll.mit.edu> for this suggestion.

Changes made in the previous release included:

Version 2.2

This release of Aegis provides 3 of the most commonly requested features: support for heterogeneous development, support for a greater range of DMTs, support for user-defined reports.

• Aegis now supports heterogeneous development. Now you can be sure that your project not only always builds and tests sucessfully, but that it does so across a configurable set of system or hardware architectures. See the *Heterogeneous Development* secion of the *Tips and Traps* chapter of the User Guide for more information.

• Aegis can now cope with a wider range of Dependency Maintenance Tools (DMTs). It now has the ability to fill development directories with symbolic links to all files in the baseline which are not present in the development directory. This allows DMTs to assume all files are present below the current directory, allowing DMTs such as *cake* and *GNU Make* to be used. See the *Dependency Maintenance Tool* section of the User Guide and *aeb*(1) for more information.

• Aegis now has a report generator, so you can create your own reports. Many "canned" reports are included in this distribution; of particular interest to many will be the *File_Activity* report, which details currently active files. See *aer*(1) for more information.

• Aegis is now configured using a shell script called *configure*, distributed with the package. This shell script is generated using GNU Autoconf. See the *BUILDING* file for more information.

• The *AEGIS* environment variable has been renamed *AEGIS_PATH*, to bring it in line with the AEGIS_PROJECT and AEGIS_CHANGE environment variable names. The old name will keep working for some time, but aegis will warn you.

• Filename lengths are now configurable. The 14 character portability limit is still the default, but a higher limit is configurable for each project, up to the filesystem filename limit. See *aepconf*(5) for more information.

• It is now possible to specify that filenames must be within the minimum character set mandated by POSIX. The default is as before, to allow any printing character. See *aepconf* (5) for more information.

• Limits on the length of project names have been relaxed. Project names are now only limited by the filesystem filename limit.

• It is now possible to specify the command to run tests, allowing a project to use a specialized test facility, rather than be forced to use shell scripts. See *aet*(1) and for more information.

• The commands which accept the −*Edit* now preserve the edited text in the event of a failure.

• The commands which delete files now accept a −*Interactive* option, which causes them to prompt the user for confirmation of file deletion. This can be made the default by an appropriate setting of the aliases or individual users preferences files. See *aenfu*(1), *aentu*(1), *aecpu*(1), and *aeuconf* (5) for more information.

• The *aecp*(1) command now accepts directory names, allowing whole directory trees to be copied into a change. The *aecpu*(1) command now has a **−UNChanged** option which allows the unchanged files to be uncopied.

• The *aeb* command now accepts file names, allowing partial builds to be performed. See *aeb*(1) for more information.

• There is a new *aechown*(1) command to facilitate reassigning the developer of a change which is in the *being developed* state.

• It is now possible for project administrators to assign changes to specific developers. See *aedb*(1) for more information.

Version 2.1

• Can now ask for history to maintained for file generated by the build. This is useful for generating patch files.

Version 2.0

• A new command has been added to facilitate changing the name of a file as part of a change. See *aemv*(1) for more information.

• It is now possible to list the locks currently held. See *ael*(1) for more information.

• If no other defaulting mechanism is specified, aegis will now attempt to guess the project name and change number from the pathname of the current directory. This only works from within development directories.

• The *aenc*, *aeca*, *aerfail*, *aeifail* and *aepa* commands now accept a **−Edit** command line argument. See the relevant manual pages for more information.

• The *aenpr* command now understands the **−MAJor** and **−MINOr** options, allowing the initial version of a project to be something other than 1.0.

• The *aed* command now understands the **−ANticipate** option. See *aed*(1) for more information.

• It is now possible to list all the outstanding changes of a project, or of all projects. See *ael*(1) for more information.

Version 1.4

• Support has been added for systems without the *seteuid* system call, or those with crippled implementations.

• Most of the unimplemented command variants have been finished. These include *New Change Undo*, *Develop Begin Undo* and *ReMove PRoject*. Most notable of the exceptions is *-ANticipate* option for the *-CoPy_file* and *-DIFFerence* command.

• The User Guide has been added to, making it a little more complete. It still needs more work, sigh.

• The code to handle automounters has been made more robust.

• The command substitutions have been vastly improved, and are now documented.

NAME

aegis − project change supervisor

aegis (ee.j.iz) *n.*, a protection, a defense.

SPACE REQUIREMENTS

You will need up to 250MB to unpack and build the *aegis* package. (This is the worst case seen so far, most systems have binaries smaller than this, 200MB is more typical.) Your mileage may vary.

SITE CONFIGURATION

The **aegis** package is configured using the *configure* shell script included in this distribution.

The *configure* shell script attempts to guess correct values for various system-dependent variables used during compilation, and creates the *Makefile* and *common/config.h* files. It also creates a shell script *config.status* that you can run in the future to recreate the current configuration.

Upgrading

The `./configure` script will look for an existing install of Aegis and use the existing configuration settings. This works best if the version you are upgrading is 4.11 or later.

To disable looking for an existing installation (maybe because you want to change the prefix), use the `./configure --with-no-aegis-configured` option.

To change the `AEGIS_UID` and `AEGIS_GID` values (these control the ownership of Aegis' system files) you need to set environment variables of these names **before** running the *./configure* script. You almost never need to do this, so if you have no idea what this is about, don't try to change them.

Before You Start

Before you start configuring, it is worth reading the *OTHER USEFUL SOFTWARE* section, below.

The *configure* script checks for the internationalization library and functions. If your system does not have them, it is worth fetching and installing **GNU Gettext** before you run the *configure* script. Make sure that the *msgfmt* command from GNU Gettext appears earlier in your command search PATH than the existing system ones, if any (this is very important for SunOS and Solaris). You must do the GNU gettext install *before* running the *configure* script, or the error messages, even for English speakers, will be terse and uninformative. Remember to use the GNU gettext configure *−−with-gnu-gettext* option if your system has native gettext tools.

The *configure* script checks for compression libraries and functions. If your system does not have them, you must download and install the **GNU zlib** compression library (see `http://www.gzip.org/zlib/` for download) and the **bzip2** compression library (see `http://www.bzip.org/` for download) before you run the *configure* script. These libraries are essential, Aegis will not build without them. (**Note:** `zlib` is not the same thing as `zlibc` which does something completely different.)

The *configure* script checks for the regular expression library and functions. If your system does not have them, it is worth fetching and installing **GNU rx** compression library before you run the *configure* script. (Note: test 81 will fail if the POSIX regular expression functions are not available.)

The GNOME libxml2 library (`http://xmlsoft.org/`) is used to parse XML, you will need version 1.8.17 or later. You do not have to install the rest of GNOME as this library is able to be used by itself. This package is **not** optional, you need it to successfully build Aegis.

The libcurl library (`http://curl.haxx.se/`) is used to fetch remote files. This library is optional, but some functionality, particularly *aedist −replay, will not work without it. If you are using a package based install, you will need the libcurl-dev or libcurl-devel package as well.*

Running Configure

Normally, you just *cd* to the directory containing *aegis*' source code and type

```
% ./configure --sysconfdir=/etc
...lots of output...
%
```

If you're using *csh* on an old version of System V, you might need to type

```
% sh configure --sysconfdir=/etc
...lots of output...
%
```

instead to prevent *csh* from trying to execute *configure* itself.

Running *configure* takes a minute or two. While it is running, it prints some messages that tell what it is doing. If you don't want to see the messages, run *configure* with its standard output redirected to */dev/null*; for example,

```
% ./configure --sysconfdir=/etc --quiet
%
```

There is a known problem with GCC 2.8.3 and HP/UX. You will need to set `CFLAGS = -O` in the generated Makefile. (The configure script sets it to `CFLAGS = -O2`.) This is because the code optimization breaks the fingerprints. If test 32 fails (see below) this is probably the reason.

There is a known problem with IRIX builds. You need to use the following configuration

```
# systune ncargs 0x8000
```

to increase the length of command lines.

For mips IRIX and IRIX64 using the MipsPro compiler up to at least version 7.3 you must specify the flag to allow −I for loop initializations. You may give either of:

```
CXXFLAGS='LANG:ansi-for-init-scope=ON'
CXXFLAGS='LANG:std'
```

Also required is *−lCio* but configure will test for that. Even using that library there remains a link failure due to:

```
Unresolved text symbol
"std::_List_base<undo_item*,std::allocator<undo_item*> >::clear(void)"
```

on several of the binaries. A work around for this problem is not known at this time.

By default, *configure* will arrange for the *make install* command to install the **aegis** package's files in */usr/local/bin*, */usr/local/com/aegis*, */usr/local/lib/aegis*, */usr/local/man* and */usr/local/share/aegis*. There are a number of options which allow you to control the placement of these files.

`--prefix=`*PATH*

This specifies the path prefix to be used in the installation. Defaults to */usr/local* unless otherwise specified. The rest of these building instructions assume you are using the default */usr/local* as the install prefix.

`--exec-prefix=`*PATH*

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. Defaults to *${prefix}* unless otherwise specified.

`--bindir=`*PATH*

This directory contains executable programs. On a network, this directory may be shared between machines with identical hardware and operating systems; it may be mounted read-only. Defaults to *${exec_prefix}/bin* unless otherwise specified.

`--datadir=`*PATH*

This directory contains installed data, such as the documentation, reports and shell scripts distributed with Aegis. On a network, this directory may be shared between all machines; it may be mounted read-only. Defaults to *${prefix}/share/aegis* unless otherwise specified. An "aegis" directory will be appended if there is none in the specified path.

`--libdir=`*PATH*
: This directory contains installed data, such as the error message catalogues. On a network, this directory may be shared between machines with identical hardware and operating systems; it may be mounted read-only. Defaults to *${exec_prefix}/lib/aegis* unless otherwise specified. An "aegis" directory will be appended if there is none in the specified path.

`--mandir=`*PATH*
: This directory contains the on-line manual entries. On a network, this directory may be shared between all machines; it may be mounted read-only. Defaults to *${prefix}/man* unless otherwise specified.

`--sharedstatedir=`*PATH*
: This directory contains share state information, such as the Aegis lock file, and information on the location of the various Aegis projects. On a network, this directory may be shared between all machines; it **must** be mounted **read-write**. Defaults to *${prefix}/com/aegis* unless otherwise specified. An "aegis" directory will be appended if there is none in the specified path.

`--sysconfdir=`*PATH*
: Location of system configuration files. You should almost always use the */etc* directory.

configure ignores any other arguments that you give it.

On systems that require unusual options for compilation or linking that the *aegis* package's *configure* script does not know about, you can give *configure* initial values for variables by setting them in the environment. In Bourne-compatible shells, you can do that on the command line like this:

```
$ CC='gcc -traditional' LIBS=-lposix \
  ./configure --sysconfdir=/etc
...lots of output...
$
```

Here are the *make* variables that you might want to override with environment variables when running *configure*.

Variable: CC
: C compiler program. The default is *cc*.

Variable: INSTALL
: Program to use to install files. The default is *install* if you have it, *cp* otherwise.

Variable: LIBS
: Libraries to link with, in the form `-l`*foo* `-l`*bar*. The *configure* script will append to this, rather than replace it.

If you need to do unusual things to compile the package, the author encourages you to figure out how *configure* could check whether to do them, and mail diffs or instructions to the author so that they can be included in the next release.

Common Problem

It is very common that other packages, such as *gettext*, *rx* and *zlib* are installed using */usr/local* as the prefix. However, the configure script can't work this out, even when it, too, is using */usr/local* as the prefix.

To cope with this, you need to say

```
$ CPPFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib \
  ./configure --sysconfdir=/etc
...lots of output...
$
```

when running configure. Substitute the appropriate prefix if you are using something other than the default */usr/local* prefix. Watch the output... it should now find your installed packages correctly.

GCC Version 3.*

On some operating systems, notabley MacOsX Jaguar and Panther, g++ versions 3.* will produce link-time errors complaining of missing typeinfo symbols. The only known fix for this problem is to use GCC version 2.95, 2.96 or 4.*. This means MacOsX Tiger does not have the problem.

AIX Command Line Lengths

For some reason, AIX has a very short command line length limit by default. You can extend this by using the command

```
$ systune ncargs 0x8000
$
```

You will need to do this to build Aegis. It has some very long link lines.

PRIVILEGES

There are a number of items in the generated *Makefile* and *common/config.h* file which affect the way *aegis* works. If they are altered too far, *aegis* will not be able to function correctly.

AEGIS_MIN_UID
: This specifies the minimum unprivileged uid on your system. UIDs less than this may not own projects, or play any other role in an aegis project. The default value is 100.

AEGIS_MIN_GID
: This specifies the minimum unprivileged GID on your system. GIDs less than this may not own projects, or play any other role in an aegis project. The default value is 10.

AEGIS_USER_UID
: This is the owner of files used by *aegis* to record pointers to your projects. It is *not* used to own projects (i.e. it must be less than AEGIS_MIN_UID). If possible, the *configure* script tries to work out what value was used previously, but you must specify the −−prefix option correctly for this to work. Because of operating system inconsistencies, this is specified numerically so that *aegis* will work across NFS. The default value is 3.

AEGIS_USER_GID
: This is the group of files used by *aegis* to record pointers to your projects. It is *not* used as the group for projects (i.e. it must be less than AEGIS_MIN_GID). If possible, the *configure* script tries to work out what value was used previously, but you must specify the −−prefix option correctly for this to work. Because of operating system inconsistencies, this is specified numerically so that *aegis* will work across NFS. The default value is 3.

DEFAULT_UMASK
: When *aegis* runs commands for you, or creates files or directories for you, it will use the defined project umask. This is a project attribute, and may be altered using the *aepa*(1) command. The DEFAULT_UMASK is the umask initially given to all new projects created by the *aenpr*(1) command. The default value of DEFAULT_UMASK is 026. See the comments in the *common/config.h* file for an explanation of the alternatives.

It is required that *aegis* run set-uid-root for all of its functionality to be available. It is **not** possible to create an "aegis" account and make *aegis* run set-uid-aegis. This is because *aegis* does things as various different user IDs, sometimes as many as 3 in the one command. This allows *aegis* to use UNIX security rather than inventing its own, and also allows *aegis* to work across NFS. To be able to do these things, *aegis* must be set-uid-root. Appendix D of the *Aegis User Guide* explains why *aegis* must run set-uid-root; please read it if you have concerns.

Remember Your Settings

It is important to remember your configuration settings. This way, it will be a simple matter when it comes time to upgrade Aegis.

BUILDING AEGIS

All you should need to do is use the

```
% make
...lots of output...
%
```

command and wait. When this finishes you should see a directory called *bin* containing several files: *aegis*, *aereport*, *aefind*, *aefp*, and *fmtgen*.

aegis The *aegis* program is a project change supervisor.

aefp The *aefp* program may be used to "fingerprint" files. It is used to test Aegis (see the testing section, below) but it isn't installed.

aereport The *aereport* program is used to query Aegis' database.

aefind The *aefind* program is used to find files.

fmtgen The *fmtgen* program is a utility used to build the *aegis* package; it is not intended for general use and should not be installed.

You can remove the program binaries and object files from the source directory by using the

```
% make clean
...lots of output...
%
```

command. To remove all of the above files, and also remove the *Makefile* and *common/config.h* and *config.status* files, use the

```
% make distclean
...lots of output...
%
```

command.

The file *aux/configure.in* is used to create *configure* by a GNU program called *autoconf*. You only need to know this if you want to regenerate *configure* using a newer version of *autoconf*.

Upgrading

When upgrading from one release to a newer one, it is important that all of the machines on your network are running the same release of Aegis. This minimizes the possibility of database incompatibilities. In general, Aegis is backwards compatible with earlier releases, but not forwards compatible in the face of new capabilities.

OTHER USEFUL SOFTWARE

Before describing how to test *aegis*, you may need to grab some other free software, because the tests require it in some cases, and because it is generally useful in others.

GNOME libxml2

The GNOME libxml2 library (`http://xmlsoft.org/`) is used to parse XML. Version 1.8.17 or later. You do not have to install the rest of GNOME as this library is able to be used by itself. This package is **not** optional, you need it to successfully build Aegis.

cook This is a dependency maintenance tool (DMT). An example of a well-known DMT is *make*(1), however this old faithful is mostly not sufficiently capable to meet the demands placed on it by the *aegis* program, but *cook* certainly is. The *cook* package is written by the same author as *aegis*. The *cook* package is necessary if test 11 is to be meaningful. It is also used in the documentation. The *cook* program may be found at the same archive site as the *aegis* program. The *cook* program is available under the terms of the GNU General Public License.

GNU diff

If the *diff*(1) utility supplied by your flavor of Unix does not have the `-c` option, you will need GNU diff for *aepatch*(1) to work (and the *aepatch*(1) tests to pass). Context differences are also helpful for reviewing changes. GNU diff is essential for Solaris, because the Solaris diff has bugs that Aegis' tests uncover.

GNU patch

For best results with the *aepatch*(1) and *aedist(1) when receiving change sets, you need the GNU patch utility.*

iso-codes

This package provides the ISO 639 and ISO 639-3 language code lists, the ISO 3166 territory code list, list as XML files.
Homepage: http://pkg-isocodes.alioth.debian.org/

RCS This is a source control package, and is available from any of the GNU archives. (It is best to compile and install RCS *after* GNU diff. This is because the RCS configuration hard-codes the pathnames of the GNU diff utilities it needs into the RCS executables.) This package isn't essential as Aegis comes with its own *aesvt*(1) history tool − although you are free to use any history tool you like.

GNU Gettext
: Many systems do not yet supply the *gettext*(3) function. Aegis uses this function to internationalize its error messages. If your system does not have this function, you should fetch and install GNU Gettext *before* running the *configure* script. If you do not, Aegis will still work, but the error messages will be rather terse, even for English speakers. (You will be able to tell if your system has the internationalization library and functions, because the *configure* script will report finding `-lintl` and *(CWlibintl.h* and `msgfmt` in its running commentary.) Please note that the GNU Gettext implementation is likely to be superior to the one supplied with your system, if any. Remember to use the GNU gettext configure *−−with-gnu-gettext* option if your system has native gettext tools.

  Please note: if you install GNU gettext package into */usr/local* (for example) you must ensure that the Aegis *./configure* script is told to also look in */usr/local/include* for include files (CFLAGS), and */usr/local/lib* for library files (LDFLAGS). Otherwise the *./configure* script will incorrectly conclude that GNU Gettext has not been installed.

  GNU Gettext version 0.11.1 or later is recommended.

GNU Groff
: This GNU software replaces the documentation tools which (sometimes) come with UNIX. They produce superior error messages, and support a wider range of functionality and fonts. The *Aegis* User Guide was prepared with GNU Groff. You need GNU Groff 1.14 or later.

bison This GNU software is a replacement for *yacc*(1). Some systems have very sick yaccs, and *bison* may be necessary if your system include files disagree strongly with your system's yacc. The generated *Makefile* will use bison if you have it.

fhist This software, available under the terms of the GNU General Public License, is a set of file history and comparison utilities. It was originally written by David I. Bell, and is based on the minimal difference algorithm by Eugene W. Myers. This copy is enhanced and maintained by the same author as *Aegis*, and may be found at the same archive site, in the same directory.

rx This library provides POSIX regular expressions, for systems which don't have them. (Note: test 81 will fail if the POSIX regular expression functions are not available.)

zlib This library provides access to the GNU Zip (de)compression algorithm(s). It is essential to have this installed before you build Aegis. The home page may be found at `http://www.gzip.org/zlib/` if you need to download it. Note: this is not the same as **`zlibc`** which is Linux specific.

tkdiff This program shows the difference between two text files, nicely highlighted in color. This is used by the *tkaer* and *aecomp* scripts (and probably others as they are contributed). By John M. Klassa, http://www.ede.com/free/tkdiff

libmagic If *libmagic*(3) is present, the *aeget*(1) CGI handler will use it to determine the MIME type of files. This is installed by **file** version 4.0 and later (`ftp://ftp.astron.com/pub/file/`), and uses the same database as the *file*(1) command. If this library is not present when Aegis is built, a much less accurate method will be used.

The tests also depend on the presence of a number of common UNIX programs, including but not limited to: *cc*, *cmp*, *diff*, *ed*, *find*, *make*, etc. Depending on your version of UNIX, some or all of these programs may be in optional packages. (This is especially true of Linux.) You need to ensure that these programs are correctly installed before you run the tests.

TESTING AEGIS

The *Aegis* program comes with a test suite. To run this test suite, use the command

```
% make sure
...lots of output...
Passed All Tests
%
```

The tests take a minute or two each, with a few very fast, and a couple very slow, but it varies greatly depending on your CPU.

Known Problems

In order to get the long form of the error messages on Solaris, it is necessary to install GNU Gettext before running ./configure, and once ./configure has been run you need to edit the Makefile to statically link the executables.

The *test/00/t0011a.sh* file assumes the *cook*(1) command by the author is somewhere in the command search path. This test reproduces the example used in Chapter 3 of the User Guide. If the *cook*(1) command is not available, this test gives a pass result without testing anything.

If you are using HPUX and GCC, test 32 fails if you use −O2. You need to edit the Makefile to only optimize at −O, delete the objects and rebuild.

If you are using Solaris' diff, test 133 will report "no result". You need to install GNU diff, because the Solaris diff has bugs.

If you are using Sun's *tmpfs* file system as your */tmp* directory, the tests will fail. This is because the *tmpfs* file system does not support file locking. Set the *AEGIS_TMP* environment variable to somewhere else before running the tests. Something like

```
% setenv AEGIS_TMP /usr/tmp
%
```

is usually sufficient if you are using C shell, or

```
$ AEGIS_TMP=/usr/tmp
$ export AEGIS_TMP
$
```

if you are using Bourne shell. Remember, this must be done before running the tests.

If the tests fail due to errors complaining of "user too privileged" you will need to adjust the *AEGIS_MIN_UID* defined in the *common/config.h* file. Similarly for "group too privileged", although this is rarer. This error message will also occur if you run the tests as root: the tests must be run as a mortal each time.

If the POSIX regular expression functions are not available, test 81 will fail. The GNU rx library provides these. Installing it and re-configuring and re-building Aegis will solve the problem.

TESTING SET-UID-ROOT

If the *Aegis* program is not set-uid-root then it runs in "test" mode which gives you some confidence that *Aegis* is working before being tested again when it is set-uid-root. Two pass testing like this means that you need not trust your system to a set-uid-root program which is not known to work.

You will need to do a little of the install, to create the directory which will contain *Aegis*' lock file. (Note that these building instructions assume you are using the default */usr/local* as the install prefix. You will need to substitute as appropriate if you are using some other prefix.)

```
# make install-libdir
mkdir /usr/local/lib/aegis
chown 3 /usr/local/lib/aegis
chgrp 3 /usr/local/lib/aegis
chmod 0755 /usr/local/lib/aegis
mkdir /usr/local/com/aegis
chown 3 /usr/local/com/aegis
chgrp 3 /usr/local/com/aegis
```

```
chmod 0755 /usr/local/com/aegis
chown root bin/aegis
chmod 4755 bin/aegis
#
```

As you can see, the previous command also changed *Aegis* to be set-uid-root. Once this has been done, *Aegis* should be tested again, in the same manner as before.

```
% make sure
...lots of output...
Passed All Tests
%
```

You should test *Aegis* as a mortal in both passes, rather than as root, to be sure the set-uid-root functionality is working correctly.

It is required that *Aegis* run set-uid-root for all of its functionality to be available. It is **not** possible to create an "aegis" account and make *Aegis* run set-uid-aegis. This is because *Aegis* does things as various different user IDs, sometimes as many as 3 in the one command. This allows *Aegis* to use UNIX security rather than inventing its own, and also allows *Aegis* to work across NFS. To be able to do these things, *Aegis* must be set-uid-root. Appendix D of the *Aegis User Guide* explains why *Aegis* must run set-uid-root; please read it if you have concerns.

INSTALLING AEGIS

As explained in the *SITE CONFIGURATION* section, above, the *Aegis* package is installed under the */usr/local* tree by default. Use the `--prefix=`*PATH* option to *configure* if you want some other path.

All that is required to install the *Aegis* package is to use the

```
% make install
...lots of output...
%
```

command. Control of the directories used may be found in the first few lines of the *Makefile* file if you want to bypass the *configure* script.

The above procedure assumes that the *soelim*(1) command is somewhere in the command search *PATH*. The *soelim*(1) command is available as part of the *GNU Groff* package, mentioned below in the *PRINTED MANUALS* section. If you don't have it, but you do have the *cook* package, then a link from *roffpp* to *soelim* will also work.

The above procedure also assumes that the *$(prefix)/man/man1* and *$(prefix)/man/man5* directories already exist. If they do not, you will need to *mkdir* them manually.

USER CONFIGURATION

The *Aegis* command is assumed to be in a generally accessible place, otherwise users will need to add the relevant directory to their PATH. Users should add

```
source /usr/local/lib/aegis/cshrc
```

to the end of their *.cshrc* file for the recommended aliases. (Note that these building instructions assume you are using the default */usr/local* as the install prefix. You will need to substitute as appropriate if you are using some other prefix.)

There is also a *profile* for users of the Bourne shell (it assumes you have a version of the Bourne shell which has functions). Users should add

```
. /usr/local/share/aegis/profile
```

to the end of their *.profile* file for the recommended aliases. (This *profile* assumes that users are using a Bourne shell which understands functions.)

The */usr/local/com/aegis/state* file contains pointers to "system" projects. Users may add their own project pointers (to their own projects) by putting a search path into the *AEGIS_PATH* environment variable. The system part is always automatically appended by *Aegis*. The default, already set by the */usr/local/lib/aegis/cshrc* file, is *$HOME/lib/aegis*. Do not create this directory, *Aegis* is finicky and wants to do this itself.

Where projects reside is completely flexible, be they system projects or user projects. They are not kept under the */usr/local/com/aegis* directory, this directory only contains pointers. (Note that these building instructions assume you are using the default */usr/local* as the install prefix. You will need to substitute as appropriate if you are using some other prefix.)

Web Interface

If you have a Web server, you may like to install the Aegis web interface. You do this by copying the *aeget* script from */usr/local/bin/aeget* into your web server's *cgi-bin* directory. There is a *aeget.instal* helper script, if you don't know where your web server's *cgi-bin* directory is.

You may prefer to use a symbolic link, as this will be more stable across Aegis upgrades. However, this requires a corresponding *follow-symlinks* setting in your web server's configuration file. (Use the *aeget.instal* −*s* option.)

You may need to wrap *aeget* with a script which sets the *AEGIS_PATH* environment variable, if you want it to be able to see more projects than just the global projects. You may also need to set the *PATH* environment variable, if you don't have the Aegis install path in the default path.

(Note that these building instructions assume you are using the default */usr/local* as the install prefix. You will need to substitute as appropriate if you are using some other prefix.)

PRINTED MANUALS

This distribution contains the sources to all of the documentation for *Aegis*, however the simplest way to get the documentation is by anonymous FTP; PostScript files of the User Guide and Reference Manual are available from the FTP sites listed in the README file.

The Reference Manual contains the README and BUILDING files, as well as all of the section 1 and section 5 manual pages. The Reference Manual is about 200 pages long.

The User Guide contains information about how to use Aegis, including a fully worked example. The User Guide is about 100 pages long.

TIME SYNCHRONIZATION

The *Aegis* program uses time stamps to remember whether various events have happened and when. If you are using *Aegis* in a networked environment, typically a server and data-less workstations, you need to make absolutely sure that all of the machines agree about the time.

If possible, use the time daemon. Otherwise, use *rdate*(8) via *cron*(8) every hour or less.

GETTING HELP

If you need assistance with *Aegis*, please do not hesitate to contact the author at

Peter Miller <pmiller@opensource.org.au>

Any and all feedback is welcome.

When reporting problems, please include the version number given by the

```
% aegis -version
aegis version 4.25.D510
...
%
```

command. Please run this command to get the exact number, do not send the text of this example.

Runtime Checking

In the *common/main.h* file, there is a define of *DEBUG* in comments. If the comments are removed, extensive debugging is turned on. This causes some performance loss, but performs much run-time checking and adds the **−TRAce** command line option.

When the **−TRAce** command line option is followed by one or more file names, it turns on execution traces in those source files. It is usually best to place this on the end of the command line so that names of the files to be traced are not confused with other file names or strings on the command line.

Problem Reports

If you send email to the author, please include the following information:

1. The type of UNIX
   The author will need to know the brand and version of UNIX you are using, or if it is not UNIX but something else. The output of "uname −sr" is usually sufficient (but not all systems have it).
2. The Version Number
   In any information you send, please include the version number reported in the *common/patchlevel.h* file, or `'aegis -vers'` if you can get it to compile.
3. The Archive Site
   When and where you obtained this version of *Aegis*. If you tell me nothing else, tell me this (and, hopefully, why you did nothing else).
4. Unpacking
   Did you have problems unpacking *Aegis*? This probably isn't a problem with the .tar.Z distribution, but you could have obtained a shar format copy.
5. Building
   Did you have problems building *Aegis*? This could have been the instructions included, it could have been the configure script, it could have been the Makefile, or anything else.
6. Testing, Non-Set-Uid
   Did you have problems with the tests? You could have had problems running them, or some of them could have failed. If some tests fail but not others, please let me know *which* ones failed, and include the fact that *Aegis* was **not** set-uid-root at the time. The −k option to *make* can be useful if some tests fail but not others.
7. Testing, Set-Uid-Root
   Did you have problems with the tests when *Aegis* was set-uid-root? You could have had problems running them, or some of them could have failed. If some tests fail but not others, please let me know *which* ones failed, and include the fact that *Aegis* was set-uid-root at the time.
8. Installation
   Did you have problems installing *Aegis*? This could have been the instructions, or anything else.

At this point it would probably be a very good idea to print out the manual entries and read them carefully. You will also want to print a copy of the User Guide; if you don't have groff, there should be a PostScript copy at the archive site. It is a known flaw that the User Guide is incomplete, contributions are most welcome.

9. The Example Project
   After reading the User Guide, it is often useful to manually run through the example in chapter 3. You will need to do more than one change, hopefully several; the first change is not representative of the system. Did you manually do the example? Did you find flaws in the User Guide or manual entries?
10. Using Aegis
    Did you have problems using *Aegis*? This is a whole can of worms. If possible, include a shell script similar to the tests which accompany *Aegis*, which reproduces the bug. Exit code 1 on failure (bug), exit code 0 on success (for when bug is fixed).
11. The Source Code
    Did you read the code? Did you write some code? If you read the code and found problems, fixed them, or extended *Aegis*, these contributions are most welcome. I reserve the right to modify or reject such contributions.

The above list is inclusive, not exclusive. Any and all feedback is greatly appreciated, as is the effort and interest required to produce it.

LICENSE

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

WINDOWS-NT

It is possible to build Aegis for Windows-NT. I have only done this using the Cygnus freeware CygWin32 system, though it may be possible with other Unix porting layers also.

Caveat

This document only describes a **single user** port of Aegis to Windows NT.

Aegis depends on the underlying security provided by the operating system (rather than re-invent yet another security mechanism). However, in order to do this, Aegis uses the POSIX *seteuid*(2) system call, which has no direct equivalent on Windows NT. This makes porting difficult. **Single user** ports are possible (*e.g.* using Cygwin (http://www.cygwin.com/), but are not usually what folks want.

Compounding this is the fact that many sites want to develop their software for both Unix and Windows NT simultaneously. This means that the security of the repository needs to be guaranteed to be handled in the same way by both operating systems, otherwise one can act as a "back door" into the repository. Many sites do not have the same users and permissions (sourced from the same network register of users) on both Unix and Windows NT, making the mapping almost impossible even if the security models did actually correspond.

Most sites using Aegis and Windows NT together do so by running Aegis on the Unix systems, but building and testing on the NT systems. The work areas and repository are accessed via Samba or NFS.

The Source

You need to FTP the Cygwin system from RedHat. It can be found at

http://www.cygwin.com/

and then follow the links. The original version used was B20.1, but more recently 1.1.7 has been used.

It is *absolutely essential* to run the *mkpasswd* and *mkgroup* commands, otherwise Aegis will give fatal errors about unknown users and groups. See the Cygwin README for instructions.

Mounting Things

You need to mount a directory onto `/tmp`, or lots of things, and especially *bash*(1), don't work. If you are in a heavily networked environment, like me, you need to know that using a networked drive for `/tmp` just doesn't work. I have no idea why. Use

mount C:/temp /tmp

instead. (Or some other local drive.)

Just a tip for all of you who, like me, know Unix much better than you know Windows-NT: the left-hand mount argument needs to be specified with a drive letter (*e.g.* `C:`*) rather than with a double slash (e.g. not* `//C`*) unless its Windows-NT name starts with* \\.

You need to follow the install instructions about */bin/sh*, otherwise shell scripts that start with `#!/bin/sh` don't work, among other things. This includes the `./configure` script, and the scripts it writes (*e.g.* `config.status`).

You will want to mount your various network drives onto the same places they appear on your Unix hosts. This way you don't need to learn two names for all your files.

Mounts persist across Cygwin sessions. They are stored in a registry file somewhere. You will not need to do all this every time!

Too Much Administrator

If you have administrator privilege on your Windows NT box, you need to get rid of it. (Have a second admin account instead.) This is because Windows NT will make the files belong to the wrong user for files on *some* partitions, like */tmp*. (This took me days to work out!) This confuses both Aegis *and* RCS.

If you get weird "Permission denied" errors from amazingly unlikely causes, this is probably why.

Before You Start

There are several pieces of software you need before you can build Aegis on Cygwin.

I'm going to keep mentioning "your local GNU mirror". You can find
GNU at `http://www.gnu.org`, however you are better off using a local mirror, and these are scattered around the globe. Follow the "mirrors" link on their front page to find your closest mirror. Also, it's often a good idea to configure these packages with the "−with-gnu-gettext" option to their ./configure commands.

Do not use WinZip to unpack the tarball. It has a nasty habit of
turning all of the newlines into CRLFs. This will confuse *lots* of utilities, especially GNU Groff. Use the "*tar xzf aegis-4.25.tar.gz*" command from within Cygwin.

Make sure the Cygwin you are using has GNU Groff 1.15 or later
(use a "groff −v" command). Grab and install the latest from your local GNU mirror, if it isn't.

util-linux
You need to get GNU rx, but to make it work you have to find a *tsort* command, so that GNU rx's *./configure* script works. Try the latest copy of `system/misc/util-linux-?.?.tar.gz` from the `metalab.unc.edu` Linux archive (or a mirror). Simply build and install *misc-utils/tsort.c* by hand.

GNU rx Once you have *tsort* installed, you will be able to get GNU rx configured. Get a copy from your local GNU mirror.

zlib You need to grab a copy of *zlib*; the same source as works for Unix will work for Cygwin. It will install as a static library.

GNU diffutils
You need GNU diffults, because when you come to configure GNU RCS (next) it would otherwise complain about a stupid *diff* and a missing *diff3* command. The *install-sh* script is broken, so you'll need to do the final step in the install by hand.

GNU RCS
All of Aegis' tests assume RCS is present. Also, you are going to need *something* for a history tool. The *install-sh* script is broken, so you'll need to do the final step in the install by hand.

Configure

The configure and build step should be the same as for Unix, as described above. All the problems I encountered were to do with getting the mounts just right. (But expect it to be dog slow compared to Linux or FreeBSD on the same box.)

Sharutils
You need the *uudecode* command for several of the tests, and this may be found in the GNU Sharutils package. You can get a copy from your local GNU mirror.

The configure step is almost the same as for Unix. I know you are itching to get typing, but read through to the install section before you configure anything.

```
bash$ ./configure
...lots of output...
bash$
```

Build

The build step is exactly the same as for Unix, and you shouldn't notice any difference...

```
bash$ make
bash$
```

Test

The tests are run in the same way as the Unix tests, but you don't need to run the set-uid-root variants, because no such thing exists under Windows NT.

```
bash$ make sure
...lots of output...
Passed All Tests
bash$
```

Unfortunately, it isn't that simple. There are a number of things you will see go wrong...

- Several tests fail because *ed* isn't there.
- Several tests fail because *ci* (RCS 5.7) dumps core much too often for my liking.
- A couple of tests fail because they don't expect the ".exe" extension on executable files.
- A couple of tests (notably, the *aedist* tests) fail because of the CRLF *vs* NL dichotomy. This means that the expected results don't match, not that it isn't working.

Despite all the bad news, the vast majority of tests pass, and the others have good excuses.

Install

Installing the software works as usual, though you need to make some choices right at the start (I told you to read this all the way through first). If you want to use the "*/usr/local*" prefix (or any other install prefix) you mount it right at the start. For anything other than the "*/usr/local*" default prefix, you also needed to give a "**-prefix=***blahblah" argument to the configure script, right at the start.*

```
bash$ make install
...lots of output...
bash$
```

// vim: set ts=8 sw=4 et :

NAME

aegis − project change supervisor

SYNOPSIS

aegis *function* [*option*...]
aegis −Help

DESCRIPTION

The *aegis* program is a transaction base software configuration management system. It is used to supervise the development and integration of changes into projects.

FUNCTIONS

The following functions are available:

−Build
The *aegis −Build* command is used to build a project. See *aeb*(1) for more information.

−Change_Attributes
The *aegis −Change_Attributes* command is used to modify the attributes of a change. See *aeca*(1) for more information.

−Change_Directory
The *aegis −Change_Directory* command is used to change directory. See *aecd*(1) for more information.

−Change_Owner
The *aegis −Change_Owner* command is used to facilitate reassignment of the developer of a change in the *being developed* state. See *aechown*(1) for more information.

−CLone
The *aegis −CLone* command is used to exactly replicate a change, usually on another branch. See *aeclone*(1) for more information.

−CoPy_file
The *aegis −CoPy_file* command is used to copy a file into a change. See *aecp*(1) for more information.

−CoPy_file_Undo
The *aegis −Copy_File_Undo* command is used to remove a copy of a file from a change. See *aecpu*(1) for more information.

−DELta_NAme
The *aegis −DELta_NAme* command is used to add a symbolic name to a project delta. See *aedn*(1) for more information.

−Develop_Begin
The *aegis −Develop_Begin* command is used to begin development of a change. See *aedb*(1) for more information.

−Develop_Begin_Undo
The *aegis −Develop_Begin_Undo* command is used to cease development of a change. See *aedbu*(1) for more information.

−Develop_End
The *aegis −Develop_End* command is used to complete development of a change. See *aede*(1) for more information.

−Develop_End_Undo
The *aegis −Develop_End_Undo* command is used to recall a change for further development. See *aedeu*(1) for more information.

−DIFFerence
The *aegis −DIFFerence* command is used to find differences between development directory and baseline. See *aed*(1) for more information.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−Integrate_Begin
The *aegis −Integrate_Begin* command is used to begin integrating a change. See *aeib*(1) for more information.

−Integrate_Begin_Undo
The *aegis −Integrate_Begin_Undo* command is used to cease integrating a change. See *aeibu*(1) for more information.

−Integrate_Fail
The *aegis −Integrate_Fail* command is used to fail a change integration. See *aeifail*(1) for more information.

−Integrate_Pass
The *aegis −Integrate_PASS* command is used to pass a change integration. See *aeipass*(1) for more information.

−List
The *aegis −List* command is used to list interesting things. See *ael*(1) for more information.

−MoVe_file
The *aegis −MoVe_file* command is used to change the name of a file as part of a change. See *aemv*(1) for more information.

−MoVe_file_Undo
The *aegis −MoVe_file_Undo* command is used to undo a change to the name of a file as part of a change. See *aemvu*(1) for more information.

−New_Administrator
The *aegis −New_Administrator* command is used to add new administrators to a project. See *aena*(1) for more information.

−New_BRanch
The *aegis −New_BRanch* command is used to add a new branch to a project. See *aenbr*(1) for more information.

−New_BRanch_Undo
The *aegis −New_BRanch_Undo* command is used to remove a new branch from a project. See *aenbru*(1) for more information.

−New_Change
The *aegis −New_Change* command is used to add a new change to a project. See *aenc*(1) for more information.

−New_Change_Undo
The *aegis −New_Change_Undo* command is used to remove a new change from a project. See *aencu*(1) for more information.

−New_Developer
The *aegis −New_Developer* command is used to add new developers to a project. See *aend*(1) for more information.

−New_File
The *aegis −New_File* command is used to add new files to a change. See *aenf*(1) for more information.

−New_File_Undo
The *aegis −New_File_Undo* command is used to remove new files from a change. See *aenfu*(1) for more information.

−New_Integrator
The *aegis −New_Integrator* command is used to add new integrators to a project. See *aeni*(1) for more information.

−New_Project
The *aegis −New_Project* command is used to create a new project to be watched over by aegis. See *aenpr*(1) for more information.

−New_Project_Alias
The *aegis −New_Project_Alias* command is used to create a new project alias. See *aenpa*(1) for more information.

−New_ReLeaSe
The *aegis −New_ReLeaSe* command is used to create a new project from an existing project. See *aenrls*(1) for more information.

−New_ReViewer
The *aegis −New_ReViewer* command is used to add new reviewers to a project. See *aenrv*(1) for more information.

−New_Test
The *aegis −New_Test* command is used to add a new test to a change. See *aent*(1) for more information.

−New_Test_Undo
The *aegis −New_Test_Undo* command is used to remove new tests from a change. See *aentu*(1) for more information.

−Project_Attributes
The *aegis −Project_Attributes* command is used to modify the attributes of a project. See *aepa*(1) for more information.

−Remove_Administrator
The *aegis −Remove_Administrator* command is used to remove administrators from a project. See *aera*(1) for more information.

−Remove_Developer
The *aegis −Remove_Developer* command is used to remove developers from a project. See *aerd*(1) for more information.

−ReMove_file
The *aegis −ReMove_file* command is used to add files to be deleted to a change. See *aerm*(1) for more information.

−ReMove_file_Undo
The *aegis −Remove_File_Undo* command is used to remove files to be deleted from a change. See *aermu*(1) for more information.

−Remove_Integrator
The *aegis −Remove_Integrator* command is used to remove integrators from a project. See *aeri*(1) for more information.

−ReMove_PRoject
The *aegis −ReMove_PRoject* command is used to remove a project. See *aermpr*(1) for more information.

−Remove_Project_Alias
The *aegis −Remove_Project_Alias* command is used to remove a project alias. See *aerpa*(1) for more information.

−Remove_ReViewer
The *aegis −Remove_ReViewer* command is used to remove reviewers from a project. See *aerrv*(1) for more information.

−RePorT
The *aegis −RePorT* command is used to generate reports from aegis' database. These reports may be written by users, or be distributed with aegis.

−Review_Fail
The *aegis −Review_Fail* command is used to fail a change review. See *aerfail*(1) for more information.

−Review_Begin
The *aegis −Review_Begin* command is used to begin to review a change. See *aerb*(1) for more information.

−Review_Begin_Undo
The *aegis −Review_Begin_Undo* command is used to stop reviewing a change. See *aerbu*(1) for more information.

−Review_Pass
The *aegis −Review_PASS* command is used to pass a change review. See *aerpass*(1) for more information.

−Review_Pass_Undo
The *aegis −Review_Pass_Undo* command is used to rescind a change review pass. See *aerpu*(1) for more information.

−Test
The *aegis −Test* command is used to run tests. See *aet*(1) for more information.

−VERSion
The *aegis −VERSion* command is used to get copyright and version details. See *aev*(1) for more information.

All function selectors are case insensitive. Function selectors may be abbreviated; the abbreviation is the upper case letters. Function selectors must appear as the first command line argument.

Notification
Many *aegis* commands are capable of notification that they have been run. The individual commands document those specific to them. For documentation on the various configurable notifications, see *aepconf*(5) and *aepattr*(5) for more information.

OPTIONS

The following options are available to all functions. These options may appear anywhere on the command line following the function selectors.

−LIBrary *abspath*
This option may be used to specify a directory to be searched for global state files and user state files. (See *aegstate*(5) and *aeustate*(5) for more information.) Several library options may be present on the command line, and are search in the order given. Appended to this explicit search path are the directories specified by the *AEGIS_PATH* environment variable (colon separated), and finally, */usr/local/lib/aegis* is always searched. All paths specified, either on the command line or in the *AEGIS_PATH* environment variable, must be absolute.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

The following options are available to *most* functions. These options may appear anywhere on the command line following the function selectors.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more

information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Change *number*
This option may be used to specify a particular change within a project. When no **−Change** option is specified, the *AEGIS_CHANGE* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default change field (see *aeuconf* (5) for more information). If that does not exist, when the user is only working on one change within a project, that is the default change number. Otherwise, it is an error.

−Change *project***.C***number*
As a shortcut, it is possible to combine the **−Project** and **−Change** options into a single option.

−Change *branch***.C***number*
Several functions accept a **−BRanch** option; it is possible to combine the **−BRanch** and **−Change** options in a single option. (This intentinally has the same form as the *${version}* substitution output for incomplete changes.)

−Change *branch***.D***number*
Several functions accept both the **−BRanch** and **−Delta** options (or **−BRanch** and **−Change-From-Delta** options); it is possible to combine them in a single option. (This intentinally has the same form as the *${version}* substitution output for completed changes.)

−Change *project***.D***number*
It is possible to combine the **−Project** and **−Change-From-Delta** options as a single option.

−Change *UUID*
Each completed change is assigned a globallay unique identifier (UUID). You can specify a change by its 36-character UUID, or any unambiguous leading predix of the UUID (it must be at least 4 characters, and not look like a number).

Listings

The following options are available to all listings. These options may appear anywhere on the command line following the function selectors.

−PAGer The output of listings and help is piped through the pager command given in the PAGER environment variable (or *more* if not set). This is the default if the command is in the foreground, and the output is a TTY. This option may be used to override any preference specified in the *aeuconf* (5) file.

−No_PAGer
This option may be used to ensure that the output of listings and help is not piped through a pager command. This is the default if the command is in the background, or if the output is not a TTY. This option may be used to override any preference specified in the *aeuconf* (5) file.

−Page_Length *number*
This option may be used to set the page length of listings. The default, in order of preference, is obtained from the system, from the *LINES* environment variable, or set to 24 lines.

−Page_Width *number*
This option may be used to set the page width of listings and error messages. The default, in order of preference, is obtained from the system, from the *COLS* environment variable, or set to 79 characters.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-UNFormatted
This option may be used with most listings to specify that the column formatting is not to be performed. This is useful for shell scripts.

-Page-Header
This option requests that page headings be present in listings and reports. This is the default.

-No-Page-Header
This option requests that page headings be omitted from listings and reports.

Abbreviations

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

The *aegis* command understands the following environment variables:

AEGIS_PATH
A colon-separated list of library directories. See the **−LIBrary** option for a description how this environment variable is used.

AEGIS_PROJECT
Names a default project. See the **−Project** option for a description how this environment variable is used.

AEGIS_CHANGE
Specifies a default change. See the **−Change** option for a description how this environment variable is used.

AEGIS_FLAGS
This environment variable is used to hold *aeuconf*(5) information, and over-rides the settings in the users *.aegisrc* file. This is intended to be used within the tests distributed with aegis, but can also be of use within some shell scripts.

AEGIS_THROTTLE
Specifies the number of seconds to delay execution within commands which set time stamps. This is intended to be used within the tests distributed with aegis, but can also be of use within some shell scripts.

AEGIS_AUTOMOUNT_POINTS
A colon-separated list of directories which the automounter may use to mount file systems. Use with extreme care, as this distorts Aegis' idea of the shape of the filesystem.

This feature assumes that paths below the automounter's mount directory are echoes of paths without it. *E.g.* When `/home` is the trigger, and `/tmp_mnt/home` is where the on-demand NFS mount is performed, with `/home` appearing to processes to be a symlink.

This is the behavior of the Sun automounter. The AMD automounter is capable of being configured in this way, though it is not typical of the examples in the manual. Nor is it typical of the out-of-the-box Linux AMD configuration in many distributions.

COLS Specifies the page width for errors and listings. See the **−Page_Width** option for a description how this environment variable is used.

EDITOR
: Specifies the program use to edit files when the **−Edit** or **−Edit_BackGround** options are used. (See also the *VISUAL* environment variable.) Defaults to *vi* if not set. See the *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

LINES
: Specifies the page length for listings. See the **−Page_Length** option for a description how this environment variable is used.

PAGER
: Specifies the program to use to view listings and help. Not used if output is to a file or a pipe. Defaults to *more* if not set.

VISUAL
: Specifies the program use to edit files when the **−Edit** option is used. (See also the *EDITOR* environment variable.) Defaults to *vi* if not set. See the *visual_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

AEGIS_DATADIR
: Overrides the datadir as specified at *configure* invocation. Useful mainly for testing.

When commands are executed by Aegis, it ensures that the AEGIS_PROJECT, AEGIS_CHANGE, AEGIS_ARCH, LINES and COLS environment variables are set appropriately. The project configuration file's *project_specific* field is also consulted, looking for value's whose name starts with "setenv:" and sets the corresponding environment variable. All of the substitutions described by *aesub*(5) are available. For example: specifying a PATH and a SEARCH_PATH to be used for all commands may be set as follows:

```
project_specific =
[
  {
    name = "setenv:PATH";
    value = "/usr/bin:/bin";
  },
  {
    name = "setenv:SEARCH_PATH";
    value = "${search_path}";
  },
];
```

As many environment variables as desired may be specified in this way.

SEE ALSO

aegis(5) aegis file format syntax

aecattr(5)
: change attributes file format

aecstate(5)
: change state file format

aedir(5) directory structures

aegstate(5)
: aegis state file format

aepattr(5)
: project attributes file format

aepconf(5)
: project configuration file format

aepstate(5)
: project state file format

aer(5) report script language definition

aesub(5)
available command substitutions

aeuconf(5)
user configuration file format

aeustate(5)
user state file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae-cvs-ci − checkin a change set to CVS

SYNOPSIS

ae-cvs-ci *project-name change-number*

DESCRIPTION

The *ae-cvs-ci* command is used to check an Aegis change set into CVS.

This script is a short wrapper around the *ae-repo-ci*(1) command.

This is usually used in the *integrate pass notify command* project attribute, as in

```
integrate_pass_notify_command =
   "$bin/ae-cvs-ci $project $change";
```

EXIT STATUS

The *ae-cvs-ci* command will exit with a status of 1 on any error. The *ae-cvs-ci* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae-repo-ci − redundant repository checkin

SYNOPSIS

ae-repo-ci −Project *name* **−Change** *number* **−REPOsitory** *type* [*option*...]
ae-repo-ci −Help
ae-repo-ci −VERSion

DESCRIPTION

The *ae-repo-ci* command is used to redundantly commit an Aegis change set into a parallel repository.

Integrate Pass Notify Command

The intended use for the ae-repo-ci command is as an *integrate_pass_notify_command* (see *aepa*(1) for more information) to do a redundant checkin of a change set into a second parallel repository.

For example, if you were using CVS, the project attribute would look something like this:

```
integrate_pass_notify_command =
    "$bin/ae−repo−ci −repo cvs "
    " −p $project −c $change";
```

You may also need to specify the module, if the module name is not the same as the project name.

Commit Messages

You are able to control the commit message, by using the `ae-repo-ci:commit-message` attribute in the *project_specific* field of the project configuration file.

The default is as if the following entry were present:

```
project_specific = [
  {
    name = "ae-repo-ci:commit-message";
    value = "$version − ${change brief_description}";
  } ];
```

All of the *aesub*(5) substitutions are available.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DIRectory *path*
This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

-Help
This option may be used to obtain more information about how to use the *ae-repo-ci* program.

−List This option may be used to obtain a list of supported repository types.

−MODule *name*
This option may be used to specify which module is to be checked out. If not set, it defaults to the trunk project name (*i.e.* the project name without any branch or version numbers).

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user’s *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REPOsitory *type*

This option is used to specify the repository type for the checkin. Known repository types are:

cvs Concurrent version System. You will need to set the CVSROOT environment variable appropriately, and the **−module** option will be relative to it.

svn

Subversion. You must specify the complete URL with the **−module** option.

The following field in the `project_specific` field of the project configuration file (see *aepconf*(5) for more information) are relevant:

svn:username

If present, the `−username` command line option will be added to *svn*(1) command lines, with this value.

svn:password 8n

If present, the `−username` command line option will be added to *svn*(1) command lines, with this value.

These options can help when you can't convince Subversion to use the correct authorization any other way.

This option must be specified, there is no default. The **−list** option may be used to obtain an up-to-date list of supported repository types.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *ae-repo-ci* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *ae-repo-ci* command will exit with a status of 1 on any error. The *ae-repo-ci* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeca(1) how to change project attributes

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae-sccs-put − put sccs version

SYNOPSIS

ae-sccs-put −y*comment* **−G***input-file history-file*

DESCRIPTION

The *ae-sccs-put* command is used to commit changes to an SCCS file. It insulates against a number of SCCS's quirks, and maps to Aegis' expectations better than using the SCCS commands directory in the history commands in the project *aegis.conf* configuration file.

The file comments *must* be specified on the command line.

The source file *must* be specified on the command line.

It is expected that there is not lock current in the history file. This is consistent with Aegis' use of its history tool.

The history file need to exist yet. It will be created (with the *sccs admin* command) if it does not.

OPTIONS

The following options are understood:

−G*source-file*

This option must be used to specify the source file to be checked into the history.

−y*comment*

This option must be used to specify the comment to be attached to the file history. You probably need to use quotes to insulate the white space in the comment.

EXIT STATUS

The *ae-sccs-put* command will exit with a status of 1 on any error. The *ae-sccs-put* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

ae-sccs-put version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae_c − set change number

SYNOPSIS

ae_c *change-number*

DESCRIPTION

The *ae_c* command is an alias used to set the AEGIS_CHANGE environment variable. No checking of the argument is performed.

This can make changing the change you are working on quick and simple.

SEE ALSO

aegis(1) For information on environment variables.

ae_p(1) Set project name.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae_diff2htm − wraps the diff2html command

SYNOPSIS

ae_diff2htm

DESCRIPTION

DESCRIPTION

The *ae_diff2htm* script wraps the diff2html command if available or otherwise falls back to a simple context diff.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

ae_p − set project name

SYNOPSIS

ae_p *project-name*

DESCRIPTION

The *ae_p* command is an alias used to set the AEGIS_PROJECT environment variable. No checking of the argument is performed.

This can make changing projects quick and simple.

SEE ALSO

aegis(1) For information on environment variables.

ae_c(1) Set change number.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeannotate − annotated source file listing

SYNOPSIS

aeannotate [*option*...] *filename*
aeannotate −Help
aeannotate −List
aeannotate −VERSion

DESCRIPTION

The *aeannotate* command is used to produce an annotated listing of the named source file.

The columns specified by the user (see the **−column** option, below) are used of the left hand side of the output. Two additional columns are always added: the line number and the source code.

If no columns are specified, the default columns are

```
−column    '${change date %Y−%m}'    Date      7
−column    '$version'                Version   9
−column    '${change developer}'     Who       8
```

The *$version* string always contains enough information to reproduce the entire project baseline at the time of the delta. The first portion is the project branch, and the second portion (following the 'D') is the delta number; use these to form the −branch and −delta options for an *aecp*(1) command.

At the end of the listing, accumulated statistics are presented, correlated to the unique columns values see in the listing.

OPTIONS

The following options are understood:

−COLumn *formula* **[** *heading* **][** *width* **]**
This option may be used to specify columns you wish to see in the output. The formula is in the for of an *aesub*(5) string. The heading is a string to be used as the column heading; defaults to the formula if not specified. The width is the width of the columns; defaults to 7 if not specified.

−File_Statistics
This option causes file statistics to be appended. This lists the number of lines in the file were changed at the same time as another file. For example, this allows you to see tests associated with source files, and *vice versa*.

-Help
This option may be used to obtain more information about how to use the *aeannotate* program.

−Diff_Option *string*
This option may be used to pass addition arguments to the diff commands that is run between each delta of the file. Use with caution: poor choice of options can render aeannotate inoperable, or yield meaningless results. Probably the best use of this option is to pass the **−b** option, to ignore white space changes, because this ignores the vast majority of cosmetic formatting changes, showing you the content changes instead. The **−i** option, to ignore case, can also be useful for case-insensitive languages.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case

letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aeannotate* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

EXIT STATUS

The *aeannotate* command will exit with a status of 1 on any error. The *aeannotate* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

EXAMPLES

If you wanted to list only the year against the lines of the file, use this column specification:

```
−column ’${change date %Y}’ Year 4
```

If you wanted to list the developer and the reviewer against the lines of the file (commonly called a “blame” listing) use this column specification:

```
−column ’${change developer}’ Develop. 8
−column ’${change reviewer}’ Reviewer 8
```

If you wanted to see the change cause of each line, use this column specification:

```
−column ’$version’ Version 9
−column ’${change cause}’ Cause 20
```

All of the *aesub*(5) substitutions are available, however only the *${change ...}* variants are particularly useful.

To see only content changes, and ignore changes in indentation (assuming you are using GNU diff), use this command:

aeannotate −diff-opt −b *filename*

COPYRIGHT

aeannotate version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis build − build a change

SYNOPSIS

aegis −Build [*option*...][*filename*...]
aegis −Build −List [*option*...]
aegis −Build −Help

DESCRIPTION

The *aegis −Build* command is used to build a project. The project configuration file is consulted for the appropriate build command, and that command is executed (see the *build_ command* and *integration_-build_command* fields in *aepconf*(5) for more information.) Output of the command is automatically logged to the *aegis.log* file at the root of the development directory tree. The build command will be executed with its current directory being the root of the development directory, irrespective of there the *aegis −Build* command was executed.

If the change is in the *being integrated* state, references to the development directory, above, should be read as the integration directory. Integration build commands are executed with the user and group set to the project's owning user and group. That is, it is not necessary for an integrator to log in as someone else, the project account for instance, in order to do an integration.

No Build Required

It is possible to configure your project so that no build is required. To do this, set the following

```
build_command = "exit 0";
```

in the project configuration file.

Process Side Effects

This command will cancel any test registrations, because building the project logically invalidates them. If the project configuration file was deleted, any diff registration will also be canceled.

Notification

The actions of the command are controlled by the *build_ command* and *integration_build_command* fields of the project *config* file. See *aepconf*(5) for more information.

File Action Adjustment

When this command runs, it first checks the change files against the projects files. If there are inconsistencies, the file actions will be adjusted as follows:

create — If a file is being created, but another change set is integrated which also creates the file, the file action in the change set still being developed will be adjusted to "modify".

modify — If a file is being modified, but another change set is integrated which removes the file, the file action in the change set still being developed will be adjusted to "create".

remove — If a file is being removed, but another change set is integrated which removes the file, the file will be dropped from the change set still being developed.

PARTIAL BUILD

If files are named on the command line, these files are appended to the build command. This is known as a partial build. Partial builds are not legal in the *being integrated* state, but can often be useful in the *being developed* state. Partial builds are not recorded in the change status, because builds are decoupled from aegis it is not possible for aegis to know if any set of partial builds is equivalent to a full build.

Warning: no change state lock is taken for a partial build, only a baseline read lock.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file

named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

SYMBOLIC LINKS

Many dependency maintenance tools, and indeed some compilers, have little or no support for include file search paths, and thus for the concept of the two-level directory hierarchy employed by Aegis. (It becomes multi-level when Aegis' branching functionality is used.) To allow these tools to be used, Aegis provides the ability to maintain a set of symbolic links between the development directory of a change and the baseline of a project, so it appears to these tools that all of the project's files are present in the development directory.

Project Configuration

The *development_directory_style* field of the project configuration file controls the appearance of the development directory. See *aepconf*(5) for more information.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
    during_build_only = true;
};
```

the user never sees the symbolic links, because they are added purely for the benefit of the dependency maintenance tool during the execution of the *aeb*(1) command.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
};
```

(the other will default to false) the symbolic links will be created at develop begin time (see *aedb*(1) for more information) and also maintained by each *aeb*(1) invocation. Note that the symbolic links are only maintained at these times, so project integrations during the course of editing change sourec files may leave the symbolic links in an inconsistent state until the next build.

When files are copied from the baseline into a change, using the *aecp*(1) command, the symbolic link pointing into the baseline, if any, will be removed before the file is copied.

Note: Using this functionality in either form has implications for how the rules file of the dependency maintenance tool is written. Rules must *remove* their targets before creating them (usually with an *rm −f* command) if you use any of the link sub-fields (both hard links and symbolic links). This is to avoid attempting to write the result on the symbolic link, which will point at a read-only file in the project baseline. This is similar to the same requirement for using the *link_integration_directory* field of the project configuration file.

User Configuration

There is a *symbolic_link_preference* field in the user configuration file (see *aeuconf*(5) for more information). This controls whether *aeb*(1) will verify the symbolic links before the build (default) or whether it will assume they are up-to-date. (This field is only relevant if *development_directory__style.-source_file_symlink* is true.)

For medium-to-large projects, verifying the symbolic links can take as long as the build itself. Assuming the symbolic links are up-to-date can be a large time-saving for these projects. It may be advisable to review your choice of DMT in such a situation.

The *aedb*(1) command **does not** consult this preference. Thus, in most situations, the symbolic links will

be up-to-date when the build is performed. The only Aegis function which may result in the symbolic links becoming out-of-date is the integration of another change, as this may alter the presence or absence of files in the baseline. In this situation, the default *aeb*(1) action is to ignore the user preference and the verify symbolic links.

There are two command line options which modify *aeb*(1) behavior further: the **−Verify-Symbolic-Links** option says to verify the symbolic links; and the **−Assume-Symbolic-Links** option says to assume the symbolic links are up-to-date. In each case the option over-rides the default and the user preference.

It is possible to obtain behaviour similar to Tom Lord'a Arch by using a setting such as:

```
development_directory_style =
{
    source_file_link = true;
    source_file_symlink = true;
};
```

It is possible to obtain behaviour similar to CVS by using a setting such as:

```
development_directory_style =
{
    source_file_copy = true;
};
```

There are many more possible configurations of the *development_directory_style*, usually with helpful build side-effects. See *aepconf*(1) and the *Depenedency Maintenance Tool* chapter of the User Guide for more information.

The symbolic link command line options and preferences apply equally to hard links and file copies (the names have historical origins).

THE BASELINE LOCK

The baseline lock is used to ensure that the baseline remains in a consistent state for the duration of commands which need to read the contents of files in the baseline.

The commands which require the baseline to be consistent (these include the *aeb*(1), *aecp*(1) and *aed*(1) commands) take a baseline *read* lock. This is a non-exclusive lock, so the concurrent development of changes is not hindered.

The command which modifies the baseline, *aeipass*(1), takes a baseline *write* lock. This is an exclusive lock, forcing *aeipass*(1) to block until there are no active baseline read locks.

It is possible that one of the above development commands will block until an in-progress *aegis −Integrate_PASS* completes. This is usually of short duration while the project history is updated. The delay is essential so that these commands receive a consistent view of the baseline. No other integration command will cause the above development commands to block.

When aegis' branch functionality is in use, a read (non-exclusive) lock is taken on the branch baseline and also each of the "parent" baselines. However, a baseline write (exclusive) lock is only taken on the branch baseline; the "parent" baselines are only read (non-exclusive) locked.

METRICS

Aegis is capable of recording metrics as part of the file attributes of a change. This allows various properties of files to be recorded for later trend analysis, or other uses.

The specific metrics are not dictated by Aegis. It is expected that the integration build will create a metrics file for each of the source files the change. These metrics files must be in the format specified by *aemetrics*(5).

The name of the metrics file defaults to "*filename*,**S**", however it may be varied, by setting the *metrics_filename_pattern* field of the project *config* file. See *aepconf*(5) for more information.

If such a metrics file exists, for each source file in a change, it will be read and remembered at integrate pass time. If it does not exist, Aegis assumes there are no relevant metrics for that file, and proceeds silently; it is not an error.

OPTIONS

The following options are understood:

name=value

Command line arguments of this form are assumed to be variable assignments for the build tool. They are passed through unchanged. They imply a partial build.

−BAse_RElative

This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative

This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−MINImum

This option may be used to request a source-only *development_directory_style*. This is useful if you want to simulate something like *aeib −minimum* in the development directory. This option is only meaningful if *development_directory_style* is being used. If the change is in the *being integrated* state, and the developer specified **−MINImum** when issuing the *aegis −Integrate_Begin* command, then this option is set by default.

-Not_Logging

This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links

This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeb 'aegis −b \!* −v'
sh$ aeb(){aegis −b "$@" −v}

ERRORS

It is an error if the change is not assigned to the current user.
It is an error if the change is not in one of the *being developed* or *being integrated* states.
It is an error if a partial build is requested and the change is in the *being integrated* state.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aedb(1) begin development of a change

aecp(1) file copy also takes a baseline read lock (non-exclusive)

aeib(1) begin integration of a change

aeipass(1)
integrate pass takes a baseline write lock (exclusive)

aet(1) run tests

aemetrics(5)
metrics values file format

aepconf (5)
project configuration file format

aeuconf (5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aebisect − search for a delta which changed project behaviour

SYNOPSIS

aebisect [*option*...] [−Branch *branch1*] −DELta *delta1* [−Branch *branch2*] −DELta *delta2* −− *command* [command_args]

DESCRIPTION

The **aebisect** command is used to determine when in a project history some property or behavior changed. It does this by means of a bisection search through the inventory of deltas. The user must specify starting and ending deltas, which may be in historical branches of the project.

For each delta tested in the search, **aebisect** sets up a development directory, builds the project, and then runs the specified *command* in the development directory. By iteration, **aebisect** finds two consecutive deltas where the return code of *command* changed.

Note: **aebisect** can take considerable CPU effort, since it (normally) does a full build from scratch for each delta tested.

OPTIONS

The following options are understood:

−Help
Show usage information.

−Project *project−name*
specify the project (otherwise done via the **AEGIS_PROJECT** environment variable)

−Change *change−number*
specify the change to use for the processing (otherwise done via the **AEGIS_CHANGE** environment variable). The change must be in the *awaiting_development* state; this ensures a correct environment for building and testing.

−Branch *branch−extension*
specify the branch for one of the deltas. Defaults to the baseline branch of the project. Use **−b −** (single dash) to specify the trunk. Branch specifiers must precede the corresponding delta specifiers.

−Logfile *logfile*
specify where normal output goes; defaults to *$HOME/aebisect.log*.

−Verbose
produce more diagnostic information (both logfile and standard output).

−Keep
do not delete working files, which are in a temporary directory. Warning: these may be voluminous!

−DIRectory *path*
specify a development directory to use for building and testing.

−Minimum
use the **−minimum** option for the builds.

−Nobuild
skip the build steps. This option is useful if the test command only involves source files. (Consider using **aeannotate**(1) instead.)

−Zero_only
treat all test result codes other than 0 as equivalent.

DIAGNOSTICS

Normally, exit status is 0 if consecutive deltas are found to bracket a change in the test command result. Exit status is 1 if errors are detected in arguments. Exit status is 2 if a subordinate command fails (possibly leaving the development directory in an uncertain state) or if the test behavior is found to be inconsistent with bisection search.

SIGNALS

aebisect will stop on INT, QUIT, and TERM signals, probably leaving the development directory in an uncertain state.

EXAMPLE

Suppose a bug was introduced by development on project *foo−4.5*, sometime between version 1.2.D003 and 4.5.D006, and you have written an Aegis test script for the bug (see **aent**(1)), called */wrk/test/00/t0007a.sh*, taking an argument for system architecture. Then the following should isolate the change which introduced the bug:

```
% aenc −p foo−4.5 −c 20 −file caf
% aebisect −p foo−4.5 −c 20 −b 1.2 −del 3 −b 4.5 −del 6 \
  −− sh /wrk/test/00/t0007a.sh linux−i486
```

Note that the full path for the test script is specified, since the command is executed in a development directory.

BUGS

aebisect depends on **aecp −delta** for historical reconstructions. This can be problematic.

It is possible for a build to fail: derived files from the baseline may poison the build, or there may have been changes in the system infrastructure since the old deltas were integrated. In such cases, **aebisect** exits. The user may then snoop around the development directory, fix something, rebuild, perform the test, and use the logfile to see how to proceed. Remember to **aedbu** when done.

In some situations the problem may be cured by an additional step between **aedb** and **aeb**. A command to be interposed may be defined via the environment variable **AEBISECT_DB_HOOK**; this command is executed after **aecp**, so it may be used to patch source files — see the script source for details.

COPYRIGHT

NAME

aebuffy − watch for changes

SYNOPSIS

aebuffy [*project-name*]

DESCRIPTION

The *aebuffy* command is used to watch for changes which the current user may be able to act upon. These include changes being developed by the user, and changes which could be reviewed or integrated by the user.

If you don't use the *project-name* command line option, you need to set the AEGIS_PROJECT environment variable, or the default_project field of the *.aegisrc* file before you invoke this command. This is especially important if you launch it from your X11 session start-up file.

Double clicking on a change will invoke the *tkaer*(1) command for that change. This does not work for changes in the *awaiting development* and *completed* states, but works for all other states.

Use the "q" key to quit.

At the moment it can only watch one project. If you are good at Tcl/Tk, improvements are most welcome.

COPYRIGHT

aebuffy version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis change attributes − modify the attributes of a change

SYNOPSIS

aegis −Change_Attributes −File *attr-file* [*option*...]
aegis −Change_Attributes −Edit [*option*...]
aegis −Change_Attributes −Fix_ARchitecture
aegis −Change_Attributes *name=value*
aegis −Change_Attributes −List [*option*...]
aegis −Change_Attributes −Help
aegis −Change_Attributes −UUID *string* [*option*...]

DESCRIPTION

The *aegis −Change_Attributes* command is used to set, edit or list the attributes of a change.

The output of the **−List** variant is suitable for use as input at a later time.

See *aecattr*(5) for a description of the file format.

The *name=value* form of the command may be used when you wish to add or modify change set attributes. If an attribute is already present, it will be modified; if there is more than one attribute with the same name, only the first will be modified. The *name+=value* form will always append the pair.

Example

When you edit the file, you will see something like this:

```
brief_description = "login(1) is too fussy";
description = "When users type their password "
    "incorrectly, after three times the login(1) "
    "program should assume they have forgotten "
    "their password and automatically reset it "
    "for them.";
cause = external_enhancement;
attribute =
[
    {
        name = "bugzilla";
        value = "666";
    },
    {
        name = "customer-priority";
        value = "high";
    },
    {
        name = "marketing-priority";
        value = "urgent-panic-headless-chicken";
    },
    {
        name = "engineering-priority";
        value = "after-heat-death-of-universe";
    }
];
```

Note the semicolons, you need to get them right. Edit the fields you want to change. When you quit the editor, they will be updated.

Known Attribute Names

While many of the anticipated used of change attributes are to allow projects to attach their own specialized data to change sets, Aegis also uses some attributes for its own purposes (and arguably, should always have done so to maximize forwards compatibility across Aegis upgrades).

aeget:inventory:hide
: boolean. If true, this change set does not appear in *aeget*(1)'s change set inventory pages, used by *aedist −replay* to decide what to download and apply. Think of it as a "local only" flag.

aeget-inventory-hide
: Do not show this change set in the file inventory. See *aeget*(1) for more information.

aegis:history_get_command
: Used when reconstructing file history, and the history tool has been changed at some point in the past.

aemakegen:debian:accepted
: Set to true when a debian package upload has succeeded, and the BTS tells you the bug fixes have been accepted.

foreign-copyright
: boolean. If true, none of the files in this change set will not be checked by the *aede-policy*(1) `copyright` validation.

integrate-begin-hint
: Suggest options when integrating. See *aeib*(1) for more information.

original-uuid
: This is set by *aedist −receive* when an incoming change set is changed before it can end development. There may be more than one. The *aeget*(1) inventory "all" page will show these additional UUIDs, used by the *aedist −pending* command..

aegis:history_get_command
: This is set by *aeipass* when integrating a change. See the *CHANGING HISTORY TOOL* section of *aepconf*(5) for more information.

Universal Unique Identifier

Each change set is assigned a unique universal identifier (UUID) at *integrate pass* time. This serves to identify the change across all sites when a geographically distributed development model is being used. This may be exploited to rapidly determine which change sets need to be downloaded.

The **−Universal_Unique_IDentifier** option is used by the *aedist*(1) and *aepatch*(1) commands to set the UUID of a change set. It should not be used by developers.

Using Change Attributes in Scripts

There are several ways you can use the attributes of an Aegis change in a shell script:

aereport(1)
: The report generator is able to access change attributes. You can then have the report generator print the necessary data.

aesub(1) Many change attributes can be accessed via the *aesub*(5) command substitutions, and printed using the *aesub*(1) command.

aeca −l The list option of the *aeca*(1) command may be used to print the values of all editable change attributes. It can be groped using *grep*(1) or *awk*(1), or similar.

aexml(1)
: It is possible to get a great deal of information in XML format, including change attributes. This format can be parsed by a variety of packages.

Use the method best suited to your particular needs.

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Description_Only
This option may be used to specify that only the change description is the subject of this command. It will be presented as plain text, without any of the quotes or escapes present when this command is not used.

−Edit
Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround
Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*
Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

−Fix_ARchitecture
This option may be used to replace change change's architecture list with all of the mandatory architectures from the project configuration file, plus any of the optional architectures that match the current machine. May not be used with −file or −edit options.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Universal_Unique_IDentifier *string*
This option may be used to set the UUID of change change.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeca 'aegis −ca \!* −v'
sh$ aeca(){aegis −ca "$@" −v}

ERRORS

It is an error if the current user is not an administrator of the specified project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

tkaeca(1)
GUI interface to the *aeca*(1) command.

aecattr(5)
change attributes file format

aecstate(5)
change state file format

aepa(5) modify the attributes of a project

aesub(5)
available command substitutions

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis change directory − change directory

SYNOPSIS

aegis −Change_Directory [*option*...][*relative-path*]
aegis −Change_Directory −List [*option*...]
aegis −Change_Directory −Help

DESCRIPTION

The *aegis −Change_Directory* command is used to obtain a path to change directory to. If the *relative-path* is supplied, this will be added to the output.

This command is usually used to calculate an argument for *cd*(1), however it can also be used to obtain an absolute path for change and project files.

OPTIONS

The following options are understood:

-BaseLine
This option may be used to specify that the project baseline is the subject of the command.

−BRanch *number*
This option may be used to specify a different branch for the origin file, rather than the baseline. (See also **−TRunk** option. Please Note: the **−BRanch** option does not take a project name, just the branch number suffix.

−GrandParent
This option may be used to specify the grandparent branch (one up from the current branch) for the origin file, rather than the baseline. (The −grandparent option is the same as the "−branch .." option.)

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Development_Directory
This option is used to specify that the development directory is the subject of the command. This is only useful for a change which is in the *being integrated* state, when the default is the integration directory.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-TRunk
This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the "−branch −" option.)

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aecd 'cd `aegis −cd \!* −v`'
sh$     aecd(){cd `aegis −cd "$@" −v`}
```

ERRORS

It is an error if the specified change is not in a state where it has a directory to change to.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aedb(1) begin development of a change

aeib(1) begin integration of a change

aerpass(1)
pass review of a change

aerfail(1)
fail review of a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis change owner − set change owner

SYNOPSIS

aegis −Change_Owner *change-number user-name* [*option*...]
aegis −Change_Owner −Help
aegis −Change_Owner −VERSion

DESCRIPTION

The *aegis −Change_Owner* command is used to reassign a change from one developer to another.

A new development directory is created for the change in the new developers default area (see *aedb*(1) for more information) and the change files are copied across. Derived files are ignored, so a new build will be required. The old development directory will be deleted.

This command must be performed by a project administrator, and the new assignee must be a developer.

Warning: capricious use of this command will alienate developers very rapidly.

Notification

This command mimics many of the actions of the *aebdu*(1) and *aedb*(1) command. In particular, it invokes the *develop_begin_undo_command* and *develop_begin_command* of the project *config* file. See *aepconf*(5) for more information.

Development Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-Interactive

Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more

information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−User *name*
This option is used to specify the user who is to develop the change.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aechown 'aegis −chown \!* −v'
sh$     aechown(){aegis −chown "$@" −v}
```

ERRORS

It is an error if the user issuing the command is not a project administrator.
It is an error if the change is not in the *being developed* state.
It is an error if the user given is not a developer.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis clEan − clean files from development directory

SYNOPSIS

aegis −CLEan [*option*...]
aegis −CLEan −Help
aegis −VERSion

DESCRIPTION

The *aegis −CLEan* command is used to remove all files which are not change source files from a development directory. This can be used to obtain a "clean" development directory before a final build, to ensure that a change is ready to end development. A new build will be required.

This command is only allowed in the "*being developed*" state, and only the change's developer may issue it. It may not be applied to branches.

All symbolic links will be removed from the development directory, even if *remove_symlinks_after_build = false* in the project *config* file. The symbolic links will be re-installed, if *create_symlinks_before_build = true*. This is to ensure that the symlinks are accurate, and that unnecessary ones are removed.

All special device files, pipes and sockets will be removed. These files cannot be source files, and it is expected that the following build will restore them.

All derived files created by previous builds of the change will be removed. It is expected that the following build will recreate them. Any temporary files you may have created in the development directory will also be removed.

The *develop_begin_command* in the project configuration file (see *aepconf*(5) for more information) will be run, if there is one. The *change_file_command* will be run, if there is one. The *project_file_command* will be run, if there is one.

You will be warned if any of the files are out-of-date and need to be merged. You will be warned if any files need to be differenced.

SYMBOLIC LINKS

Many dependency maintenance tools, and indeed some compilers, have little or no support for include file search paths, and thus for the concept of the two-level directory hierarchy employed by Aegis. (It becomes multi-level when Aegis' branching functionality is used.) To allow these tools to be used, Aegis provides the ability to maintain a set of symbolic links between the development directory of a change and the baseline of a project, so it appears to these tools that all of the project's files are present in the development directory.

Project Configuration

The *development_directory_style* field of the project configuration file controls the appearance of the development directory. See *aepconf*(5) for more information.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
    during_build_only = true;
};
```

the user never sees the symbolic links, because they are added purely for the benefit of the dependency maintenance tool during the execution of the *aeb*(1) command.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
};
```

(the other will default to false) the symbolic links will be created at develop begin time (see *aedb*(1) for more information) and also maintained by each *aeb*(1) invocation. Note that the symbolic links are only

maintained at these times, so project integrations during the course of editing change sourec files may leave the symbolic links in an inconsistent state until the next build.

When files are copied from the baseline into a change, using the *aecp*(1) command, the symbolic link pointing into the baseline, if any, will be removed before the file is copied.

Note: Using this functionality in either form has implications for how the rules file of the dependency maintenance tool is written. Rules must *remove* their targets before creating them (usually with an *rm −f* command) if you use any of the link sub-fields (both hard links and symbolic links). This is to avoid attempting to write the result on the symbolic link, which will point at a read-only file in the project baseline. This is similar to the same requirement for using the *link_integration_directory* field of the project configuration file.

User Configuration

There is a *symbolic_link_preference* field in the user configuration file (see *aeuconf*(5) for more information). This controls whether *aeb*(1) will verify the symbolic links before the build (default) or whether it will assume they are up-to-date. (This field is only relevant if *development_directory__style.-source_file_symlink* is true.)

For medium-to-large projects, verifying the symbolic links can take as long as the build itself. Assuming the symbolic links are up-to-date can be a large time-saving for these projects. It may be advisable to review your choice of DMT in such a situation.

The *aedb*(1) command **does not** consult this preference. Thus, in most situations, the symbolic links will be up-to-date when the build is performed. The only Aegis function which may result in the symbolic links becoming out-of-date is the integration of another change, as this may alter the presence or absence of files in the baseline. In this situation, the default *aeb*(1) action is to ignore the user preference and the verify symbolic links.

There are two command line options which modify *aeb*(1) behavior further: the **−Verify-Symbolic-Links** option says to verify the symbolic links; and the **−Assume-Symbolic-Links** option says to assume the symbolic links are up-to-date. In each case the option over-rides the default and the user preference.

It is possible to obtain behaviour similar to Tom Lord’a Arch by using a setting such as:

```
development_directory_style =
{
    source_file_link = true;
    source_file_symlink = true;
};
```

It is possible to obtain behaviour similar to CVS by using a setting such as:

```
development_directory_style =
{
    source_file_copy = true;
};
```

There are many more possible configurations of the *development_directory_style*, usually with helpful build side-effects. See *aepconf*(1) and the *Depenedency Maintenance Tool* chapter of the User Guide for more information.

The symbolic link command line options and preferences apply equally to hard links and file copies (the names have historical origins).

Notification

The notification commands that would be run by the *aecp*(1), *aedb*(1), *aenf*(1), *aent*(1) and *aerm*(1) commands are run, as appropriate. The *project_file_command* is also run, if set. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−TOuch
This option may be used to request that each change source file have its last-modified time-stamp be updated to the current time. This is the default. Derived files and other non-source file are left alone.

−No_TOuch
This option may be used to request that the last-modified time-stamp of each source file be left unmodified.

−MINIMum
This option may be used to request a minimum set of symbolic links, when the *create_symlinks_to_baseline* functions are being used. This is useful if you want to simulate something like *aeib −minimum* in the development directory. This option is not meaningful if symbolic links are not being used.

This option also says not to remove normal files which occlude project source files. This is a common technique used to temporarily over-ride project source files. The *"aecp −read-only" command would have been more appropriate.*

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis clone − make an exact copy of a change

SYNOPSIS

aegis −CLone [*option*...] *change-number* [change-number]
aegis −CLone −Help
aegis −CLone −VERSion

DESCRIPTION

The *aegis −CLone* command is used to create exact replicas of changes. This is of most use when a change need to be applied to several parallel branches.

One change number *must* be supplied. This is the change to be replicated. If any branch options are given (see below) the mandatory change number applies to the branch specified. If no branch is specified, the change applies to the project (implicit or explicit).

If the optional second change number is supplied, this is the change number to be created to hold the replica; if it is not supplied, the next available change number will be used.

If the change to be replicated has been completed, the appropriate file revisions will be extracted from history; otherwise the files will be copied from the development directory of the change to be copied. Be warned: if a file in the change which was cloned subsequently changes, those changes *will not* automagically be tracked. It is best if changes are cloned at a stable time, such as one of the states after develop end, or even after integrate pass.

Development Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

WHITEOUT

Aegis provides you with what is often called a "view path" which indicates to development tools (compilers, build systems, *etc*) look first in the development directory, then in the branch baseline, and so on up to the trunk baseline.

The problem with view paths is that in order to remove files, you need some kind of "whiteout" to say "stop looking, it's been removed."

When you user the *aerm*(1) or *aemv*(1) commands, this means "add information to this change which will remove the file from the baseline when this change is integrated". *I.e.* while the change is in the *being developed* state, the file is only "removed" in the development directory − it's still present in the baseline, and will be until the change is successfully integrated.

When you use the *aerm*(1) or *aemv*(1) commands, Aegis will create a 1K file to act as the whiteout. It's contents are rather ugly so that if you compile or include the "removed" file accidentally, you get a fatal error. This will remind you to remove obsolete references.

When the change in integrated, the removed file is *not* copied/linked from the baseline to the integration directory, and is *not* copied from the development directory. At this time it is physically gone (no whiteout). It is assumed that because of the error inducing whiteout all old references were found and fixed while the change was in the *being developed* state.

File Manifests

When generating list of files to be compiled or linked, it is important that the file manifest be generated from information known by Aegis, rather than from the file system. This is for several reasons:

(a) Aegis knows exactly what (source) files are where, whereas everything else is inferring Aegis' knowledge; and

(b) looking in the file system is hard when the view path is longer that 2 directories (and Aegis' branching method can make it arbitrarily long); and

(c) The whiteout files, and anything else left "lying around", will confuse any method which interrogates the file system.

The easiest way to use Aegis' file knowledge is with something like an *awk*(1) script processing the Aegis file lists. For example, you can do this with *make*(1) as follows:

```
# generate the file manifest
manifest.make.inc: manifest.make.awk
    ( aegis −l cf −ter ; aegis −l pf −ter ) | \
    awk −f manifest.make.awk > manifest.make.inc
# now include the file manifest
include manifest.make.inc
```

Note: this would be inefficient of you did it once per directory, but there is nothing stopping you writing numerous assignments into the *manifest.make.inc* file, all in one pass.

It is possible to do the same thing with Aegis' report generator (see *aer*(1) for more information), but this is more involved than the *awk*(1) script. However, with the information "straight from the horse's mouth" as it were, it can also be much smarter.

This file manifest would become out-of-date without an interlock to Aegis' file operations commands. By using the *project-file_command* and *change_file_command* fields of the project *config* file (see *aepconf*(5) for more information), you can delete this file at strategic times.

```
/* run when the change file manifest is altered */
change_file_command = "rm −f manifest.make.inc";
/* run when the project file manifest is altered */
project_file_command = "rm −f manifest.make.inc";
```

The new file manifest will thus be re-built during the next *aeb*(1) command.

Options and Preferences

There is a **−No-WhiteOut** option, which may be used to suppress whiteout files when you use the *aerm*(1) and *aemv*(1) commands. There is a corresponding **−WhiteOut** option, which is usually the default.

There is a *whiteout_preference* field in the user preferences file (see *aeuconf*(5) for more information) if you want to set this option more permanently.

Whiteout File Templates

The *whiteout_template* field of the project *config* file may be used to produce language-specific error files. If no whiteout template entry matches, a very ugly 1KB file will be produced − it should induce compiler errors for just about any language.

If you want a more human-readable error message, entries such as

```
whiteout_template =
[
    {
        pattern = [ "*.[ch]" ];
        body = "#error This file has been removed.";
    }
];
```

can be very effective (this example assumes *gcc*(1) is being used).

If it is essential that *no* whiteout file be produced, say for C source files, you could use a whiteout template such as

```
whiteout_template =
[
    { pattern = [ "*.c" ]; }
];
```

because an absent *body* sub-field means generate no whiteout file at all.

You may have more than one whiteout template entry, but note that the order of the entries is important. The first entry which matches will be used.

Notification

The notification commands that would be run by the *aecp*(1), *aedb*(1), *aenf*(1), *aent*(1) and *aerm*(1) commands are run, as appropriate. The *project_file_command* is also run, if set. See *aepconf*(5) for more information.

Cloning and Merging

When you use *aeclone*(1) to clone a change set, and then integrate one of the two change sets, you will observe that Aegis says that the files of the un-integrated change are now out-of-date.

If you run *aem*(1) to bring the out-of-date files back up-to-date, *fmerge*(1) and some (but not) all other merging tools, it signals just about everything as a conflict, even though both alternatives are identical.

The problem is that two changes making identical edits to the same place in the same file are a logical conflict, even if not an actual conflict, and it takes a human to figure out the difference. Think of a shopping list: the ensuite needs more soap, and so does the main bathroom. The second "soap" on the merge of the two shopping lists isn't a duplicate, you really do need two boxes of soap. Sometimes edits of source files are the same: sometimes the logical conflict is resolved by applying both identical edits, not just one.

This is just the *fmerge*(1) command being more conservative than RCS's *merge*(1) command.

The easiest way to deal with this common situation it to run an

```
aecpu −unchanged
```

command *before* you run the *aem*(1) merge command, and you will have less grief. It's also worth remembering that Aegis stashes the original file with a `,B` suffix (B for backup) so you can simply

```
mv fubar,B fubar
```

if you know that all of the conflicts are logical conflicts.

OPTIONS

The following options are understood:

−BRanch *number*

This option may be used to specify a different branch for the origin file, rather than the baseline. (See also **−TRunk** option. Please Note: the **−BRanch** option does not take a project name, just the branch number suffix.

−GrandParent

This option may be used to specify the grandparent branch (one up from the current branch) for the origin file, rather than the baseline. (The −grandparent option is the same as the "−branch .." option.)

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use 'pwd' to make an absolute path, it usually gives the wrong answer.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−WhiteOut
This option may be used to request that deleted files be replaced by a “whiteout” file in the development directory. The idea is that compiling such a file will result in a fatal error, in order that all references may be found. This is usually the default.

−No_WhiteOut
This option may be used to request that no “whiteout” file be placed in the development directory.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined change number. Useful for writing scripts.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user’s *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TRunk
This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the “−branch −” option.)

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

ERRORS

It is an error if the current user is not an administrator of the project. (In some cases it is possible for developers of a project to create changes, see *aepattr*(5) for more information.)

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenc(1) Create a new change.

aeca(1) modify the attributes of a change

aena(1) add a new administrator to a project

aepa(1) modify the attributes of a project

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aecomp − compare two changes

SYNOPSIS

aecomp *number* [*number*]

DESCRIPTION

The *aecomp* script is used to compare two changes, using *tkdiff*(1) to display the changes. If you give one change on the command line, the other change is determined in the usual way. If you give two changes, those are the two compared. Both changes must be in the *being developed* state (it's only a script, after all).

Basically, aecomp allows you to specify two project/change pairs (ie the compared changes don't have to be in the same branch or project). aecomp attempts to use defaults (for project and change number) where possible. As a minimum, it needs a single change number as an option. A list of files which are common between the two changes is constructed and presented. Double clicking on any of the file names will *tkdiff* the two versions (ie the files in each change).

It is useful after you have used *aeclone*(1), then modified a change and subsequently are wondering what on earth you did. Files are considered to be "common" if they have the same name. In the case of a file which has been moved, it's original filename is used (the diff of course takes place against the new file name).

AUTHOR

Scott Finneran <scottf@lucent.com>

NAME

aecomplete − command completion

SYNOPSIS

aecomplete *cmd-name incomplete-word previous-word*

DESCRIPTION

The *aecomplete* command is used to perform command completion for shells.

See *bash*(1) for more information about Bash command completion, and how this command is expected to be executed.

At present, this is the only shell supported. The code has been written to be extensible, should other shells have programmable completion by external programs.

EXIT STATUS

The *aecomplete* command will exit with a status of 1 on any error. The *aecomplete* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aecomplete version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis copy file − copy a file into a change

SYNOPSIS

aegis −CoPy_file [*option*...] *filename*...
aegis −CoPy_file −INDependent [*option*...] *filename*...
aegis −CoPy_file −List [*option*...]
aegis −CoPy_file −Help

DESCRIPTION

The *aegis −CoPy_file* command is used to copy a file into a change. The named files will be copied from the baseline into the development directory, and added to the list of files in the change. The version of files copied from the baseline is remembered.

This command may be used to copy tests into a change, not just source files. Tests are treated just like any other source file, and are subject to the same process.

Warning: If there are files in the development directory of the same name they will be overwritten by this command.

You may also name directories. All of the source files in the directories named, and all directories below them, will be copied from the baseline into the development directory, and added to the list of files in the change.

When copying files explicitly, it is an error if the file is already part of the change. When you name a directory, all of the source files in the project below that directory are copied, except any which are already in the change. It is an error if none of the files implicitly named by the directory can be used.

If you want to change a copied source file to be executable (shell scripts, for example) then you simply use the normal *chmod*(1) command; the reverse to make it not executable. If any of the file's executable bits are set at *aede*(1) time the file is remembered as executable and all execute bits (minus the project's umask) will be set by subsequent *aecp*(1) commands.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because adding another file logically invalidates them. If the project configuration file was added, any diff registration will also be canceled.

When the change files are listed (*aegis −List Change_Files −TERse*) the copied files will appear in the listing. When the project files are listed with an explicit change number (*aegis −List Project_Files −TERse −Change* `N`) none of the change's files, including the copied files, will appear in the terse listing. These two features are very helpful when calling aegis from within a DMT to generate the list of source files.

THE BASELINE LOCK

The baseline lock is used to ensure that the baseline remains in a consistent state for the duration of commands which need to read the contents of files in the baseline.

The commands which require the baseline to be consistent (these include the *aeb*(1), *aecp*(1) and *aed*(1)

commands) take a baseline *read* lock. This is a non-exclusive lock, so the concurrent development of changes is not hindered.

The command which modifies the baseline, *aeipass*(1), takes a baseline *write* lock. This is an exclusive lock, forcing *aeipass*(1) to block until there are no active baseline read locks.

It is possible that one of the above development commands will block until an in-progress *aegis −Integrate_PASS* completes. This is usually of short duration while the project history is updated. The delay is essential so that these commands receive a consistent view of the baseline. No other integration command will cause the above development commands to block.

When aegis' branch functionality is in use, a read (non-exclusive) lock is taken on the branch baseline and also each of the "parent" baselines. However, a baseline write (exclusive) lock is only taken on the branch baseline; the "parent" baselines are only read (non-exclusive) locked.

TEST CORRELATIONS

The "aegis −Test −SUGgest" command may be used to have aegis suggest suitable regression tests for your change, based on the source files in your change. This automatically focuses testing effort to relevant tests, reducing the number of regression tests necessary to be confident that you have not introduced a bug.

The test correlations are generated by the "aegis −Integrate_Pass" command, which associates each test in the change with each source file in the change. Thus, each source file accumulates a list of tests which have been associated with it in the past. This is not as exact as code coverage analysis, but is a reasonable approximation in practice.

The *aecp*(1) and *aenf*(1) commands are used to associate files with a change. While they do not actively perform the association, these are the files used by *aeipass*(1) and *aet*(1) to determine which source files are associated with which tests.

Test Correlation Accuracy

Assuming that the testing correlations are accurate and that the tests are evenly distributed across the function space, there will be a less than *1/number* chance that a relevant test has not been run by the "aegis −Test −SUGgest *number*" command. A small amount of noise is added to the test weighting, so that unexpected things are sometimes tested, and the same tests are not run every time.

Test correlation accuracy can be improved by ensuring that:

- Each change should be strongly focused, with no gratuitous file inclusions. This avoids spurious correlations.
- Each item of new functionality should be added in an individual change, rather than several together. This strongly correlates tests with functionality.
- Each bug should be fixed in an individual change, rather than several together. This strongly correlates tests with functionality.
- Test correlations will be lost if files are moved. This is because correlations are by name.

The best way for tests to correlate accurately with source files is when a change contains a test and exactly those files relating to the functionality under test. Too many spurious files will weaken the usefulness of the testing correlations.

Notification

The *copy_file_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

File Action Adjustment

When this command runs, it first checks the change files against the projects files. If there are inconsistencies, the file actions will be adjusted as follows:

create — If a file is being created, but another change set is integrated which also creates the file, the file action in the change set still being developed will be adjusted to "modify".

modify — If a file is being modified, but another change set is integrated which removes the file, the file action in the change set still being developed will be adjusted to "create".

remove If a file is being removed, but another change set is integrated which removes the file, the file will be dropped from the change set still being developed.

OPTIONS

The following options are understood:

−as-needed
Usually it is an error if a file is already in a change set, and is redundantly added to the change set again. This option says to ignore such files.

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−BRanch *number*
This option may be used to specify a different branch for the origin file, rather than the baseline. (See also **−TRunk** option. Please Note: the **−BRanch** option does not take a project name, just the branch number suffix.

−GrandParent
This option may be used to specify the grandparent branch (one up from the current branch) for the origin file, rather than the baseline. (The −grandparent option is the same as the "−branch .." option.)

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning "when the branch started".

−DELta_Date *string*
This option may be used to specify a particular date and time in the project's history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−INDependent
This option is used to specify that the copy is to be run independent of any particular change. The files will be copied relative to the current directory (but see the −Output-Directory **option).**

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Output *filename*
This option may be used to specify an output file of a file being copied from the baseline. Only one baseline file may be named when this option is used. The file name "−" is understood to mean the standard output. This option does not add the file to the set of change files. *No locks* are taken when this option is used, not even the baseline read lock.

−Output-Directory *path*
This option may only be used with the **−INDependent** option, to specify the output directory for the copied files, rather than the current directory. The directory will be created if it does not exist already.

-OverWriting
This option may be used to force overwriting of files. The default action is to give an error if an existing file would be overwritten.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Read_Only
This option may be used to specify that the file is to be used to insulate the change from the baseline. The user does not intend to edit the file. These files must be uncopied before development may end.

−REScind
This option may be used to rescind (roll back) a completed change. The change to rescind (roll back) is specified in the usual way, with one of the −delta options.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-TRunk
This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the "−branch −" option.)

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case

letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aecp ’aegis −cp \!* −v’
sh$ aecp(){aegis −cp "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is already in the change and the **−OverWrite** option is not specified.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

EXAMPLES

Here are some simple examples. Remember that most commands are relative to the current directory, even though these examples assume you are at the base of the development directory tree.

Copy Whole Project

To copy the whole project into your change, use the command

aecp .

The trailing dot is part of the command, it means "the currect directory and everything below it". This works for any directory in your project source tree, if you want to be more seledctive.

Prroduce Earlier Project Version

You you wish to exactly reproduces the sources for an earlier version of your project, you need to know the edelta number (use *ael proj-history* to find it). Then use this command:

aecp −delta *n* .

where *n* is the delta number from the project history. Again, the trailing dot is part of the command. By using the *$version* substitution (see *aesub*(5) for more information) you can embed this delta number into your program before distributing it.

It is also possible to give a previous change number, instead, using this command:

aecp −delta-from-change *n* .

where *n* is the change number of interest. Again, the trailing dot is part of the command.

Rescind a Change

When you need to rescind (back out) a completed change, it will probably have been some time ago, so you need to know the delta number or change number. Use this command:

aecp −delta *n* −rescind .

where *n* is the delta number of interest. All of the other −delta variantrs also work, so if you know the change number, you can be more selective about which files to copy:

aecp −delta-from-change *n* ‘aegis −l cf −ter −c *n*‘

where *n* is the change number of interest. This only copies the files which were in the offending change.

SEE ALSO

aeb(1) build also takes a baseline read lock (non-exclusive)

aecpu(1)
reverse action of aecp

aedb(1) begin development of a change

aedn(1) assign a name to a delta

aeipass(1)
integrate pass takes a baseline write lock (exclusive)

aemv(1) rename a file as part of a change

aenf(1) add a new file to a change

aerm(1) add files to be deleted to a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis copy file undo − reverse action of aecp

SYNOPSIS

aegis −CoPy_file_Undo [*option*...] *filename*...
aegis −CoPy_file_Undo −List [*option*...]
aegis −CoPy_file_Undo −Help

DESCRIPTION

The *aegis −CoPy_file_Undo* command is used to delete files previously copied into a change. The named files will be removed from the list of files in the change.

The file is deleted from the development directory unless the **−Keep** option is specified. The **−Keep** option should be used with great care, as you can confuse tools such as *make*(1) by leaving these files in place.

You may name a directory to delete from the change all files in that directory tree previously copied into the change, other files in the tree will be ignored. It is an error if there are no relevant files.

Branch *vs* Change

The *aecpu*(1) command may only be applied to a change. If you wish to perform the same operation on a branch, use the *aemt*(1) command, through the agency of a change.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because deleting a file logically invalidates them. If the project configuration file was deleted, any diff registration will also be canceled.

The difference file (`,D`) will also be removed, however any DMT derived files (e.g a `.o` file from a `.c` file) will not be removed. This is because aegis is decoupled from the DMT, and cannot know what these derived file may be called. You may need to delete derived files manually.

Notification

The *copy_file_undo_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−BAse_RElative
: This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
: This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-Interactive
Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Read_Only
This option may be used to specify that only insulation files are to be uncopied. If you specify **−UNChanged** as well, only unchanged insulation files will be uncopied.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−Verify_Symbolic_Links
This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−UNChanged
Examine the named files to see if they are unchanged. Only remove the unchanged files from the change, and leave the files which have changed. If no files are named on the command line all change files are checked.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aecpu 'aegis −cpu \!* −v'
sh$ aecpu(){aegis −cpu "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy files into a change

aemt(1) make branch files transparent

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aecvsserver − serve CVS client protocol against Aegis projects

SYNOPSIS

aecvsserver server
aecvsserver pserver
aecvsserver −VERSion

DESCRIPTION

The *aecvsserver* command is used to serve the CVS client protocol. The repository, of course, is stored within Aegis.

The server works by retrieving file contents from locations within Aegis change sets and repositories. When necessary, appropriate *aegis*(1) commands are executed by the server to fulfill the requests.

This code is still experimental. At the present time only a limited number of CVS commands are understood. If you would like to extend this code, contributions are welcome. The following commands are thought to work at this time: add, admin, checkout, commit, init, remove, update.

server

To use the server, you will need to set the following environment variables:

```
CVSROOT=:ext:hostname/aegis
CVS_RSH=ssh
CVS_SERVER=aecvsserver
```

pserver

It is also possible to use *aecvsserver* as a cvs pserver, with all the usual caveats about how insecure this access method is, because it transmits the password *almost* in the clear. The root and modules are as above.

MODULES

The CVS concept of modules is mapped onto Aegis concept of projects and changes. The special CVSROOT administrative module is simulated.

Projects as Modules

Each Aegis project appears to the CVS client as a module; the module's name is the same as the Aegis project's name. This type of module isn't immediately useful except for the *cvs export* command, or to perform a read-only *cvs checkout* command.

You can't commit to a project-named module. This because Aegis requires all operations which would change the repository to be performed through a change set.

It is theoretically possible to code *aecvsserver* to create a change (via *aenc*(1) and *aedb*(1) commands), then add the necessary files (via *aenf*(1) and *aecp*(1) commands), then build (via the *aeb*(1) command), then test (via the *aet*(1) command), and finally to end development of the change (via the *aede*(1) command). As the CVS protocol documentation says

> "The protocol makes it possible for updates to be atomic with respect to checkins; that is, if someone commits changes to several files in one cvs command, then an update by someone else would either get all the changes, or none of them. The current cvs server can't do this, but that isn't the protocol's fault."

This code is yet to be written. Contributions welcome.

The protocol, however, doesn't make it particularly easy, either. The semantics of the `Modify` request change depending on whether it is *followed* by the `commit` request or the `update` request.

Changes as Modules

Each Aegis change set also appears to the CVS client as a module; it's name is *project*`.C`*number*. All *cvs add* commands, *cvs remove* commands, *cvs update* commands and *cvs commit* commands are performed against the change set, not directly to the baseline. It is necessary for the change set to already exist, and once you have run the *cvs commit* command, it will the be necessary to use the *aede*(1) command and the rest of the usual Aegis process.

Once a change is no longer in the *being developed* state, it cannot be changed via *aecvsserver*(1) and you

will need to create a new Aegis change set, and then *cvs checkout* a new client-side work area.

Please note: if you are experimenting with the interface via *cvs −d :fork:/aegis* or similar, the work area you create **must** be outside the Aegis change set's development directory.

CVSROOT

The CVSROOT module's contents are synthesized from Aegis meta-data. You can't add or modify files in this module; you need to administer Aegis directly with *aegis*(1) commands.

EXIT STATUS

The *aecvsserver* command will exit with a status of 1 on any error. The *aecvsserver* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aecvsserver version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis difference − find differences between a change and the baseline

SYNOPSIS

aegis −DIFFerence [*filename*...] [*option*...]
aegis −DIFFerence −List [*option*...]
aegis −DIFFerence −Help

DESCRIPTION

The *aegis −DIFFerence* command is used to generate difference listings between source files in the the development directory and the baseline. The purpose is to enable reviewers to find each and every edit performed on the source files. The difference listings will be placed into files named for the sources files but with an additional "`,D`" suffix.

The command used to perform the differences is specified in the *diff_command* field of the project configuration file (see *aepconf*(5) for more information).

It is possible to configure a project to omit the diff step as unnecessary, by the following setting:

```
diff_command = "exit 0";
```

This disables all generation, checking and validation of difference file for each change source file. The merge functions of the *aediff*(1) command are unaffected by this setting.

Please note that the *history_content_limitation* field of the project configuration file does **not** apply to the *diff_command* field.

If no files are named on the command line, all files in the change will be differenced.

You may name a directory on the command line, and all files in the change in that directory tree will be differenced.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Notification

The actions of the command are controlled by the *diff_command* and *merge_command* fields of the project *config* file. See *aepconf*(5) for more information.

THE BASELINE LOCK

The baseline lock is used to ensure that the baseline remains in a consistent state for the duration of commands which need to read the contents of files in the baseline.

The commands which require the baseline to be consistent (these include the *aeb*(1), *aecp*(1) and *aed*(1) commands) take a baseline *read* lock. This is a non-exclusive lock, so the concurrent development of changes is not hindered.

The command which modifies the baseline, *aeipass*(1), takes a baseline *write* lock. This is an exclusive lock, forcing *aeipass*(1) to block until there are no active baseline read locks.

It is possible that one of the above development commands will block until an in-progress *aegis −Integrate_PASS* completes. This is usually of short duration while the project history is updated. The

delay is essential so that these commands receive a consistent view of the baseline. No other integration command will cause the above development commands to block.

When aegis' branch functionality is in use, a read (non-exclusive) lock is taken on the branch baseline and also each of the "parent" baselines. However, a baseline write (exclusive) lock is only taken on the branch baseline; the "parent" baselines are only read (non-exclusive) locked.

File Action Adjustment

When this command runs, it first checks the change files against the projects files. If there are inconsistencies, the file actions will be adjusted as follows:

create If a file is being created, but another change set is integrated which also creates the file, the file action in the change set still being developed will be adjusted to "modify".

modify If a file is being modified, but another change set is integrated which removes the file, the file action in the change set still being developed will be adjusted to "create".

remove If a file is being removed, but another change set is integrated which removes the file, the file will be dropped from the change set still being developed.

CONFLICT RESOLUTION

If the version of a file in the change is not the same as the version of the file in the baseline, it is out-of-date; some other change has altered the file while this change was being developed.

When a difference is requested for an out-of-date file, a merge is performed between the common ancestor, the version in the baseline, and the version in the development directory. The command used to perform the merge is specified by the *merge_command* field of the project configuration file (see *aepconf*(5) for more information).

Please note that the *history_content_limitation* field of the project configuration file does **not** apply to the *merge_command* field.

After the merge is performed the version of the file will be changed to be the current version, marking the file as up to date, and a new build will be required.

The original file in your development directory is preserved with an "`,B`" suffix (B for backup). The source file contains the result of the merge. You should edit the source files, to make sure the automatic merge has produced sensible results.

This merge process works most of the time. Usually two changes to two logically separate areas of functionality will alter two logically separate parts of any files they may have in common. There are pathological cases where this merge process is spectacularly useless, but these are surprisingly rare in practice.

If you don't want the automatic merge results, simply use the *mv*(1) command to restore the contents from the "`,B`" file.

If any merges are required no differences will be performed. An error message and a non-zero exit status will also result. This is to ensure that developers notice that merges have been done, and that they reconcile the sources and the merged *,D* files before the next difference. See the **−No_Merge** and **−Only_Merge** options, below, for exact control of when merging is performed.

Cloning and Merging

When you use *aeclone*(1) to clone a change set, and then integrate one of the two change sets, you will observe that Aegis says that the files of the un-integrated change are now out-of-date.

If you run *aem*(1) to bring the out-of-date files back up-to-date, *fmerge*(1) and some (but not) all other merging tools, it signals just about everything as a conflict, even though both alternatives are identical.

The problem is that two changes making identical edits to the same place in the same file are a logical conflict, even if not an actual conflict, and it takes a human to figure out the difference. Think of a shopping list: the ensuite needs more soap, and so does the main bathroom. The second "soap" on the merge of the two shopping lists isn't a duplicate, you really do need two boxes of soap. Sometimes edits of source files are the same: sometimes the logical conflict is resolved by applying both identical edits, not

just one.

This is just the *fmerge*(1) command being more conservative than RCS's *merge*(1) command.

The easiest way to deal with this common situation it to run an

```
aecpu −unchanged
```

command *before* you run the *aem*(1) merge command, and you will have less grief. It's also worth remembering that Aegis stashes the original file with a `,B` suffix (B for backup) so you can simply

```
mv fubar,B fubar
```

if you know that all of the conflicts are logical conflicts.

INTEGRATION

During integration, it is also necessary to difference a change. This provides the difference between the branch and its parent, for when development on a branch is completed and it is to be reviewed. The baseline of a branch is the development directory of the composite change it represents.

OPTIONS

The following options are understood:

−ANticipate *change-number*
: This option is used to nominate a source for the reference files, rather than the baseline. This may be used to synchronize with a change without having to wait for it to arrive in the baseline. It is an error if the anticipated change is not in one of the *'being reviewed'* or *'awaiting integration'* or *'being integrated'* states. A merge is always performed, because the anticipated change is "about" to make any common file out-of-date. You will still have to perform a "real" merge later.

−BRanch *number*
: This option may be used to specify a different branch for the origin file, rather than the baseline. (See also **−TRunk** option. Please Note: the **−BRanch** option does not take a project name, just the branch number suffix.

−GrandParent
: This option may be used to specify the grandparent branch (one up from the current branch) for the origin file, rather than the baseline. (The −grandparent option is the same as the "−branch .." option.)

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

-List
: This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
: This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

-TRunk
: This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the "−branch −" option.)

−No_Merge
: This option is used to cause only file differences to be generated, even when file versions are out-of-date. If not set, the default is to use the *diff_preference* field of the *aeuconf*(5) file.

−Only_Merge
: This option is used to cause only file merges to be performed on files with out-of-date versions. Other source files are ignored. If not set, the default is to use the *diff_preference* field of the

aeuconf(5) file.

−Automatic_Merge
This option is used to perform *−Only_Merge* if any source files have out-of-date versions, otherwise *−No_Merge is performed.* Only merges or differences will be performed, it will never use a mixture. If not set, the default is to use the *diff_preference* field of the *aeuconf*(5) file.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aed 'aegis −diff \!* −v'
sh$ aed(){aegis −diff "$@" −v}
For user's convenience, particularly when they have selected the "no merge" preference, there is also a merge alias:
csh% alias aem 'aegis −diff −only_merge \!* −v'
sh$ aem(){aegis −diff −only_merge $* −v}

ERRORS

It is an error if the change is not in the *being developed* or *being integrated* states.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeb(1) build also takes a baseline read lock (non-exclusive)

aecp(1) copy file also takes a baseline read lock (non-exclusive)

aedb(1) begin development of a change

aeipass(1)
integrate pass takes a baseline write lock (exclusive)

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis develop begin − begin development of a change

SYNOPSIS

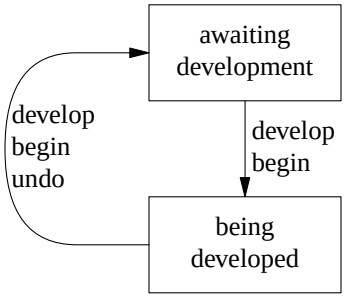
aegis −Develop_Begin *change-number* [*option*...]
aegis −Develop_Begin −List [*option*...]
aegis −Develop_Begin −Help

DESCRIPTION

The *aegis −Develop_Begin* command is used to commence development of a change.

The development directory for the change will be created automatically; below the directory specified in the default_development_directory field of *aeuconf*(5), or if not set below the directory specified in the default_development_directory field of *aepattr*(5), or if not set below the current user's home directory. It is rare to need to know the exact pathname of the development directory, as the *aecd*(1) command can take you there at any time.

Successful execution of this command will move the specified change from the *awaiting development* state to the *being developed* state.



Notification

The *develop_begin_command* in the project configuration file (see *aepconf*(5) for more information) will be run, if specified. This is run after the aegis locks are released, so additional aegis commands may be run from here, if used with care. The symbolic links (see below) have *not* yet been created.

Development Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

ADMINISTRATOR OVERRIDE

It is possible for project administrators to use the **−User** option to force a developer to start developing a change. Some sites prefer to work this way. Note that developers still have the ability to use the *aedbu*(1) command.

Warning: capricious use of this command will rapidly alienate developers. The defaulting rules, particularly for the change number, depend on aegis and the developer agreeing on what the developer is currently working on.

The *forced_develop_begin_notify_command* project attribute (see *aepattr*(5) for more information) will be run when an administrator uses the **−User** option, in an attempt to minimize the surprises for developers. A suitable command is

```
forced_develop_begin_notify_command =
    "$datadir/db_forced.sh $p $c $developer";
```

This command will send e-mail to the developer, informing her that the change has been assigned to her.

SYMBOLIC LINKS

Many dependency maintenance tools, and indeed some compilers, have little or no support for include file search paths, and thus for the concept of the two-level directory hierarchy employed by Aegis. (It becomes multi-level when Aegis' branching functionality is used.) To allow these tools to be used, Aegis provides the ability to maintain a set of symbolic links between the development directory of a change and the baseline of a project, so it appears to these tools that all of the project's files are present in the development directory.

Project Configuration

The *development_directory_style* field of the project configuration file controls the appearance of the development directory. See *aepconf*(5) for more information.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
    during_build_only = true;
};
```

the user never sees the symbolic links, because they are added purely for the benefit of the dependency maintenance tool during the execution of the *aeb*(1) command.

By using a setting such as

```
development_directory_style =
{
    source_file_symlink = true;
};
```

(the other will default to false) the symbolic links will be created at develop begin time (see *aedb*(1) for more information) and also maintained by each *aeb*(1) invocation. Note that the symbolic links are only maintained at these times, so project integrations during the course of editing change sourec files may leave the symbolic links in an inconsistent state until the next build.

When files are copied from the baseline into a change, using the *aecp*(1) command, the symbolic link pointing into the baseline, if any, will be removed before the file is copied.

Note: Using this functionality in either form has implications for how the rules file of the dependency maintenance tool is written. Rules must *remove* their targets before creating them (usually with an *rm −f* command) if you use any of the link sub-fields (both hard links and symbolic links). This is to avoid attempting to write the result on the symbolic link, which will point at a read-only file in the project baseline. This is similar to the same requirement for using the *link_integration_directory* field of the project configuration file.

User Configuration

There is a *symbolic_link_preference* field in the user configuration file (see *aeuconf*(5) for more information). This controls whether *aeb*(1) will verify the symbolic links before the build (default) or whether it will assume they are up-to-date. (This field is only relevant if *development_directory__style.-source_file_symlink* is true.)

For medium-to-large projects, verifying the symbolic links can take as long as the build itself. Assuming the symbolic links are up-to-date can be a large time-saving for these projects. It may be advisable to review your choice of DMT in such a situation.

The *aedb*(1) command **does not** consult this preference. Thus, in most situations, the symbolic links will be up-to-date when the build is performed. The only Aegis function which may result in the symbolic links

becoming out-of-date is the integration of another change, as this may alter the presence or absence of files in the baseline. In this situation, the default *aeb*(1) action is to ignore the user preference and the verify symbolic links.

There are two command line options which modify *aeb*(1) behavior further: the **−Verify-Symbolic-Links** option says to verify the symbolic links; and the **−Assume-Symbolic-Links** option says to assume the symbolic links are up-to-date. In each case the option over-rides the default and the user preference.

It is possible to obtain behaviour similar to Tom Lord'a Arch by using a setting such as:

```
development_directory_style =
{
    source_file_link = true;
    source_file_symlink = true;
};
```

It is possible to obtain behaviour similar to CVS by using a setting such as:

```
development_directory_style =
{
    source_file_copy = true;
};
```

There are many more possible configurations of the *development_directory_style*, usually with helpful build side-effects. See *aepconf*(1) and the *Depenedency Maintenance Tool* chapter of the User Guide for more information.

The symbolic link command line options and preferences apply equally to hard links and file copies (the names have historical origins).

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*

This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is

usually useful for shell scripts.

−User *name*
This option is used to specify the user who is to develop the change. This option may only be used by a project administrator.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aedb 'aegis −db \!* −v'
sh$ aedb(){aegis −db "$@" −v}

ERRORS

It is an error if the change does not exist.
It is an error if the change is not in the *awaiting development* state.
It is an error if the current user is not a developer of the specified project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeb(1) build a change

aecd(1) change directory

aecp(1) copy files into a change

aed(1) find differences between a change and the baseline

aedbu(1)
undo the effects of aedb

aede(1) complete development of a change

aemv(1) rename a file as part of a change

aenc(1) add a new change to a project

aend(1) add a new developer to a project

aenf(1) add new files to a change

aent(1) add a new test to a change

aepa(1) modify the attributes of a project

aerm(1) add files to be deleted to a change

aet(1) run tests

aepattr(5)
project attributes file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

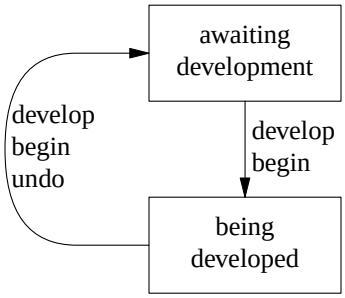
aegis develop begin undo − undo the effects of aedb

SYNOPSIS

aegis −Develop_Begin_Undo *change-number* [*option*...]
aegis −Develop_Begin_Undo −List [*option*...]
aegis −Develop_Begin_Undo −Help

DESCRIPTION

The *aegis −Develop_Begin_Undo* command is used to reverse the effects of the 'aegis −Develop_Begin' command. The development directory is discarded, even if the change has files associated with it, and even if the development directory is not empty; all files in the development directory will be lost. The change is returned to the *awaiting development* state.



Notification

The *develop_begin_undo_command* field of the project *config* file is run, if set. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-Interactive

Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep

This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep

This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−New-Change-Undo
By using this option it is possible to end developemnt of the change, and remove the change, all in the one command. The executing user must have both created the change and be the devloper of the change, or must be a project administrator. (Order is important here, this option mast appear on the command line *after* the **−Develop-Begin-Undo** option.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−User *name*
Ignored for backwards compatibility.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aedbu 'aegis −dbu \!* −v'
sh$ aedbu(){aegis −dbu "$@" −v}

ERRORS

It is an error if the change is no assigned to the current user.
It is an error if the change is not in the *being developed* state.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aedb(1) begin development of a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aede-policy − check change set is ready for aede

SYNOPSIS

aede-policy [*option*...][*policy*...]
aede-policy −Help
aede-policy −VERSion
aede-policy −List

DESCRIPTION

The *aede-policy* command is used to verify that a change set is ready to end development. This is intended to be used by the *develop_end_policy_command* field of the project configuration file.

```
develop_end_policy_command =
  "aede-policy −p $project −c $change all";
```

If any of the policies should fail, the *aede-policy* command will fail with an exit status of 1. This, in turn, will cause the *aede*(1) command to leave the change in the *being developed* state.

Note that the *aede*(1) command sets the appropriate environment variables, so the **−Project** and **−Change** options are rarely necessary.

If no policies appear on the command line, the *aede-policy* project specific attribute will be checked. If it exists, it contains a list of space separated policy names.

The *aede-policy*(1) command expects to be invoked on changes in the `being_developed` state. If invoked for a change in the `being_integrated` state (common if invoked as part of the build) it will silently do nothing. All other change states will result in a fatal error message.

POLICIES

There are a range of policies that can be selected.

all — Check all of the *copyright*, *crlf*, *description* and *printable* policies.

comments

This policy checks for C comments in C++ files, or C++ comments in C files. The forms of the comments give sublimial hints to the reader as to what language is being read. Mismatched comments make the code subtly harder to read and thus harder to maintain.

copyright

This policy checks that each file in the change set contains a copyright notice of the form

Copyright (C) **year** *something*

where **year** is the current year (you can have a range of years, too). Binary files are ignored. The *something* part is either the project specific *copyright-owner* attribute, or the executing users full name.

foreign-copyright

Change sets marked with a *foreign-copyright=true* attribute are ignored, as are files similarly marked.

crlf — This policy checks that all files are using UNIX line termination (NL), not DOS line termination (CRLF). Binary files are ignored.

description

This policy checks that the change set *brief_description* and *description* attributes have been updated to something other than the defaults.

escape-hyphen

This policy checks that hypen in roff sources (such as *man*(1) pages) that contain unescaped minus or hyphen characters. This is one of the more annoying warnings produced by *lintian*(1) when building Debian packages.

aede-policy-escape-hyphen

This check is not applied to files carrying a *aede-policy-escape-hyphen=false* attribute.

fsf-address
: This policy checks that the FSF address, if present in source files, is up-to-date. This is useful for Free Software projects.

gpl-version

gpl-version=*nn*
: This policy checks files that cite the GNU GPL in their file headers, to be sure they contain the correct version of the GNU GPL. Defaults to version 3 if no version number specified.

line-length

line-length=*nn*
: This policy checks that files have this maximum line length. Defaults to 80 if no width is specified. It understands *vim*(1) mode lines, particularly for the “tabsize” setting.

aede-policy-line-length
: Can be overridden per file using the *aede-policy-line-length* file attribute.

man-pages
: This policy requires that each installable program be accompanied by a *man*(1) manual page.

merge-fhist
: This policy requires that there be no *fmerge*(1) conflict lines present in any source files. The name comes from the name of the package containing this tool: *fhist*.

merge-rcs
: This policy requires that there be no *merge*(1) conflict lines present in any source files. The name comes from the name of the package containing this tool: *rcs*.

no-tabs
: This policy checks that files have no tabs characters in them. This is useful when a team of developers all use different editors and different tab stops. By only using spaces, the code is presented to all developers the same way.

  foreign-copyright
  : This check is not applied to change sets with a *foreign-copyright=true* attribute, because you have little control over them (change the tabs in a later change set, if at all).

  aede-policy-tabs-allowed
  : This check is not applied to files which are called `Makefile` or similar, and it is not applied to files carrying a *aede-policy-tabs-allowed=true* attribute.

printable
: This policy checks that each file in the change set contains only printable text characters and white space.

content-type
: The *content-type* file attribute is taken into account; if there is no *content-type* file attribute, or there is no charset specified by the *content-type* file attribute, plain 7-bit ASCII text is assumed.

reserved
: This policy checks that C and C++ identifiers reserved by the ANSI C and C++ Standards are used. See section 2.10 of both standards. Only C and C++ source files are checked.

text
: This policy checks that each file in the change set contains only text, although international character sets are acceptable. This is basically a test for NUL characters, because everything else could be part of a valid character encoding of some international character set.

version-info
: This policy checks the version-info rules for shared libraries, as laid out by the *libtool*(1) manual, and required by the Debian Policy Manual. This is done by examining the actual shared libraries, the one being built, and the one in the ancestor baseline (*i.e.* the one to be replaced) to confirm that the version-info strings conform. By examining the actual shared libraries, an objective view

of what has been added, modified and removed can be obtained.

The shared library to examine is obtained from a `project_specific` attribute:

aede-policy:version-info:library
: This is set to the baseline-relative name of the shared library file. You don't have to add the secret *libtool*(1) ".libs" directory, this policy can work that out for itself.

aemakegen:version-info
: This is the string which *aemakegen*(1) would use if it were invoked. This is also checked. If you aren't using *aemakegen*(1), it is a good idea to set this attribute anyway and access it via *aesub*(1) from within your build system.

This policy requires *nm*(1)'s **−−dynamic** option to work correctly on the `.so` file (it is part of the GNU binutils package).

vim-mode
: This policy checks that each file in the change set contains contains a *vim*(1) mode line. Binary files are ignored.

aede-policy-vim-mode-required
: Set this attribute to false on files for which this is not to be checked.

white-space
: This policy checks that there is no white space on the ends of lines, that there are no blank lines at the ends of files.

If no policy is specified, only the *description* policy will be checked.

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aede-policy* program.

−List List all of the available validations.

−Project *name*
: This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aede-policy* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aede-policy* command will exit with a status of 1 on any error. The *aede-policy* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aede(1) end development of a change

aepconf(5)
project configuration file

COPYRIGHT

aede-policy version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis develop end − complete development of a change

SYNOPSIS

aegis −Develop_End [*option*...]
aegis −Develop_End −List [*option*...]
aegis −Develop_End −Help

DESCRIPTION

The *aegis −Develop_End* command is used to notify aegis of the completion of the development of a change.

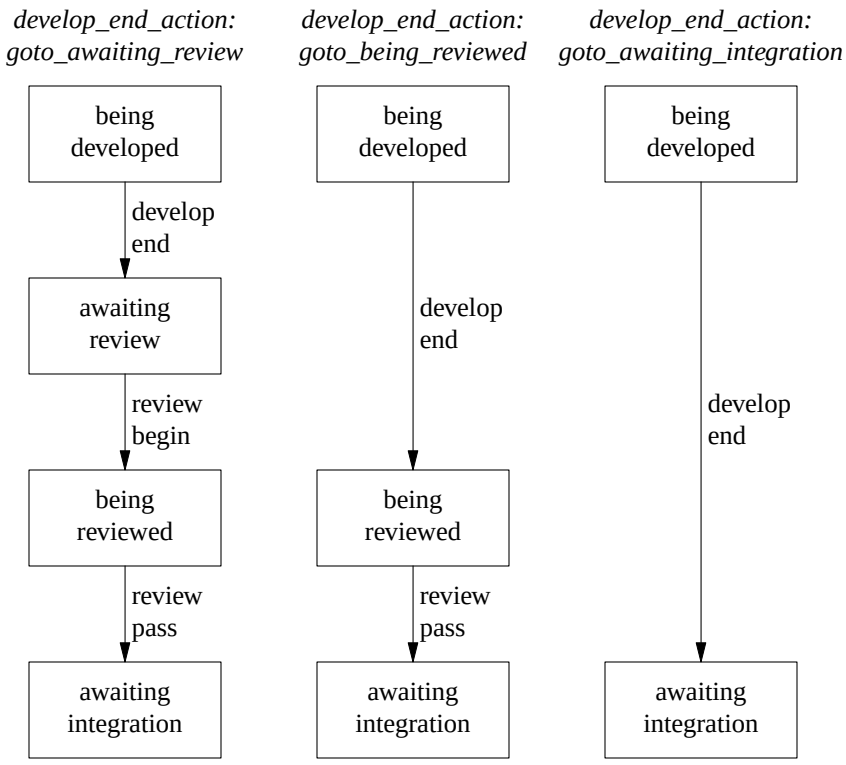
This command checks that you have successfully completed an ’aegis −Build’ command since any change source file was edited. See *aeb*(1) for more information.

This command checks that you have successfully completed an ’aegis −DIFFerence’ command since any change source file was edited. See *aed*(1) for more information.

This command checks that you have successfully completed an ’aegis −Test’ command since the last successful build, unless the change has a *test_exempt* attribute, or the build command is `"exit 0"`. This command checks that you have successfully completed an ’aegis −Test −BaseLine’ command, unless the change has a *test_baseline_exempt* attribute. This command checks that you have successfully completed an ’aegis −Test −REGression’ command, unless the change has a *test_regression_exempt* attribute. See *aet*(1) and *aecattr*(5) for more information.

If the change includes the project configuration file, this command checks project file names, to make sure they conform to the *maximum_filename_length* and *posix_filename_charset* field settings. See *aepconf*(5) for more information.

Successful execution of the command advances the change from the *being developed* state to the *being reviewed* state, by default. The *develop_end_action* project attribute controls which of the following 3 paths are taken.



(This is the default.)

Please Note: the third alternative, skipping reviews altogether, should only be used for single person projects. All self-respecting commercial enterprise will avoid this alternative.

Because branches may extend for many months or even years, it is common for the user who initiated the branch to be no longer with the project, or even the company. For this reason, project administrators may end the development of branches. For normal changes in this situation, use the *aechown*(1) command.

If the project configuration file has specified the presence of `Signed-off-by:` lines, a suitable line containing the current user's email address will be appended to the change description.

The change is no longer considered assigned to the developer.

Branches

If you get an error message telling you that you can't end a branch because a file needs to be merged, see the Branching chapter of the Aegis User Guide for more information.

While changes and branches are almost identical in the ways you manipulate them within Aegis, actual file changes must always be done in a change. Thus, it is necessary to create a new change on the branch and do a cross-branch grandparent merge before you will be able to develop-end a branch which is giving you this error.

Notification

On successful completion of the command, the *develop_end_notify_command* field of the project attributes file is run, if set. See *aepa*(1) and *aepattr*(5) for more information.

If your project has configured the *develop_end_action* in the project confifuration file to *goto_awaiting_-integration* then the *review_pass_notify_command* in the project attributes file is run instead, if set.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

−Signed_Off_By
This option may be used to have a `Signed-off-by:` line appended to the change set description.

−No_Signed_Off_By
This option may be used to prevent a `Signed-off-by:` line from being appended to the change set description.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aede 'aegis −de \!* −v'
sh$     aede(){aegis −de "$@" −v}
```

ERRORS

It is an error if the change is not assigned to the current user.

It is an error if The change is not in the *being developed* state.

It is an error if there has been no successful *'aegis −Build'* command since a change file was last edited.

It is an error if there has been no successful *'aegis −DIFFerence'* command since a change file was last edited.

It is an error if there has been no successful *'aegis −Test'* command since a change file was last edited.

It is an error if there has been no successful *'aegis −Test −BaseLine'* command since a change file was last edited.

It is an error if an read-only file is still copied into the change. Read-only files are to insulate a change from the baseline during development; they must be removed before development may end.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeb(1) build a change

aeca(1) list or modify attributes of a change

aed(1) difference a change

aedb(1) begin development of a change

aede-policy(1)
validate change set is ready to end

aedeu(1)
recall a change for further development

aerfail(1)
fail a change review

aerpass(1)
pass a change review

aet(1) test a change

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis develop end undo − recall a change for further development

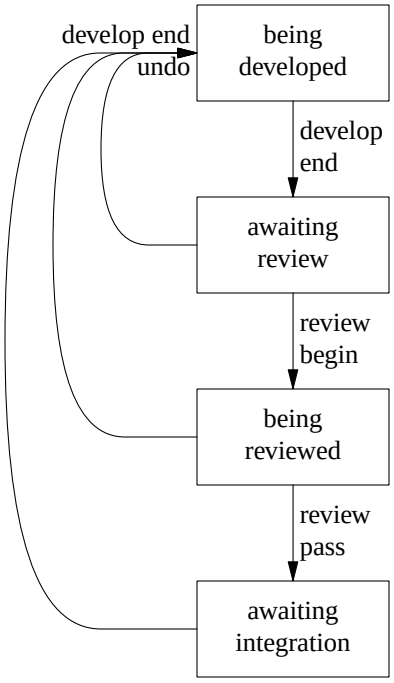
SYNOPSIS

aegis −Develop_End_Undo *change-number* [*option*...]
aegis −Develop_End_Undo −List [*option*...]
aegis −Develop_End_Undo −Help

DESCRIPTION

The *aegis −Develop_End_Undo* command is used to recall a change for further development.

Successful execution of this command returns the change to the *being developed* state.



The files are changed back to being owned by the current user, and cease to be read-only.

Notification

On successful completion of the command, the *develop_end_undo_notify_command* field of the project attributes file is run, if set. See *aepa*(1) and *aepattr*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more

information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aedeu 'aegis −deu \!* −v'
sh$ aedeu(){aegis −deu "$@" −v}

ERRORS

It is an error if the change is not in one of the *awaiting review* or *being reviewed* or *awaiting integration* states.

If is an error if the project has been configured to use the *awaiting review* state, and the change is currently in the *being reviewed* state. This is because the change currently belongs to the reviewer.

It is an error if the change was not developed by the current user.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

*aede*1 complete development of a change

*aerpass*1
pass review of a change

*aerfail*1 fail review of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aediff − file differences between deltas

SYNOPSIS

aediff [*option*...] *filename*
aediff −Help
aediff −VERSion

DESCRIPTION

The *aediff* command is used to obtain the difference between versions of the given *filename* across different file versions, a specified by the command lien options.

If two changes or deltas are specified, the difference between the versions of the file in each will be output.

If only one change or delta is specified, the second version defaults to the current change.

If no changes or deltas are specified, the first version defaults to the baseline and the second version defaults to the current change.

Examples

To see the difference in the project configuration file, *aegis.conf*, between deltas 1.2.D003 and 4.5.D067 the following command may be used:

```
aediff aegis.conf −c 1.2.D003 −c 4.5.D067
```

To see the differences in the project configuration file, between the head of the 7.6 branch and the current change, the following command may be used:

```
aediff −branch 7.6 −bl  aegis.conf
```

Many, many other combinatiosn are possible.

Using Graphical Tools

It is possible to use a graphical diff tool with the *aediff*(1) command. This is done by using the −command *option, or setting the* `AE2DIFF` *environment variable. For example, to use the tkdiff(1) command to display the differences you would use a command such as:*

```
aediff −command=tkdiff filename
```

If you use this option, many of the diff(1)-specific options will be ignored.

OPTIONS

The following options are understood:

-BaseLine

This option may be used to specify that the project baseline is the subject of the command.

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−COMmand *string*

This option may be used to set the command used to display differences. Using this option will cause *diff*(1)-specific options to be ignored. If not set, defaults to the value of the `AE2DIFF` environment variable, or "diff" otherwise.

−CONtext [*lines*]

Use the context output format, showing *lines* (an integer) lines of context, or three if *lines* is not given. For proper operation, *patch*(1) typically needs at least two lines of context.

−DELta *number*

This option may be used to specify a particular delta in the project’s history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning “when the branch started”.

−DELta_Date *string*
This option may be used to specify a particular date and time in the project's history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−Ignore_Blank_Lines
Ignore changes that just insert or delete blank lines.

−Ignore_All_Space
Ignore white space when comparing lines.

−Ignore_Case
Ignore changes in case; consider upper-case and lower-case to be the same.

−Ignore_Space_Change
Ignore changes in amount of white space.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Show_C_Function
Show which C function each change is in.

-TRunk
This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the "−branch −" option.)

−unified [*lines*]
Use the unified output format, showing *lines* (an integer) lines of context, or three if lines is not given. For proper operation, *patch*(1) typically needs at least two lines of context.

-Help
This option may be used to obtain more information about how to use the *aediff* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aediff* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aediff* command will exit with a status of 1 on any error. The *aediff* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aedist − remotely distribute a change

SYNOPSIS

aedist −Send [*option*...]
aedist −Receive [*option*...]
aedist −REPlay [*option*...] −f *URL*
aedist −MIssing [*option*...] −f *URL*
aedist −PENding [*option*...] −f *URL*
aedist −Inventory [*option*...]
aedist −ARChive [*option*...]
aedist −List [*option*...]
aedist −Help
aedist −VERSion

DESCRIPTION

The *aedist* command is used to send and receive change sets to facilitate geographically distributed development. The expected transport mechanism is e-mail, however other mechanisms are equally possible.

The basic function is to reproduce a change, so a command like

```
aedist −send | aedist −receive
```

may be used to clone a change, though less efficiently than *aeclone*(1). The file format used is designed to withstand mail servers, so activities such as

```
aedist −send | e-mail | aedist −receive
```

(where *e-mail* represents sending, transporting and receiving your e-mail) will reproduce the change on a remote system. With suitable tools (such as PGP) is it possible to

```
aedist −send | encrypt | e-mail | decrypt | aedist −receive
```

The mechanism is also designed to allow web-based distribution such as

```
aedist −send | web-server → web-browser | aedist −receive
```

by the use of appropriate CGI scripts and mailcap entries.

It is possible to support both a “push” model and a “pull” model using this command. For suggestions and ideas for various ways to do this, see the Aegis Users Guide.

SEND

The send variant takes a specified change, or baseline, and constructs a distribution package containing all of the change attributes and source file attributes and source file contents. The result is compressed, and encoded into a text format which can be sent as e-mail without being corrupted by the mail transfer agents along the way.

Options

The following options are understood by the send variant:

−BaseLine
This option may be used to specify the source of a project, rather than a change. Implies the *−Entire_Source* option, unless over-ridden.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−COMPATibility *version-number*
This option may be used to specify the version of *aedist*(1) which will be *receiving* this change set. This information is used to select which features to include in the data, and which to omit. By default, the latest feature set will be used.

−compression-algorithm *name*
This option may be used to specify the compression to be used. They are listed on order of compression effeciency.

none Use no compression (not always meaningful for all commands).

gzip Use the compression used by the *gzip*(1) program.

bzip2 Use the compression used by the *bzip2*(1) program.

More compression algorithms may be added in the future.

−COMPress
This option is deprecated in favour of the **−comp-alg=gzip** or **−comp-alg=bzip2** options.

−No_COMPress
This options is deprecated in favour of the **−comp-alg=none** option.

−Content_Transfer_Encoding *name*
This option may be used to specify the content transfer encoding to be used. It may take one of the following values:

None No content transfer encoding is to be performed.

Base64 The MIME base 64 encoding is to be used. This is the default.

Quoted_Printable
The MIME quoted printable encoding is to be used.

Unix_to_Unix_encode
The ancient unix-to-unix encoding is to be used.

These encodings may be abbreviated in the same way as comment line options.

−Ascii_Armor
This means the same as the "−cte=base64" option above.

−No_Ascii_Armor
This means the same as the "−cte=none" option above.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning "when the branch started".

−DELta_Date *string*
This option may be used to specify a particular date and time in the project's history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−Description_Header
This option may be used to add an RFC 822 style header to the change description being sent, with a From and Date line. This is the default.

−No_Description_Header
This option suppresses the description header.

−Entire_Source
This option may be used to send the entire source of the project, as well as the change source files.

−Ignore_UUID
This option may be used to ignore the UUID, if present, of the outgoing change set.

−No_Ignore_UUID
This option forces the *aedist* command to use the outgoing change set's UUID information. This is the default (unless the compatibility option will to avoid attributes).

−Mime_Headers
This option may be use to force the presence of mime headers in the output, in circumstances they would usually be absent.

−No_Mime_Headers
This option may be use to force the absence of mime headers in the output, in circumstances where they would usually be present.

−Partial_Source
This option may be used to send only source files of a change. This is the default, except for the *−BaseLine* option.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−PATch This option is deprecated. Please use the **−COMPATibility** option instead.

−No_PATch
This option is deprecated. Please use the **−COMPATibility=4.6** option instead.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Signed_Off_By
This option may be used to have a `Signed-off-by:` line appended to the change set description.

−No_Signed_Off_By
This option may be used to prevent a `Signed-off-by:` line from being appended to the change set description.

RECEIVE

The receive variant takes a change package created by the send variant and creates an Aegis change (see *aenc*(1)) to implement the change within. Files are added to the change (see *aerm*(1), *aecp*(1), *aenf*(1) and *aent*(1)) and then the file contents are unpackaged into the development directory.

The change is then built (see *aeb*(1)), differenced (see *aed*(1)), and tested (see *aet*(1)). If all of this is successful, development of the change is ended (see *aed*(1)). The automatic process stops at this point, so that a local reviewer can confirm that the change is desired.

Notification

The *aedist* command invokes various other Aegis commands. The usual notifications that these commands would issue are issued.

Options

The following options are understood by the receive variant:

−Change *number*
This option may be used to choose the change number to be used, otherwise one will be chosen automatically.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, just as for the *aecp*(1) command. You may also use a delta name instead of a delta number.

−DIRectory *path*
This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename ‘−’ is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−PATch This option may be used to apply patches from the input, if available. This generally results in fewer merge problems, but it requires the two repositories to be well synchronized. This is the default.

−No_PATch
This option may be used to ignore patches in the input, if any are present.

−Ignore_UUID
This option may be used to ignore the UUID, if present, of the incoming change set.

−No_Ignore_UUID
This option force the *aedist* command to use the change set’s UUID. This is the default.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined change number. Useful for writing scripts.

−Project *name*
This option may be used to set the project name. If not specified, the project name in the input package will be used, rather than the usual project name defaulting mechanism.

−Trojan This option may be used to treat the change set as if it had a Trojan horse attack in it.

−No_Trojan
This option may be used to treat the change set as if it definitely does not have a Trojan horse attack in it. *Use with extreme care.* You need to have authenticated the message with something like PGP first **and** know the the author well.

Security
Receiving changes by e-mail, and automatically committing them to the baseline without checking them, would be a recipe for disaster. A number of safeguards are provided:

- The format of the package is confirmed to be correct, and the package verified for internal consistency, before it is unpacked and acted upon.
- The automatic portion of the process stops when development ends. This ensures that a local reviewer validates the change before it is committed, preventing accidental or malicious damage.
- If the change seeks to update the project *config* file, the automatic process terminates before the build or difference occurs. This is because this file could contain trojans for these operations, so a human must examine the file before the change proceeds any further.
- There is a *potential_trojan_horse = [string];* field in the project*config* file. Nominate build configuration files, shell scripts, code generators, *etc* here to specify files in addition to the project configuration file which should cause the automatic processing to halt.
- The use of e-mail authentication and encryption systems, such as PGP and GPG, are encouraged. However, it is expected that this processing will occur after *aedist −send* has constructed the package and

before *aedist −receive* examines and acts on the package. Verification of the sender is the surest defense against trojan horses.

- Automatic sending and receiving of packages is supported, but not implemented within the aedist command. It is expected that the aedist command will be used within shell scripts customized for your site and its unique security requirements. See the Aegis User Guide for several different ways to do this.
- The more you use Aegis' test management facilities (see *aent*(1) and *aet*(1)) the harder it is for an inadequate change to get into the baseline.

Duplicate Storms

In a distributed development environment, it is common for change sets to eventually be propagated back to the originator. There are situations (particularly in some star topologies) where several copies of the package will return to the originator.

If these change sets are not detected at the review stage, and are propagated out yet again, there is the possibility of an exponential explosion of redundant packages being distributed again and again.

To combat this, changes are checked after the files are unpacked, but before and build or difference or test is performed. The "*aecpu −unchanged*" command is used to exclude all files that the local repository already has in the desired form. If no change files remain after this, the change is dropped entirely (see *aedbu*(1) and *aencu*(1)).

REPLAY

If you are tracking a remote site which makes a project available via the *aeget*(1) web interface, you can automatically synchronize with the remote site using the *aedist −replay* command.

For example, Aegis developers can track the master project with a command of the form:

```
aedist −p aegis.4.25 −replay −f aegis.sourceforge.net
```

This command is internally rewritten as

```
aedist −replay −p aegis.4.25 −f \
  http://aegis.sf.net/cgi-bin/aeget/aegis.4.25/?inventory
```

If your `cgi-bin` directory is somewhere else, you will need to use the long form.

The change set inventory page is human readable if you want to see what it contains. The links on this page provide all the information necessary to download any of the change sets listed.

This command reads the list of change set UUIDs from the remote repository, and compares it with the list of change set UUIDs in the local repository, and fetches any that are not present locally.

Each of the change sets required are downloaded and unpacked by issuing a command such as

```
aedist −rec −f \
  http://aegis.sf.net/cgi-bin/aeget/aegis.4.19.C010/?aedist
```

If this completes successfully (and it is possible it won't, either because of trojan warnings, or some conflict between local changes and the incoming remote changes), and your project has its *develop_end_action* set to *goto_awaiting_integration*, the change will be integrated using a command such as:

```
aeintegratq −p aegis.4.25 −c 10
```

and then starts over again for the next missing change set.

This command will attempt to use the same change number as in the remote repository, if it is available.

Options

The following options are understood by this variant:

−EXclude_UUID *UUID*

This option may be used to exclude some change sets from being downloaded and unpacked. This option may be used more than once.

−No_EXclude_UUID *UUID*

This option may be used to explicitly list change sets to be downloaded and unpacked, to the exclusion of all others. This option may be used more than once.

−EXclude_VERsion *pattern*
This option may be used to explicitly exclude some change set from being downloaded and unpacked. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−INclude_VERsion *pattern*
This option may be used to explicitly list change sets to be downloaded and unpacked, to the exclusion of all others. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename '−' is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−MAXimum
This option may be used to download as many changes as possible by excluding the maximum number of local changes sets, by excluding both local change sets UUIDs (the default) but also excluding UUIDs mentioned in change "original-uuid" attributes.

−PErsevere
This option may be used to specify that all relevant change sets should be downloaded, even if some fail. Defaults to the user's *persevere_preference* if not specified, see *aeuconf*(5) for more information.

−No_PErsevere
This option may be used to specify that the downloading of change sets should stop after the first failure. Defaults to the user's *persevere_preference* if not specified, see *aeuconf*(5) for more information.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Trojan This option is passed to any *aedist*(1) commands spawned by this command.

−No_Trojan
This option is passed to any *aedist*(1) commands spawned by this command.

−Not_Compatibility
This option must be used when using *aedist −replay* against a file based inventory.

MISSING

If you want to see the change sets that *aedist −replay* may download before it goes ahead and does it, you can use a command such as:

```
aedist −missing −f aegis.sf.net
```

In particular, this allows you to select appropriate UUIDs for the *aedist −replay −exclude* or *−no-exclude* options.

Options

The following options are understood by this variant:

−EXclude_UUID *UUID*
This option may be used to exclude some change sets from being listed. This option may be used more than once.

−No_EXclude_UUID *UUID*
This option may be used to explicitly list change sets to be listed, to the exclusion of all others. This option may be used more than once.

−EXclude_VERsion *pattern*
This option may be used to explicitly exclude some change set from being listed. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−INclude_Version *pattern*
This option may be used to explicitly list change sets to be listed, to the exclusion of all others. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−MAXimum
This option may be used to download as many changes as possible by excluding the maximum number of local changes sets, by excluding both local change sets UUIDs (the default) but also excluding UUIDs mentioned in change "original-uuid" attributes.

PENDING

If you want to see the change sets that a remote repository is missing with respect to yours, you can use a command such as:

```
aedist −pending −f aegis.sf.net
```

Options

The following options are understood by this variant: **−EXclude_UUID** *UUID* This option may be used to exclude some local change sets from being listed. This option may be used more than once.

−No_EXclude_UUID *UUID*
This option may be used to explicitly list local change sets to be listed, to the exclusion of all others. This option may be used more than once.

−EXclude_VERsion *pattern*
This option may be used to explicitly exclude some local change set from being listed. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−INclude_VERsion *pattern*
This option may be used to explicitly list local change sets to be listed, to the exclusion of all others. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

INVENTORY

The inventory variant can be used as an alternative to aeget to generate the inventory used by the replay, missing and pending variants. The idea is to run the inventory variant on the development machine and then upload its output to the public repository. In order to generate the inventory you can use a command such as:

```
aedist −inventory −proj project > inventory.html
```

Options

The following options are understood by this variant:

−AEGET
This option is used by aeget to require the original *aeget*(1) behavior.

−All This option is used to require the inclusion of the UUIDs contained in the original-UUID attribute of each change.

−EXclude_Version *pattern*
This option may be used to explicitly exclude some change set to be added to the inventory file. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−INclude_Version *pattern*
This option may be used to explicitly list change sets to be added to the the inventory file, to the exclusion of all others. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−path_prefix_add
This option is used to add a path prefix to the URLs generated in the inventory.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

ARCHIVE

As an alternative to using the *aeget*(1) CGI program, the aedist archive variant is provided. This variant can be used to populate a directory with the aedist archives of each change with an UUID. The archives will have a name based on the UUID of the change with extension ".ae", the fingerprint of the archive will be stored in a file with the same (base)name with extension ".fp". Running the archive variant multiple times against the same target directory will update that directory, adding the files of changes integrated after the last run and regenerating the files if a corruption is detected.

Options

The following options are understood by the archive variant:

−Change-Directory *directory*
This option is used to designate the directory to be populated with the *aedist*(1) generated files. If this option is not used then the current directory is used as the target of the command. The directory must exists and be accessible by the user running the command.

−EXclude_Version *pattern*
This option may be used to explicitly exclude some change set to be added to the target directory. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−INclude_Version *pattern*
This option may be used to explicitly list change sets to be added to the target directory, to the exclusion of all others. The *pattern* is matched against the version as displayed in the inventory. This option may be used more than once.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

LIST

The list variant can be used to list the contents of a package without actually unpacking it first. The output is reminiscent of the *aegis −list change-details* output.

Options

The following options are understood by the list variant:

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename '−' is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide

a progress bar.)

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default. Only useful with the −List option.

OPTIONS

The following options to this command haven't been mentioned yet:

-Help
This option may be used to obtain more information about how to use the *aedist* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aedist* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

FILE FORMAT

The file format re-uses existing formats, rather than introduce anything new. This means it is possible to extract the contents of a package even when aedist is unavailable.

- The source files and other information is stored as a *cpio*(1) archive.
- The archive is compressed using the *bzip2*(1) format. Typically primary source files are ASCII text, resulting in significant compression.
- The compressed result is encoded using the MIME base64 encoding. This makes the result approximately 33% larger than the compressed binary would be, but still smaller than the primary sources.

The *cpio* archive is used to store

`etc/project-name`
This contains the project name to apply the package to, unless over-ridden by the −project command line option.

`etc/change-number`
This contains the change number of the original change, this may be preserved if available on the target repository unless over-ridden by the −change command line option.

`etc/change-set`
This contains the change attributes and the list of source files and usages, in *aecstate*(5) format.

`patch/`*filename*
Each modified or renamed file in the package (named in *etc/change-set*) appears under the *patch* directory. The file may be empty unless some edits was done on the source repository.

`src/`*filename*
Each source file in the package (named in *etc/change-set*) appears under the *src/* directory.

Extra files, or files out of order, are a fatal error.

EXIT STATUS

The *aedist* command will exit with a status of 1 on any error. The *aedist* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

CREDITS

This program evolved through discussion with a number of people. If I have forgotten anyone, it wasn't intentional.

Ralf Fassel	<ralf@akutech.de>	Catching trojan horses.
Walter Franzini	<walter.franzini@sys-net.it>	coding −replay download
Florian Xhumari	<Florian.Xhumari@inria.fr>	On the need for pull interfaces.
Graham Wheeler	<gram@cdsec.com>	HTTP pull interfacing.

NAME

aegis delta name − assign a symbolic name to a project delta

SYNOPSIS

aegis −DELta_NAme [*option*...] *name*
aegis −Help
aegis −VERSion

DESCRIPTION

The *aegis −DELta_NAme* command is used to add a symbolic name to a project delta. This is so that this name may be used, rather than the number, when extracting previous versions of the file using the *aecp*(1) command.

The **−DELta** *number* option on the command line specifies a delta number of the project. That is, it is the delta number assigned to an integration. Delta names may only be applied to project baselines. If no delta number is given on the command line, the current baseline is the default.

A name must be given on the command line. This is the name which will be assigned to the delta. If the name has already been used, you will be given a fatal error message. If you also specify the **−OverWriting** option the name will be removed from its previous delta and assigned to the requested delta.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−DELta *number*
This option may be used to specify a particular delta in the project's history to name.

−DELta_DAte *string*
This option may be used to specify a particular date and time in the project's history.

−Delta_From_Change *number*
This option may be used to specify a particular delta in the project's history, based on when the given change was successfully integrated.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-OverWriting
This option may be used to force overwriting of files. The default action is to give an error if an existing file would be overwritten.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy a file into a change, particularly the **−DELta** option

aeib(1) start the integration of a change

ael(1) list interesting this, particularly the *project_history* listing

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeedit − edit a change's files

SYNOPSIS

aeedit [**−p** *project-name*] [**−c** *change-number*]

DESCRIPTION

The *aeedit* command is used to edit all of the files in a change. For editors with one buffer per file, this can be very useful except for changes with huge numbers of files.

The editor's current directory is changed to the top of the change's development directory tree.

If you have PlasticFS installed, the editor's environment will be configured to present the development directory as the complete search path.

OPTIONS

The following options are understood:

−p *project-name*
This option may be used to set the project name.

−c *change-number*
This option may be used to set the change number.

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

COPYRIGHT

aeedit version 4.25.D510

NAME

aegis file attributes − modify the attributes of a file

SYNOPSIS

aegis −File_ATtributes −File *attr-file* [*option*...] *filename*
aegis −File_ATtributes −Edit [*option*...] *filename*
aegis −File_ATtributes [*option*...] *name=value filename*
aegis −File_ATtributes −UUID *number* **−File** *filename*
aegis −File_ATtributes −Help

DESCRIPTION

The *aegis −File_ATtributes* command is used to set, edit or list the attributes of a file.

The output of the **−List** variant is suitable for use as input at a later time.

See *aefattr*(5) for a description of the file format.

Attribute names are not case sensitive. File attributes with a name starting with an upper case letter will appear in *ael*(1) and *aeget*(1) listings, while those starting with a lower case letter will not.

Shorthand

If you are only setting the values of unique attributes, it is possible to do this from the command line, using the *name=value* form.

Note that this usage will replace the first attribute with the given name. If there is more than one attribute of that name, the second and subsequent attributes are unchanged. If there is no attribute of the given name, it will be appended.

You may set more than one attribute at a time, provided that their names are unique. Attribute names are not case sensitive.

Known Attribute Names

While many of the anticipated uses of file attributes are to allow projects to attach their own specialized data to individual files, Aegis also uses some attributes for its own purposes (and arguably, should always have done so to maximize forwards compatibility across Aegis upgrades).

aede-policy-crlf-allowed
: boolean. If true, the *crlf* policy of the *aede-policy*(1) command does not apply.

aede-policy-escape-hyphen
: boolean. If false, the *escape-hyphen* policy of the *aede-policy*(1) command does not apply.

aede-policy-line-length
: integer. The maximum allowed line length in the *line-length* policy of the *aede-policy*(1) command; infinity if 0.

aede-policy-tabs-allowed
: boolean. If true, the *no-tabs* policy of the *aede-policy*(1) command does not apply.

aeipass-options:assign-file-uuid
: boolean. If false, *aeipass* will not assign a fresh UUID to this file. This flag is set by *aedist* −rec if the action associated with the file is a create and the file is missing the UUID. This behaviour is needed to prevent the effect of having different UUIDs assigned to the same file in different repositories.

aemakegen:noinst
: boolean. If true, *aemakegen*(1) will not cause the program to be installed. Usually attached to the source file containing the *main* function, or to script files. Defaults to false if not defined (*i.e.* do install program).

content-type
: This is taken directly from the MIME definition of Content-Type. It remembers what sort of file this is. It is anticipated that a *diff* tool, for example, could make use of this attribute to provide format-specific file difference listings. Some change set interchange formats are capable of

carrying this information.

entire-source-hide
: boolean. If true, this file is not included by the *aedist −entire-source* flag. The *aetar* and *aerevml* commands work similarly. Think of it as a "local only" flag.

foreign-copyright
: boolean. If true, this file will not be checked by the *aede-policy*(1) `copyright` validation.

local-source-hide
: boolean. If true, this file is not included by *aedist* change sets. The *aetar* and *aerevml* commands work similarly. Change sets which contain *only* thee files will be omitted from the *aedist −inventory* output. Think of it as a "local only" flag.

test/*arch*/elapsed
: This is used to estimate test duration. See *aet*(1) for more information.

OPTIONS

The following options are understood:

−BAse_RElative
: This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
: This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

-List
: This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
: This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
: This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts. **−Universal_Unique_IDentifier** *number* This option may be used to set the UUID of a file.

-Verbose
: This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
: This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see

aeuconf(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aefa 'aegis −fat \!* −v'
sh$     aefa(){aegis −fat "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the specified project.
It is an error if the current user is not the developer of the specified change.
It is an error if the file is not included in the specified change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeca(5) modify the attributes of a change

aefattr(5)
file attributes file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aefind − search for files in directory hierarchy

SYNOPSIS

aefind [*option*...] *path*... *expression*
aefind −Help
aefind −VERSion

DESCRIPTION

The *aefind* command is used to search the combined directory tree of a change and its project. It is intentionally similar to *find (1)*, however it unifies the directory stack of a change and its branch baseline, and the branch's ancestors' baselines if any.

For each file found in the directory tree, the given expression is evaluated from left to right, according to the rules of precedence (see the section on OPERATORS, below), only until the outcome is known, at which point aefind moves on to the next file name.

If no directory is named on the command line, the current directory is assumed.

Files which have been removed from the project, even if they somehow remain in the directory tree, will not be reported.

OPTIONS

The following options are understood:

-BaseLine
This option may be used to specify that the project baseline is the subject of the command.

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aefind* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Resolve
This option may be used to request that filenames be absolute paths, referring to the fully resolved file name. This is the default.

−No_Resolve
This option may be used to request that filenames be base relative names, relative to the root of the "stacked" directory tree.

−Verbose
This option may be used to request that the expression be printed again on the standard output. This is the expression as understood by *aefind*, to assist you in ensuring that you and the command agree. The expression is fully parenthesized, and all implicit operators made explicit. Where possible, constant expressions will have been folded.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aefind* are long, this means ignoring the extra leading '−'. The "**−−***option***=***value*" convention is also understood.

EXPRESSIONS

The expression is made up of basic elements, tests (which return a true or false value), and actions (which have side effects and return a true or false value), all separated by operators.

BASIC ELEMENTS

{} The value of this expression is the file name of the file currently being considered. The value is affected the the **−Resolve** option.

{−} The value of this expression is the file name of the file currently being considered, relative to the base of the directory stack.

{+} The value of this expression is the absolute path of the file currently being considered.

number Numbers may be specified directly, for use with other tests and operators. In the style of C, they may be hexadecimal with a "0x" prefix, octal with a "0" prefix, or decimal otherwise.

string Strings may be specified directly, for use with other tests and operators. If the string contains shell meta-characters, you may need to quote it.

−FAlse The value of this expression is always false.

−NOW The value of this expression is the current time, at the start of execution.

−TRue The value of this expression is always true.

OPERATORS

The **−and** operator is assumed where the operator is omitted. You will need to quote many of the operators, to protect them from interpretation by the shell. Each operator must be a separate command line argument.

(*expr* **)** Force precedence.

+ *expr* Unary plus. Is is an error if the argument cannot be coerced to a number.

− *expr* Unary minus. Result is the numeric negative of the argument. Is is an error if the argument cannot be coerced to a number.

! *expr* Logical negation of the sense of the expression. Is is an error if the argument cannot be coerced to a boolean.
Synonym: **−Not**

~ *expr* Bitwise not of the argument. Is is an error if the argument cannot be coerced to an integer.

expr1 ***** *expr2*
This operation multiplies the two values. Is is an error if the arguments cannot be coerced to numbers.

expr1 **/** *expr2*
This operation divides the argument value by the second. Is is an error if the arguments cannot be coerced to numbers. Is is an error if the second argument is zero.

expr1 **%** *expr2*
This operation produces the remainder of the division of the first argument by the argument. Is is an error if the arguments cannot be coerced to numbers. Is is an error if the second argument is

zero.

expr1 **~** *expr2*
: Is is an error if the arguments cannot be coerced to strings. The first argument is the pattern, and the second is the string. Is is an error if the first argument is not a valid pattern. The result is true if the pattern matches, and false if it does not. This operation performs a shell file pattern comparison.

expr1 **+** *expr2*
: This operation adds the two values. Is is an error if the values cannot be coerced to numbers.

expr1 **−** *expr2*
: This operation subtracts the second values from the first. Is is an error if the values cannot be coerced to numbers.

expr1 **##** *expr2*
: This operation concatenates the arguments. Is is an error if the arguments cannot be coerced to strings. (Note: this is *not* the same as the **:** operator of the *expr*(1) command.)

expr1 **<<** *expr2*
: Shift the first argument left by the number of bits specified by the second argument. The left argument is treated as an *unsigned* number. Is is an error if the values cannot be coerced to numbers.

expr1 **>>** *expr2*
: Shift the first argument right by the number of bits specified by the second argument. The left argument is treated as an *unsigned* number. Is is an error if the values cannot be coerced to numbers.

expr1 **<** *expr2*
: Compare the values and produce true if the first value is less than the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **<=** *expr2*
: Compare the values and produce true if the first value is less than or equal to the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **>** *expr2*
: Compare the values and produce true if the first value is greater than the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **>=** *expr2*
: Compare the values and produce true if the first value is greater than or equal to the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **==** *expr2*
: Compare the values and produce true if the first value is equal to the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **!=** *expr2*
: Compare the values and produce true if the first value is not equal to the second value, false otherwise. If both values can be coerced to numbers, the comparison is numeric; if both values can be coerced to strings, the comparison is lexicographic; otherwise is it an error.

expr1 **&** *expr2*
: This operation produces the bitwise-and of the two values. Is is an error if the values cannot be coerced to numbers.

expr1 **|** *expr2*
This operation produces the bitwise-or of the two values. Is is an error if the values cannot be coerced to numbers.

expr1 **&&** *expr2*
Result is true if both expressions are true. Short circuit evaluation is used, and so *expr2* is not evaluated if *expr1* is false. Is is an error if the arguments cannot be coerced to booleans.
Synonym: **−And**

expr1 expr2
Logical and (implied). Result is true if both expressions are true. Short circuit evaluation is used, and so *expr2* is not evaluated if *expr1* is false. Please note that implicit operator plays merry hell with operator precedence, because there is no operator. If you are getting odd results, use explicit operators.

expr1 **||** *expr2*
Result is true if either expression is true. Short circuit evaluation is used, and so *expr2* is not evaluated if *expr1* is true. Is is an error if the arguments cannot be coerced to booleans.
Synonym: **−Or**

expr1 **?** *expr2* **:** *expr3*
The value of this expression is *expr2* if *expr1* is true, and *expr3* otherwise. The *expr1* is always evaluated, but only one of *expr2* or *expr3* will be evaluated. It is an error if the value of *expr1* cannot be coerced to boolean.

expr1 **,** *expr2*
Both *expr1* and *expr2* are always evaluated. The value of *expr1* is discarded; the value of the expression is the value of *expr2*.

Operators have precedence as described by the following table, highest to lowest:

Operator	Direction
(unary) + − ~ !	←
* / % ~	→
+ − ##	→
<< >>	→
< <= > >=	→
== !=	→
&	→
^	→
|	→
&& *(implied)* -And	→
|| -Or	→
? :	←
,	→

FUNCTIONS
There are a number of built-in functions which may be used in the expression. Functions may be invoked using a syntax similar to C functions.

name **(** *arguments* **)**

You need to leave spaces around the parentheses so that they are separate command line arguments.

atime This function may be used to determine the last-accessed-time of a file. It takes one argument.

basename
This function returns the basename of the string argument passed to it. It takes one argument.

ctime This function may be used to determine the last-change-time of an inode. It takes one argument.

delete This function may be used to delete a file. It takes one argument. Always returns true. See also the **−delete** action, below.

execute This function may be used to execute a command. The arguments are assembled into the command to be executed. Use the special "{}" argument to insert the name of the current file. The function returns true of the command's exist status is zero. All following arguments to find are taken to be arguments to the command until an argument consisting of ";" is encountered. The command is executed in the starting directory. See also the **−execute** action, below.

gid This function may be used to determine the gid of a file. It takes one argument.

inode This function may be used to determine the inode number of a file. It takes one argument.

mode This function may be used to determine the access mode (permissions) of a file. It takes one argument.

mtime This function may be used to determine the last-modified-time of a file. It takes one argument.

print This function may be used to print a value. It takes one argument. Always returns true. See also the **−print** action, below.

size This function may be used to determine the size in bytes of a file. It takes one argument.

type This function may be used to determine the type of a file. It takes one argument. It returns a string: "block_special", "character_special", "directory", "file", "named_pipe", "socket" or "symbolic_link".

uid This function may be used to determine the uid of a file. It takes one argument.

TESTS

Most tests exist to provide compatibility with *find*(1).

−Access_Minutes [*relative-operator*] *number*

True if the current file was accessed exactly *number* minutes ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(now − atime ({+})) / 60 *relative-operator number*" expression.

−Access_Time [*relative-operator*] *number*

True if the current file was accessed exactly *number* days ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(now − atime ({+})) / 86400 *relative-operator number*" expression.

−Change_Minutes *number*

True if the current file's inode was changed exactly *number* minutes ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(now − ctime ({+})) / 60 *relative-operator number*" expression.

−Change_Time *number*

True if the current file's inode was changed exactly *number* days ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(now − ctime ({+})) / 86400 *relative-operator number*" expression.

−Modify_Minutes *number*

True if the current file was modified exactly *number* minutes ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(now − mtime ({+})) / 60 *relative-operator number*" expression.

−Modify_Time *number*

True if the current file was modified exactly *number* days ago, false otherwise. If a relative operator is given (<, <=, ==, !=, > or >=) a relative comparison will be made, rather than the implicit equality test. This is *not* identical to the similar *find*(1) test. This is shorthand for the "(

now − mtime ({+})) / 86400 *relative-operator number*" expression.

−Newer *filename*
True if the current file was modified after the given file. This is shorthand for the "mtime ({+}) > mtime (*filename*)" expression.

−Name *pattern*
Base of file name (the path with the leading directories removed) matches shell pattern *pattern*. This is short-hand for the "*pattern* ~ basename ({})" expression.

−PAth *pattern*
File name matches shell pattern *pattern*. Note that the file name if affected by the **−resolve** option. This is short-hand for the "*pattern* ~ {}" expression.

−Type *string*
The file type matches the type given. This is shorthand for the "type ({}) == *string*" expression. Type names are matched similar to options:

Block	The file is a block special file.
Character	The file is a character special file.
Directory	The file is a directory.
File	The file is a normal file.
Link	The file is a symbolic link.
Pipe	The file is FIFO (a named pipe).
Socket	The file is a UNIX domain socket.

ACTIONS

−print This will print the full file name on the standard output, followed by a newline. The **−Resolve** option will affect what is printed. This is short-hand for the "print ({})" expression.

−delete This will delete the file, if it is in the development directory tree. This is short-hand for the "delete ({})" expression.

−execute *string...* **;**
The may be used to execute a command. This is short-hand for the "execute (*string* **:** ...)" expression.

EXIT STATUS

The *aefind* command will exit with a status of 1 on any error. The *aefind* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aefinish − finish a change

SYNOPSIS

aefinish [*option*...]
aefinish -Help
aefinish -VERSion

DESCRIPTION

The *aefinish* command is used to finish development or integration of a change set. It examines the state of the change set, and executes the necessary Aegis commands to advance the change set to the next state.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aefinish* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aefinish* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aefinish* command will exit with a status of 1 on any error. The *aefinish* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aefp − calculate file fingerprint

SYNOPSIS

aefp [*option*...][*filename*...]
aefp −Help
aefp −VERSion

DESCRIPTION

The *aefp* program is used to calculate the fingerprints of files. A fingerprint is a hash of the contents of a file. The default fingerprint is cryptographically strong, so the probability of two different files having the same fingerprint is less than 1 in 2**200.

The fingerprint is based on Dan Berstien <djb@silverton.berkeley.edu> public domain fingerprint 0.50 beta package 930809, posted to the alt.sources newsgroup. This program produces identical results; the expected test results were generated using Dan's package.

The fingerprint is a base-64-sanely-encoded fingerprint of the input. Imagine this fingerprint as something universal and permanent. A fingerprint is 76 characters long, containing the following:

1. A Snefru-8 (version 2.5, 8 passes, 512→256) hash. (Derived from the Xerox Secure Hash Function.)
2. An MD5 hash, as per RFC 1321. (Derived from the RSADSI MD5 Message-Digest Algorithm.)
3. A CRC checksum, as in the new cksum utility.
4. Length modulo 2^40.

The output format is not expected to be compatible with anything. However, options are available to produce the purported output of Merkle's snefru program, the purported output of RSADSI's mddriver −x, or the purported output of the POSIX cksum program.

If no files are named as input, the standard input will be used. The special file name "−" is understood to mean the standard input.

OPTIONS

The following options are understood:

−Checksum
Print the CRC32 checksum and length of the named file(s).

−Identifier
Print a condensed form of the fingerprint (obtained by performing a CRC32 checksum on the full fingerprint described above − a definite overkill). This is an 8-digit hexadecimal number, useful for generating unique short identifiers out of long names. The first character is forced to be a letter (g-p), so there is no problem in using the output as a variable name.

−Help
Provide some help with using the *aefp* program.

−Message_Digest
Print the RSA Data Security, Inc. MD5 Message-Digest Algorithm hash of the named file(s).

−Snefru Print the Snefru hash of the named file(s), derived from the Xerox Secure Hash Function.

−VERSion
Print the version of the *aefp* program being executed.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option.

The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aefp* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aefp* command will exit with a status of 1 on any error. The *aefp* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

Portions of this program are derived from sources from other people, sometimes with liberal copyrights, and sometimes in the public domain. These include:

Dan Bernstien
: See *common/fp/README* for details.

Gary S Brown.
: See *common/fp/crc32.c* for details.

RSA Data Security, Inc.
: See *common/fp/md5.c* for details.

Xerox Corporation
: See *common/fp/snefru.c* for details.

In addition to the above copyright holders, there have been numerous authors and contributors, see the named files for details. Files names are relative to the root of the *aegis* distribution.

NAME

aeget − Aegis CGI file access

SYNOPSIS

aeget

DESCRIPTION

The *aeget* command is used with Apache (or CGI conforming any other web server) to access the files of an Aegis project. The files are searched for along the appropriate search path, including all ancestor baslines, not just the baseline of the branch.

This is useful when developing web sites using Aegis.

Install

In order to use *aeget*(1), you need to copy it into your *cgi-bin* directory.

You may prefer to use a symbolic link, as this will be more stable across Aegis upgrades. However, this requires a corresponding *follow-symlinks* setting in your web server's configuration file.

Usage

Once *aeget*(1) is installed, files may be accessed via

http://localhost/cgi-bin/aeget/*project-name*/

If no project name is given, a list of projects will be generated. This will lead you through a series of menus, giving access to manu useful pages of information about your projects.

Cascading Style Sheets

The web interface uses Cascading Style Sheets. You can give the web interface a personalised look and feel, by creating stylesheets in the web server's Document Root directory. The interface will use it's default styles, then styles from a global style sheet called `aedefault.css`, and then styles from a project stylesheet called *projectname*`.css` (replace *projectname* with the name of the project).

There is an example style sheet in `/usr/local/share/aedefault.css` which demonstrates the style elements used. This particular stylesheet is not designed to be aesthetically pleasing, but to exercise all of the elements. Using this stylesheet unmodified will give psychedelic results. Use it as a template.

PROJECT ATTRIBUTES

You can set your own project specific page headers and footers by using the "html:meta", "html:body-begin" and "html:body-end" project specific attributes.

```
project_specific =
[
  {
    name = "html:body-begin";
       value = "<i>This text goes immediately after the
       &lt;BODY&gt; and before any text generated by
       <i>aeget</i>(1).</i>";
  },
  {
    name = "html:body-end";
       value = "<i>This text goes immediately before the
       &lt;/BODY&gt; and after all text generated by
       <i>aeget</i>(1).</i>";
  },
];
```

These fields may be used to customize your web pages for your project-specific or company-specific needs. Each project is configured independently.

CHANGE ATTRIBUTES

If you wish to prevent a change set appearing in the change set inventory used by *aedist −replay* to determine what needs to be downloaded, set the following change set attribute:

```
attribute =
```

```
[
  {
    name = "aeget:inventory:hide";
    value = "true";
  },
];
```

You must use the *aeca*(1) command for this, the *tkaeca*(1) command can not edit change set attributes.

DEBUGGING and TESTING

You can run the *aeget*(1) program from the command line if you set the appropriate environment variables. This is how you debug or test *aeget*(1) command.

REQUEST_METHOD
: This is how the script is being invoked. For *aeget*(1) command, this is always "`GET`".

SCRIPT_NAME
: This is the path of the script name, from the HTTP client's point of view. Typically this is "`/cgi-bin/aeget`".

PATH_INFO
: This is the portion of the URL between the script name and the question mark. For *aeget*(1) this is usually the project name or the project name and the change number. No project name will get you the project list page.

QUERY_STRING
: This the portion of the URL after the question mark.

The above will not means much if you are not familiar with CGI scripts. For the URL `http://localhost/cgi-bin/aeget/aegis.4.1.C10?menu` would have Apache set the following environment variables

```
REQUEST_METHOD=GET \
SCRIPT_NAME=/cgi-bin/aeget \
PATH_INFO=/aegis.4.1.C10 \
QUERY_STRING='menu' \
aeget
```

Output is written to stdout. Tests scripts can easily capture this and compare it with expected results. Make sure you avoid false negatives because of the date tacked onto the end of most pages.

Apache

If you see "serious server error" pages when accessing *aeget*(1) via a web server, the stderr text is usually available in the server's error log.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis.cgi − Aegis web interface script

SYNOPSIS

aegis.cgi

DESCRIPTION

The *aegis.cgi* command is used to interface between a web server and Aegis.

Deprecated

This scripts is DEPRECATED since a long time and has been removed from the aegis distribution as of aegis−4.24.1. The preferred way to publish an aegis aegis repository on the Web is using *aeget*(1).

Installation

If you have a Web server, you may like to install the Aegis web interface. You do this by copying the *aegis.cgi* script from */usr/local/bin/aegis.cgi* into your web server's *cgi-bin* directory. There is an *aegis.cgi.i* helper script, if you don't know where your web server's *cgi-bin* directory is.

You may prefer to use a symbolic link, as this will be more stable across Aegis upgrades. However, this requires a corresponding *follow-symlinks* setting in your web server's configuration file. (Use the *aegis.cgi.i −s* option.)

Within the `aegis.cgi` script, you may set the *AEGIS_PATH* environment variable, if you want it to be able to see more projects than just the global projects. You do this by creating a */usr/local/lib/aegis.cgi.conf* file (there isn't one, by default) and setting the *AEGIS_PATH* environment variable in it. This is a fragment of Bourne shell script, not just the name.

Usage

Once installed, it should be possible to see the project list by following this URL:

http://localhost/cgi−bin/aegis.cgi?

You may need to replace *localhost* with the machine's exact name.

Cascading Style Sheets

The web interface uses Cascading Style Sheets. You can give the web interface a personalised look and feel, by creating stylesheets in the web server's Document Root directory. The interface will use it's default styles, then styles from a global style sheet called `aedefault.css`, and then styles from a project stylesheet called *projectname*`.css` (replace *projectname* with the name of the project).

There is an example style sheet in `/usr/local/bin/aedefault.css` which demonstrates the style elements used. This particular stylesheet is not designed to be aesthetically pleasing, but to exercise all of the elements. Using this stylesheet unmodified will give psychedelic results. Use it as a template.

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegrep − print lines matching a pattern

SYNOPSIS

aegrep [*option*...] *pattern*
aegrep −Help
aegrep −VERSion

DESCRIPTION

The *aegrep* command is used to search the source files for lines containing a match to the given *pattern*. By default, *aegrep* prints the matching lines.

There is no need to name files on the command line, all the project and change source files are supplied automatically. All non-source files are ignored.

Most of the *grep*(1) options are understood, in their long form.

OPTIONS

The following options are understood:

−−After−Context=*number*
Print *number* lines of trailing context after matching lines. Places a line containing a group separator (−−) between contiguous groups of matches.

−−Before−Context=*number*
Print *number* lines of trailing context before matching lines. Places a line containing a group separator (−−) between contiguous groups of matches.

−−Byte−Offset
Print the 0-based byte offset within the input file before each line of output.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−−color Surround the matched (non-empty) strings, matching lines, context lines, file names, line numbers, byte offsets, and separators (for fields and groups of context lines) with escape sequences to display them in color on the terminal.

−−Context=*number*
Print *number* lines of output context. Places a line containing a group separator (−−) between contiguous groups of matches.

−−Count
Suppress normal output; instead print a count of matching lines for each input file. With **−−invert−match** option count non-matching lines.

−−extended−regexp
Interpret *pattern* as an extended regular expression.

−−Files−With−Matches
Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.

−−Files−WithOut−Matches
Suppress normal output; instead print the name of each input file from which no output would normally have been printed. The scanning will stop on the first match.

−−fixed−strings
Interpret *pattern* as a list of fixed strings, separated by newlines, any of which is to be matched.

-Help
This option may be used to obtain more information about how to use the *aegrep* program.

−−Ignore−Case
Ignore case distinctions in both the *pattern* and the source files.

−−Invert−Match
Invert the sense of matching, to select non-matching lines.

−−Initial−Tab
Make sure that the first character of actual line content lies on a tab stop, so that the alignment of tabs looks normal.

−−Line−Buffered
Use line buffering on output.

−−Line−Number
Prefix each line of output with the 1-based line number within its input file.

−−Line−Regexp

−−Maximum−Count=*number*
Stop reading a file after *number* matching lines.

−−no−messages
Suppress error messages about nonexistent or unreadable files.

−−Only−Matching
Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.

−−Perl−Regexp
Interpret *pattern* as a Perl regular expression.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−−Unix−Byte−Offset
Report Unix-style byte offsets.

−−Word−Regexp
Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegrep* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aegrep* command will exit with a status of 1 on any error. The *aegrep* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aefind(1)
search for files in directory hierarchy

grep(1) print lines matching a pattern

COPYRIGHT

aegrep version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis integrate begin − begin integrating a change

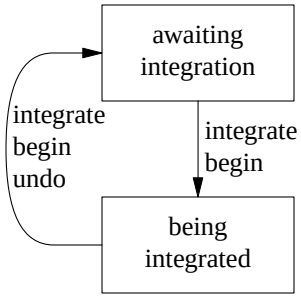
SYNOPSIS

aegis −Integrate_Begin *change-number* [*option*...]
aegis −Integrate_Begin −List [*option*...]
aegis −Integrate_Begin −Help

DESCRIPTION

The *aegis −Integrate_Begin* command is used to begin the integration of a change into the baseline of a project.

The change will advance from the *awaiting integration* state to the *being integrated* state.



A (logical) copy of the baseline is created in an *integration directory* and the the files of the change are added to the integration directory. The time stamps of files copied from the baseline are preserved, time stamps on the files copied from the development directory are all set to the time of the beginning of the integration. The *'aegis −Change_Directory'* command may be used to locate the integration directory. The change will be assigned to the current user.

Please note that only regular files and symbolic links are copied (linked) from the baseline to the integration directory. This has some implications:

- Special files (devices, named pipes, *etc*) will not be reproduced in the integration directory; you will need to create these as part of the build.
- If the case of the **−minimum** option (see below), only primary source files are copied (linked) across. Derived files (including symbolic links) are expected to be created as part of the build.
- If the case of the **−minimum** option, directories are only created when required to hold a file which satisfies the above criteria. If you need special empty directories, or directories which contain only special files, or only contain derived files, you need to create them as part of the build.

The *link_integration_directory* field of the project configuration file (see *aepconf*(5) for more information) controls whether the copy of the baseline is done by copying the files or by creating hard links to the files. The hard links are just one of the constraints on the location of the integration directory. The integrate begin will abort with an error if this copy operation fails, e.g. by running out of disk space. If this should happen, the change will remain in the *awaiting integration* state, and the integration directory will be removed.

The change will be assigned a delta number. Delta numbers are incremented once for each *aegis −Integrate_Begin* command for the project. If an integration is subsequently aborted with either the *aegis −Integrate_Begin_Undo* or *aegis −Integrate_FAIL* command, the delta number will not be re-used.

It is not possible to choose the integration directory, as there are many constraints upon it, including the fact that it must be on the same device as the baseline directory, and that many UNIX implementations don't allow renaming directories up and down the trees. The integration directory will be in the project directory, and named for the delta number.

Notification

On successful completion of this command, the *integration_begin_command* field of the project *config* file is run, if set. See *aepconf*(5) for more information.

Minimum Integrations

Aegis provides a **minimum** integration capability which may be used for various reasons. The term **minimum** may be a bit counter intuitive. One might think it means to the **minimum** amount of work, however it actually means use a **minimum** of files from the baseline in populating the *delta* directory. This normally leads to actually building everything in the project from sources and, as such, might be considered the most robust of builds.

Note that any change which removes a file, whether by *aerm*, *aemv* or *aemt*, results in an implicit **minimum** integration. This is intended to ensure nothing in the project references the removed file.

A project may adopt a policy that a product release should be based on a minimum integration. Such a policy may be a reflection of local confidence, or lack thereof, in the project's DMT (Dependency Maintenance Tool) or build system. Or it may be based on a validation process wishing to make a simple statement on how the released package was produced.

Another, more transient, reason a to require a minimum integration might be when upgrading a third party library, compiler or maybe even OS level. Any of these events would signal the need for a minimum integration to ensure everything is rebuilt using the new resources.

The cost of a **minimum** integration varies according to type and size of the project. For very large projects, especially those building large numbers of binaries, the cost can be large. However large projects also require significant time to fully populate the delta directory. A minimum integration only copies those files under Aegis control, skipping all "produced" files. In the case where a file upon which everything depends is changed, everything will be built anyway so the copy of the already built files is a waste of time. This means that sometimes a minimum can be as cheap as a normal integration.

Change Set Attributes

The follwoing user-defined change set attributes are understood:

integrate-begin-hint
: If this is set to "minimum" or "maximum", it is as if these options appeared on the command line. Only consulted if neither −minimum nor −maximum appear on the command line.

All other user defined change set attributes are ignored.

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

-List
: This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-MAXimum
: This option may be used to cause all files to be copied into the integration directory. This is the default, unless the change requires the deletion of a file.

-MINImum
: This option may be used to cause only the source files to be copied into the integration directory. The default is to copy all files, unless the change requires the deletion of a file.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "**−−***option***=***value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeib 'aegis −ib \!* −v'
sh$ aeib(){aegis −ib "$@" −v}

ERRORS

It is an error if the change is not in the *awaiting integration* state.
It is an error if the current user is not an integrator of the project.
It is an error if there is an integration in progress for the project.
It is an error if the current user developed the change and the project is configured to disallow developers to integrate their own changes (default).
It is an error if the current user reviewed the change and the project is configured to disallow reviewers to integrate their such changes (default).

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeb(1) build a change

aecd(1) change directory

aeibu(1) reverse the aeib command

aeifail(1)
fail integration of a change

aeintegratq(1)
Automate the integration queue.

aeipass(1)
pass integration of a change

aeni(1) add new integrators to a project

aerpass(1)
pass review of a change

aet(1) run tests

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis integrate begin undo − reverse the aeib command

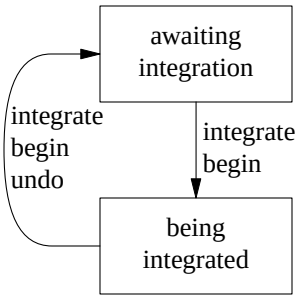
SYNOPSIS

aegis −Integrate_Begin_Undo [*option*...]
aegis −Integrate_Begin_Undo −List [*option*...]
aegis −Integrate_Begin_Undo −Help

DESCRIPTION

The *aegis −Integrate_Begin_Undo* command is used to reverse the actions of the *'aegis −Integrate_Begin'* command.

Successful execution of this command will move the change from the *being integrated* state to the *awaiting integration* state. The integration directory will be deleted. The change will cease to be assigned to the current user.



In the unlikely event that an integrator has wandered away and left an integration incomplete (say, went on holidays and won't be back for two weeks), project administrators are also able to use this command.

Notification

On successful completion of this command, the *integration_begin_undo_command* field of the project *config* file is run, if set. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf* (5) for more

information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeibu 'aegis −ibu \!* −v'
sh$ aeibu(){aegis −ibu "$@" −v}

ERRORS

It is an error if the change is not in the *being_integrated* state.
It is an error if the change is not assigned to the current user and the current user is not a project administrator.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeib(1) begin integration of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis integrate fail − fail a change integration

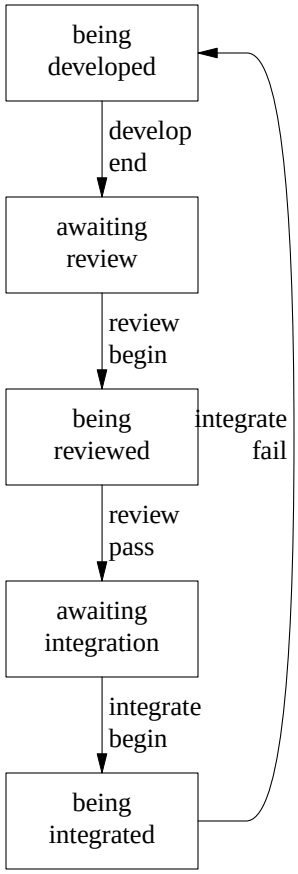
SYNOPSIS

aegis −Integrate_Fail −File *reason-file* [*option*...]
aegis −Integrate_Fail −REASon *'reason-text'* [*option*...]
aegis −Integrate_Fail −Edit [*option*...]
aegis −Integrate_Fail −List [*option*...]
aegis −Integrate_Fail −Help

DESCRIPTION

The *aegis −Integrate_Fail* command is used to inform aegis that a change has failed integration.

The change will be returned from the *being integrated* state to the *being developed* state. The change will cease to be assigned to the current user, and will be reassigned to the originating developer. The integration directory will be deleted.



The reviewer and the developer will be notified by mail. See the integrate_fail_notify_command in *aepconf*(5) for more information.

The *reason-file* will contain a description of why the change was failed. The file is in plain text. It is recommended that you only use newline to terminate paragraphs, (rather than to terminate lines) as with will result in better formatting in the various listings.

Notification

On successful completion of this command, the *integrate_fail_notify_command* field of the project attributes is run, if set. See *aepattr*(5) and *aepa*(1) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Edit

Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround

Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_-command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*

Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-Keep

This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-No_Keep

This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf* (5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*

This option may be used to provide the failure reason on the command line, rather than in a file. You will need to use quotes to insulate the spaces from the shell.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be

added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeifail 'aegis −ifail \!* −v'
sh$ aeifail(){aegis −ifail "$@" −v}

ERRORS

It is an error if the change is not in the *being integrated* state.
It is an error if the change is not assigned to the current user.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeib(1) begin integration of a change

aeipass(1)
pass integration of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeimport − import foreign repository into Aegis

SYNOPSIS

aeimport [*option*...] *dirname*
aeimport −Help
aeimport −VERSion

DESCRIPTION

The *aeimport* command is used to create a new project, and populate it by importing a foreign repository (such as RCS or CVS) without loss of project history.

Please note: unless you specify a version (see the **−version** option, below) this command will default to creating branches to support version 1.0. If you discovered this too late, all is not lost: you can use the *aenbru*(1) command to get rid of the branches you didn't want.

Directory

The project directory, under which the project baseline and history and state and change data are kept, will be created at this time. If the **−DIRectory** option is not given, the project directory will be created in the directory specified by the default_project_directory field of *aeuconf*(5), or if not set in current user's home directory; in either case with the same name as the project.

Staff

The project is created with the current user and group as the owning user and group. The current user is an administrator for the project. The project has no other administrators (use *aena*(1) to add more).

The project will have all user names found in the history files (see blow) installed as developers, reviewers and integrators. This is probably too broad, but fairly accurately reproduces the wide-open permissions found in most repositories, and you will want to use *aerd*(1), *aerrv*(1) and *aeri*(1) as appropriate to winnow this list.

If only one name is found, the project will be set to "`developers_may_review = true;`" otherwise it will be false (see *aepattr*(5) for more information). Use *aepa*(1) to change this if you want a different setting.

The project's umask is derived from the current user's umask, but modified to guarantee that group members will have access and that only the project owner will have write access. In general, it's best of the project is *not* owned by an account with any other role, as this prevents a whole class of "oops, I thought I was somewhere else" errors.

The project's history commands (see *aepconf*(5) for more information) are set to those suitable for RCS. The build command is set to "exit 0"; you need to set it to something suitable. The symbolic link farm is turned on.

Pointer

The project pointer will be added to the first element of the search path, or */usr/local/com* if no path is set. If this is inappropriate, use the **−LIBrary** option to explicitly set the desired location. See the **−LIBrary** option for more information.

Alternatively, unset the AEGIS_PATH environment variable to add the project to the global project list.

Version

You may specify the project version in two ways:

1. The version number may be implicit in the project name, in which case the version numbers will be stripped off. For example, "aeimport −p example.1.2" will create a project called "example" with branch number 1 created, and sub-branch 2 of branch 1 created.
2. The version number may be stated explicitly, in which case it will be subdivided for branch numbers. For example, "aeimport −p example −version 1.2" will create a project called "example" with branch number 1 created, and sub-branch 2 of branch 1 created.

In each case, these branches may be named wherever a project name may be given, such as "−p example.1"

and "−p example-1.2". The actual punctuation character is unimportant.

You may have any depth of version numbers you like. Both methods of specifying version numbers may be used, and they will be combined. If you want no version numbers at all, use **−version** with a single dash as the argument, as in "`−version −`"

If no version number is given, either explicitly or implicitly, version 1.0 is used.

Project Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

THE PROCESS

Most file version systems do not operate using change sets. In order to import such repositories into Aegis it is necessary to "discover" these change sets. The following steps are taken:

1. The directory (*dirpath*) given on the command line, and all directories below it, are scanned for appropriate files (for example, RCS and CVS use files with a "`,v`" suffix). These files are read to obtain the file's history.

   If you have been using a non-standard file suffix, aeimport won't be able to find the files.

   If you have more than one module in your CVS repository, aeimport doesn't (yet) understand the CVSROOT/modules file. Pointing aeimport at your whole CVSROOT may produce an unexpectedly large result.

2. The history files discovered in the previous step are copied into the location used by Aegis. Unlike some other tools, Aegis has a repository per project, rather than all projects sharing the same repository.

   This also means that Aegis will not modify the original history files. In particular, if the import produces unexpected results, simply remove the project (see *aermpr*(1) for more information) and start again.

   It is not possible to leave all your history files under, say, $CVSROOT and have Aegis point to them.

3. For each user mentioned in the various file histories, the time stamps are examined to find groups of files which were committed at around the same time. Files changed within 1 minute of each other are considered a group.

   Files change within one minute, but by different users, are *not* considered a group. This does not usually present a problem as developers mostly work alone. In rare cases where developers work together, only one of them does the commit.

   In some cases the time window may be too large, and several very small changes may be seen as one larger change set. In practice, this isn't very common.

4. Groups of files are stored into the Aegis database as completed changes (i.e. as if *aeipass*(1) has already run). The description of the change is the concatenation of all the unique comments found attached to the relevant file versions. The time stamp used for the change is the latest time stamp of any file in the group.

   There are times when small typographical errors between file comments result in longer-than-expected change descriptions. This can be corrected with *aeca*(1) or *tkaeca*(1) if desired. There are also times

when the reverse is true: some files have no comments at all, and the resulting description is less than useful.

5. Tags are turned into delta names by transferring delta names from the files they are attached to, to the change sets they are attached to. When a tag would appear to be attached to more than one change, it is attached only to the latest change.

   In common usage, the tags serve a similar purpose as Aegis' delta numbers. They are all (typically) applied in a single CVS command, in order that a particular release may be recreated later. However, because each file will be at a different version, and each will have had its latest version included in various random change sets.

   Tags are used for other things too. The method given here is simply a guess, but it's one which works reasonably well.

Once aeimport has completed importing a project, you will be able to examine the results using the *ael project_history* and *ael change_details* commands. (See *ael*(1) for more information.)

Limitations

The aeimport program is far from perfect. There are a number of known limitations.

- At this time, there is no support for branching. (As soon as I figure out how to discern the root of a branch across loosely coupled files, I'll implement it. Ideas and/or code contributions welcome.)
- Only RCS and SCCS formats are understood at present. It should be straight forward to add support for additional formats in the future. Only step 1 of the above process requires attention, the rest is file format neutral.
- There is no support for CVS modules, and there needs to be.
- You can't specify the time window size used to determine change sets. Time will tell whether this is necessary, but it begs the question: how will you know what window size you need in order to use the option at all.
- You can't import a CVS repository into an existing project. You may only create a new project from a CVS repository.
- You can't import a remote CVS repository.

OPTIONS

The following options are understood:

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

−FORmat *name*

This option may be use to specify which history format is being imported. The following formats are understood:

RCS Release Control System format has been around for quite a while. It is the format underlying CVS (Concurrent Version System). This is the default if no format name is specified.
Note: you *must* have RCS installed before you run *aeimport* if you use this format, because RCS commands will be run during the import process. The import will fail if RCS is not installed. You can find a freeware implementation at `ftp.gnu.org`, or a local mirror.

SCCS Source Code Control System is one of the earliest Unix version systems. (I'm told this is the format underlying BitKeeper.)
Note: you *must* have SCCS installed before you run *aeimport* if you use this format, because SCCS commands will be run during the import process. The import will fail if SCCS is not installed. The GNU Compatibly Stupid Source Control (CSSC) is a freeware implementation

of SCCS, and it may be found at `ftp://alpha.gnu.org/gnu/CSSC/`

−LIBrary *abspath*
This option may be used to specify a directory to be searched for global state files and user state files. (See *aegstate*(5) and *aeustate*(5) for more information.) Several library options may be present on the command line, and are search in the order given. Appended to this explicit search path are the directories specified by the *AEGIS_PATH* environment variable (colon separated), and finally, */usr/local/lib/aegis* is always searched. All paths specified, either on the command line or in the *AEGIS_PATH* environment variable, must be absolute.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-Help
This option may be used to obtain more information about how to use the *aeimport* program.

−VERSion *number*
This option may be used to specify the version number for the project. Version numbers are implemented as branches. Use a single dash ("−") as the argument if you want no version branches created.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aeimport* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aeimport* command will exit with a status of 1 on any error. The *aeimport* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeintegratq − integrate changes into projects

SYNOPSIS

aeintegratq [*option*...] *project-name*...

DESCRIPTION

The *aeintegratq* command is used to manage the integrations of one or more changes in one or more projects. Normally run via *cron*(1) or *at*(1) with the name of a single project, aeintegratq will manage all operations for integration even when **−Build** and **−Test** are required on multiple architectures. If a change review is revoked after the queue is running aeintegratq will notice the bad state and silently move on. If one or more changes are ended or passed after the queue is running, and −loop has been given, aeintegratq will notice the new change[s] and integrate them. Additional options allow the integrator full control over most aspects of queue management such as the order of integration of multiple changes.

OPTIONS

The following options are understood:

Option Summary

−h Help, show usage information.

−H Help, show usage plus all helpful comment information.

−a run on Any machine (normally only IntegrationHost)

−s run remote operations via ssh (default rsh)

−n No action, just tell what would be done.

−ib *s* Specify (remote) server for ibegin.

−ip *s* Specify (remote) server for ipass.

−k Keep the scripts and report files.

−K Keep the temp file even if integration passes.

−loop Loop to process more changes if they become available before aeintegratq completes. It will stop when there is nothing more to be done.

−M *list* Minimum, run given changes −*minimum*

−P *list* Precious, do not **IFail** changes in *list*, just stop.

−R *list* Ready, specify order and subset, *e.g.* −R 29,45

−S *stage* Pick up at given stage (diff|build|test|integrate)

−c *change-number*
specify Change to integrate at Stage

−p *project-name*
specify single Project name

NOTE: if custom options such as −P −R −S −c −p are given only a single project may be integrated since the options would be meaningless to the next project given.

Some options are present only for testing and investigation. Note that options are rarely required for normal operations.

Control Options

The following options are available for special needs. They control the order and disposition of each change **awaiting_integration** in a given project.

−R[eady] *number1,number2...*
This option is used to specify order or subset to integrate. Only those changes listed will be attempted, and in exactly the order given. This applies to queue looping if **−loop** is given. In particular note unless the list includes future changes, future loops will not integrate them.

Useful if a particular change must go in before another for some reason. Or if only integrating one or two changes when several are **awaiting_integration** in the given project. A single change may also be specified with the **−c[hange]** *number* option, which is common for other aegis commands. However the **−R** option allows a list and if given will override any **−c** given.

−P[recious] *number1,number2...*

−P[recious] **all**

This option is used to specify that a particular change or subset of changes should be considered **precious**. It neither implies order nor limits the queue run to that subset; it only means that the changes should be considered **precious**. Note that at least one number (or the keyword *all*) must be given.

The concept of **precious** means that if the given change were to fail anywhere in the integration process, then the process simply stops and leaves the problem change in the delta directory. The **−IFail** would not actually be executed. This is sometimes useful to diagnose a problem which only occurs during integrations. It is also useful if the failure is due to a transient problem such as unreliable machines on the network. In such a case the integration can be resumed after fixing the problem. See the *stage* options below.

If, on the other hand, a **precious** change makes it through the integration process successfully, the option has no effect.

−M[inimum] *number1,number2...* or *all*

Integrate the given change[s] with the **−minimum** option. Such changes will be put on the end of the queue so that the last integrations of a run will be a minimum. This feature allows practical use of minimum integrations without requiring **−minimum** on each and every integration. See the section below on *Minimum integrations* for more information. If **−loop** is given any change[s] specified as minimum will run at the end of the loop in which they are ready, they will not be pushed to the final loop.

−ib[server] *server-name* or ""

−ip[server] *server-name* or ""

To specify a remote server on which to run **−ibegin** or **0 respecively. These options are rarely needed, but may be useful if a project is hosted on a different file server and has a large baseline. By having the −ibegin run on that server the network traffic would be greatly reduced and for large projects and/or slow networks can greatly reduce the time required for −ibegin. The option form of giving an empty name depends on the output of df −k giving a parseable host name. If that is not true on your integration host architecture, you will have to specify the server name.**

−display *display-value* **or ""**

To specify a valid X display for use during integration operations.

Stage Options

The following options allow [re]starting an integration which has already progressed through some stages. This is useful to deal with failed (*precious*) integrations, or to finish automatically an integration begun by hand.

−S[tage] **diff**

−S[tage] **build**

−S[tage] **test**

−S[tage] **integrate**

Pick up the integration at the given **stage**. Requires **−c**[hange] *number* option to specify the change number.

Advanced Controls

The integrator may provide for special situations such as operations required after **−Build** and before **−Test**, or at the end of a queue run. Such capabilities are provided by **hooks** and **strategies** described below.

Hooks

There are a set of *hooks* available which are run, if present, before and after each stage of the integration. They can be used to help ensure that the integrator actually gets some sleep while managing large projects.

These hooks are searched for in the directory **$HOME/integration_hooks**. None need exist; aeintegratq will only pay attention to any that do exist. Hooks may be any form of executable (script, etc) and are called with 2 arguments: **project-name change-number**. They run as the integrator on the machine from which aeintegratq was started. They are named using the project name along with a suffix according to what place in the integration process you want them to run.

Note that if a hook for project **foo** exists it is also used for any branches under that project. For example, if you have provided **foo.pre_ip**, it will be run for foo.1 and foo.1.0 as well. If for some reason you want different (or no) action for project **foo.1.0**, then you would provide **foo.1.0.pre_ip** which does what you wish, including nothing, effectively overriding **foo.pre_ip**.

Here is how to map particular places in the integration process to hook suffixes.

run at time	extension
before attempting −Integrate_Begin	.pre_ib
after −Integrate_Begin completes	.ib
before attempting −Diff	.pre_d
after −Diff completes	.d
before attempting −Build	.pre_b
after −Build completes	.b
before attempting −Build on <arch>	.pre_<arch>b
after −Build on <arch> completes	.<arch>b
before attempting −Test	.pre_t
after −Test completes	.t
before attempting −IPass	.pre_ip
after −IPass completes	.ip
before attempting −IFail	.pre_if
after −IFail completes	.if

The hook program should exit with 0 if successful or 1 if not. A non-zero exit causes the change being integrated to fail immediately unless it was marked precious.

Note that in most cases anything done via an **.ip** hook should probably be done instead by the *ipass_notify* command in the project attributes file (see *aepattr*(5) for more information), or the *build_time_adjust_notify_command* in the project configuration file (see *aepconf*(5) for more information), but the hook can provide a temporary way to keep going until the permanent solution can be implemented.

In addition two special hooks, **aeintegratq.end** and **aeintegratq.fail**, are recognized. They are called when **aeintegratq** finishes a queue run. They are called with 2 arguments like any other hook (**project-name change-number**) although both the project-name and change-number given are of the last change integrated and may be less than useful.

The **.end** hook is called if/when the queue run is finished and was successful. Note that this does not mean that no changes failed, only that no queue errors occurred. This hook might be used to invoke another queue run on a different project/branch, or possibly even on the same project, if other changes may have been ended and/or reviewed while the first run was in progress, see also the **−loop** option. These conditions arise quite often with flex time engineers. Another use of the **.end** hook is to automatically build a new package using the newly integrated project as source.

If queue errors were encountered, or a change failed that was marked *precious*, then the **.fail** hook is called. An obvious use of that hook would be an e-mailed page to the integrator.

Strategy or Oops-retry

Sometimes a persistent build problem will plague integrations. This can be very annoying if it ruins an overnight run, especially if the cure is simple when it happens. Examples of this can be timeouts due to a busy data server or other transient errors. Note that this applies only to **−Build** related problems.

To deal with such problems the integrator may provide a *strategy* script specific to a project. An executable program should be found in **$HOME/strategy.<project_name>**. The program will be run as the integrator with the *delta* directory as current directory. The program may do any commands necessary to clean up and/or diagnose the error. If the script finds the problem to be transient and fix-able, it exits successfully (with 0 status) and aeintegratq will re-launch the **−Build** and log the re-try. Otherwise the script should exit with a 1 and the change will fail.

Multi-Architecture integrations

For projects which build and test on multiple architectures, aeintegratq requires *arch_hosts* be installed and have available at least one machine of each architecture required. This is also true if the host from which aeintegratq is run is of a different architecture from the target architecture of the project being integrated.

If you wish to take advantage of multiple architecture automatic integrations, you can install *arch_hosts* or provide a more simple script which will return a machine name according to architecture and job type.

Minimum integrations

provides a **minimum** integration capability which may be used for various reasons. The term **minimum** may be a bit counter intuitive. One might think it means to do the **minimum** amount of work, however it actually means use a **minimum** of files from the baseline in populating the *delta* directory. Since no constructed files are put in the *delta* directory, this normally leads to actually building everything in the project from sources and, as such, might be considered the most robust of builds.

Note that any change which removes a file, whether by *aerm* or *aemv*, results in an implicit **minimum** integration. This is intended to ensure nothing in the project references the removed file.

A project may adopt a policy that a product release should be based on a minimum integration. Such a policy may be a reflection of local confidence, or lack thereof, in the project's DMT (Dependency Maintenance Tool) or build system. Or it may be based on a validation process wishing to make a simple statement on how the released package was produced.

Another, more transient, reason a to require a minimum integration might be when upgrading a third party library, compiler or maybe even OS level. Any of these events would signal the need for a minimum integration to ensure everything is rebuilt using the new resources. This can be done with minimum overhead using the **−M** option as described above.

The cost of a **minimum** integration varies according to type and size of the project. For very large projects, especially those building large numbers of binaries, the cost can be large. However large projects also require significant time to fully populate the delta directory. A minimum integration only copies those files under aegis control, skipping all "produced" files. In the case where a file upon which everything depends is changed, everything will be built anyway so the copy of the already built files is a waste of time. This means that sometimes a minimum can be as cheap as a normal integration.

Manual Tests

allows tests to be defined as *manual* which may be necessary if the test requires human interaction or some transient resource. Such tests can be problematic for automatic integrations and generally must have some means to pass without running during integrations. For this, and other, reasons most sites seek to avoid *manual* tests. There are a number of ways to code a test such that it will pass automatically during integrations. Just one example for shell script tests might be:

```
CSTATE=‘aesub −p $AEGIS_PROJECT −c $AEGIS_CHANGE ’${state}’‘
if [ "$CSTATE" = "being_integrated" ]
then
   echo "‘basename $0‘ passes during integration"
   exit 0
fi
```

Optional Support Programs

There are some programs which aeintegratq will use if they are installed.

• *arch_hosts* was mentioned previously. It is optional only if your projects and your file server are of a single architecture.

• *aelogres* may enhance the information provided in −*IFail* entries. Normally all you get is the last 10 lines of the log file, which is not bad if tests fail, but can be terrible for failed builds. If you provide a program named *aelogres* which knows how to extract a better succinct report of problems, the output of that program will be used instead of the simple tail. It is called with a −*i* option.

• **sound_all_machines**, if available, will be called when integrations either pass or fail. It can be helpful to announce the fact that an integration has finished. If it passed, developers will probably want to do an **aed** to bring their changes up to date. The audio announcement provides another timely hint.

The sound files are searched for in the /usr/local/com/sounds directory. They will have endings of *_pass* and *_fail* according to the results of a given attempt. Two sound files are required: *integration_pass* and *integration_fail*. Others will be used if provided to customize the sounds so that each developer may have one or more personal sounds. If a file named <*developer*>*_pass* is located, it will be used. If a set of files exist named <*developer_pass.[0-9]* they will be used in random sequence. The same search rule applies to *_fail* sets. The *sound_all_machines* program may use a host list and play the sound file on each machine or, assuming that other audio capabilities exist, might do any form of announcement desired.

EXIT STATUS

The *aeintegratq* command will exit with a status of 1 on any error. The *aeintegratq* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

FILES

Control files are searched for in the *$HOME* directory. They are named strategy.<project>, They need not exist if no special action is necessary.

The hook scripts are searched for in the *$HOME/integration_hooks* directory. They are named <project>.<stage>. Also *aeintegratq.end* and *aeintegratq.fail*. These hooks also need not exist if no special action is desired.

COPYRIGHT

NAME

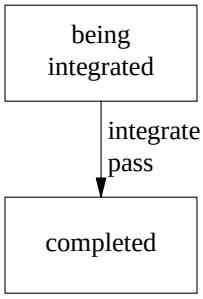
aegis integrate pass − pass a change integration

SYNOPSIS

aegis −Integrate_Pass [*option*...]
aegis −Integrate_Pass −List [*option*...]
aegis −Integrate_Pass −Help

DESCRIPTION

The *aegis −Integrate_Pass* command is used to notify aegis that a change has passed integration. The change is advanced from the *being integrated* state to the *completed* state.



This command updates the file histories, so that future *aecp*(1) commands may extract previous file versions from history, and so that future *aed*(1) commands may merge out-of-date files. The history is updated using the *history_create_command* and *history_put_command* fields of the project configuration file (see *aepconf*(5) for more information). The integrate pass will abort with an error if one of these history commands should fail, *e.g.* by running out of disk space. If this should happen, the change will remain in the *being integrated* state, and the integration directory is unaltered.

Once the history has been updated, the integration directory is renamed as the baseline directory, and the old baseline directory is deleted.

Once integrate pass is complete the change is no longer assigned to the current user.

History Tools Modify Files

Many history tools (*e.g.* RCS and SCCS) can modify the contents of the file when it is committed. This usually requires the use of specific “keyword” strings, and there are usually options to turn this behavior off, but users familiar with version control tools (as opposed to configuration management systems) will often use these features. The problem is that if the commit changes the file, the source file in the repository now no longer matches the object file in the repository. *I.e.* the history tool has compromised the referential integrity of the repository. By default, a fatal error is emitted if the file is changed by the check-in, however this can be modified to a be warning or even ignored completely; see the *history_put_trashes_file* field of *aepconf*(5) for more information.

File Modification Times

The modification times of all files modified since the beginning of integration (see *aeib*(1) for more information) are updated to be since the beginning of integrate pass. The order of modification times will be preserved, however the time range will be compressed to the greatest extent possible. This ensures that subsequent development builds will notice that baseline files have changed.

Note that if there are many new files with all different timestamps in the integration directory, and if the number of files with different timestamps exceeds the number of seconds since the start of the integrate-pass command, Aegis may have to set file modification times into the future.

The *build_time_adjust* field of the project *config* file controls Aegis’ behavior in this case. (See *aepconf*(5) for more information.) There are three settings:

adjust_and_sleep
: This setting, which is the default, causes Aegis to sleep until the file modification times would no longer be in the future. This avoids both development build problems and integration build problems, both of which which can arise as a result "interesting" file modification times.

adjust_only
: Aegis will issue a warning that the file modification times extend into the future, but will not sleep. This may cause integration build problems, particularly if you are using *aeintegratq*(1). Development builds may perform redundant builds, however *aet −reg* should not produce false negatives.

dont_adjust
: This is highly inadvisable. It is provided solely for some very rare circumstances. This setting causes Aegis not to adjust the file modification times at all. This can have very unhappy side-effects, especially of the integration build was *before* one or more development builds; the commonest symptom being that development builds do not always cause a relink of the necessary executables, and *aet −reg* may give false negatives. It is **strongly** recommended that you do not use this setting.

If you use *cook*(1), see the *time-adjust-back* flag for how to compress the time range even further. This usually makes the sleep (or the warning period) significantly shorter.

Notification

On successful completion of this command, after the directory rename has ocurred and after the database has been updated, the *integration_pass_notify_command* field of the project attributes is run, if set. See *aepattr*(5) and *aepa*(1) for more information. This command is run as the project owner.

Some compilers bury absolute path names into object files and executables. The renaming of the integration directory to become the new baseline breaks these paths. The above command is passed an environment variable called AEGIS_INTEGRATION_DIRECTORY so that the appropriate symlink may be placed, if desired.

Other commands run by this command include the *history_create_command*, *history_put_command* and *history_query_command* fields of the project *config* file. See *aepconf*(5) for more information.

THE BASELINE LOCK

The baseline lock is used to ensure that the baseline remains in a consistent state for the duration of commands which need to read the contents of files in the baseline.

The commands which require the baseline to be consistent (these include the *aeb*(1), *aecp*(1) and *aed*(1) commands) take a baseline *read* lock. This is a non-exclusive lock, so the concurrent development of changes is not hindered.

The command which modifies the baseline, *aeipass*(1), takes a baseline *write* lock. This is an exclusive lock, forcing *aeipass*(1) to block until there are no active baseline read locks.

It is possible that one of the above development commands will block until an in-progress *aegis −Integrate_PASS* completes. This is usually of short duration while the project history is updated. The delay is essential so that these commands receive a consistent view of the baseline. No other integration command will cause the above development commands to block.

When aegis' branch functionality is in use, a read (non-exclusive) lock is taken on the branch baseline and also each of the "parent" baselines. However, a baseline write (exclusive) lock is only taken on the branch baseline; the "parent" baselines are only read (non-exclusive) locked.

The History Lock

Where a project has a number of branches active simultaneously, it is possible for independent integrate pass commands for different branches to be issued very close together. The is an exclusive *history lock* taken by integrate pass to ensure that only one branch is updating the file history at a time, thus preventing history file corruption.

TEST CORRELATIONS

The “aegis −Test −SUGgest” command may be used to have aegis suggest suitable regression tests for your change, based on the source files in your change. This automatically focuses testing effort to relevant tests, reducing the number of regression tests necessary to be confident that you have not introduced a bug.

The test correlations are generated by the “aegis −Integrate_Pass” command, which associates each test in the change with each source file in the change. Thus, each source file accumulates a list of tests which have been associated with it in the past. This is not as exact as code coverage analysis, but is a reasonable approximation in practice.

The *aecp*(1) and *aenf*(1) commands are used to associate files with a change. While they do not actively perform the association, these are the files used by *aeipass*(1) and *aet*(1) to determine which source files are associated with which tests.

Test Correlation Accuracy

Assuming that the testing correlations are accurate and that the tests are evenly distributed across the function space, there will be a less than *1/number* chance that a relevant test has not been run by the “aegis −Test −SUGgest *number*” command. A small amount of noise is added to the test weighting, so that unexpected things are sometimes tested, and the same tests are not run every time.

Test correlation accuracy can be improved by ensuring that:

- Each change should be strongly focused, with no gratuitous file inclusions. This avoids spurious correlations.
- Each item of new functionality should be added in an individual change, rather than several together. This strongly correlates tests with functionality.
- Each bug should be fixed in an individual change, rather than several together. This strongly correlates tests with functionality.
- Test correlations will be lost if files are moved. This is because correlations are by name.

The best way for tests to correlate accurately with source files is when a change contains a test and exactly those files relating to the functionality under test. Too many spurious files will weaken the usefulness of the testing correlations.

METRICS

Aegis is capable of recording metrics as part of the file attributes of a change. This allows various properties of files to be recorded for later trend analysis, or other uses.

The specific metrics are not dictated by Aegis. It is expected that the integration build will create a metrics file for each of the source files the change. These metrics files must be in the format specified by *aemetrics*(5).

The name of the metrics file defaults to “*filename*,**S**”, however it may be varied, by setting the *metrics_filename_pattern* field of the project *config* file. See *aepconf*(5) for more information.

If such a metrics file exists, for each source file in a change, it will be read and remembered at integrate pass time. If it does not exist, Aegis assumes there are no relevant metrics for that file, and proceeds silently; it is not an error.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aeipass 'aegis −ipass \!* −v'
sh$ aeipass(){aegis −ipass "$@" −v}

ERRORS

It is an error if the change is not assigned to the current user.
It is an error if The change is not in the *being integrated* state.
It is an error if there has been no successful *'aegis −Build'* command for the integration.
It is an error if there has been no successful *'aegis −Test'* command for the integration.
It is an error if there has been no successful *'aegis −Test −BaseLine'* command for the integration.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeib(1) begin integration of a change

aeifail(1)
fail integration of a change

aemeasure(1)
simple file metrics

aemetrics(5)
metrics values file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis list − list (possibly) interesting things

SYNOPSIS

aegis −List [*option*...] *list-name*
aegis −List −List [*option*...]
aegis −List −Help

DESCRIPTION

The *aegis −List* command is used to list information. There are a number of possible *list-name*s, as follows (abbreviations as for command line options):

Administrators
: List the administrators of a project.

Branch_Details
: List full information about all the changes in the branch in large format. This listing will recurse down the full branch tree.

Change_Details
: List full information about a change in large format.

Change_Files
: List all files in a change. The verbose version includes details such as the action being taken, the edit number of the file, and whether it's being moved. The terse version only lists the files names (and omits removed files), this is useful for shell scripts and interfacing with build tools.

Change_File_History
: This listing shows the history of each file in the change. It includes each delta number (so that you may reproduce it) and brief description, of each change which affected each file.

Change_File_INventory
: This listing shows the filenames and their corresponding UUIDs. When a file is renamed, its UUID remains constant. (If the UUID column has the filename in it, this is a backwards compatibility value, for accessing the file history.)

Change_History
: List the history of a change.

Change_INventory
: List the changes of a project with their UUIDs.

Changes
: List the changes of a project. The verbose version includes details such as the state of the change and it's brief description. The terse version lists only the change numbers, which is good for shell scripts.

Default_Change
: List the default change for the specified user (defaults to the current user if no user specified).

Default_Project
: List the default project for the specified user (defaults to the current user if no user specified).

Developers
: List the developers of a project.

INComplete
: List the changes between the *awaiting review* and *being integrated* states, inclusive. Defaults to all users if no user name specified.

Integrators
: List the integrators of a project.

List_List
: List all lists available.

Locks
: List all currently active locks.

Outstanding_Changes
: List all changes owned by the specified user that are not yet completed (default to all users if no user specified).

All_Outstanding_Changes
: List all changes not yet completed, for all projects.

Project_Details
: List full information about all the changes in the project in large format. This listing will recurse down the full branch tree below the project.

Project_Files
: List all files in the baseline of a project. The verbose version includes details such as the action being taken, the edit number of the file. The terse version only lists the files names (and omits removed files), this is useful for shell scripts and interfacing with build tools. If a change number is given, files included in the change are omitted from the list (giving the change's perspective on what the project files are).

Project_File_INventory
: This listing shows the filenames and their corresponding UUIDs. When a file is renamed, its UUID remains constant. (If the UUID column has the filename in it, this is a backwards compatibility value, for accessing the file history.)

Project_History
: List the integration history of a project.

Projects
: List all projects.

Project_Aliases
: List all project aliases. If you use the **−Project** command line option, the list will only include aliases of the specified project, or the project of the specified alias.

Reviewers
: List the reviewers of a project.

State_File_Name
: Prints the absolute path of the project's or change's state file. Useful for cookbooks and makefiles.

Users_Changes
: List of changes owned by the specified user (defaults to current user if no user specified).

Version
: List version of a project or change. This includes the major and minor version number, and the previous version number if available. The list of copyright years is also printed.

Most of these lists are available from other aegis functions. Many aegis functions provide more specific lists.

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−PAGer The output of listings and help is piped through the pager command given in the PAGER environment variable (or *more* if not set). This is the default if the command is in the foreground, and the output is a TTY. This option may be used to override any preference specified in the *aeuconf* (5) file.

−No_PAGer
This option may be used to ensure that the output of listings and help is not piped through a pager command. This is the default if the command is in the background, or if the output is not a TTY. This option may be used to override any preference specified in the *aeuconf* (5) file.

−Page_Length *number*
This option may be used to set the page length of listings. The default, in order of preference, is obtained from the system, from the *LINES* environment variable, or set to 24 lines.

−Page_Width *number*
This option may be used to set the page width of listings and error messages. The default, in order of preference, is obtained from the system, from the *COLS* environment variable, or set to 79 characters.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf* (5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-UNFormatted
This option may be used with most listings to specify that the column formatting is not to be performed. This is useful for shell scripts.

-Page-Header
This option requests that page headings be present in listings and reports. This is the default.

-No-Page-Header
This option requests that page headings be omitted from listings and reports.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias ael 'aegis −l \!* −v'
sh$     ael(){aegis −l "$@" −v}
```

ERRORS

It is an error if the list name given is unknown.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aedb(1) begin development of a change (listing option)

aeib(1) begin integration of a change (listing option)

aelcf(1) list change files

aelpf(1) list project files

aer(1) report generator

aerpass(1)
pass review of a change (listing option)

aeuconf(5)
user configuration file format

aels(1) annotated directory listing

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aelcf − list change files

SYNOPSIS

aelcf [*option*...]
aelcf −Help
aelcf −VERSion

DESCRIPTION

The *aelcf* command is used to list the files which make up a change. The file names are printed one per line on the output.

If there are no files matching your criteria (see below) the output will be empty, and no error will be issued.

This is very similar to the *aegis −l cf* listing, but it only lists file names, it lists no other attributes, and it is considerably faster.

If your filenames have newlines in them, you have a problem. You can use any of the *posix_filename_charset*, *dos_filename_required*, *windows_filename_required*, or *shell_safe_filenames* fields in your project configuration file to prevent this. See *aenf*(1) and *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−ACtion *name*
This option may be used to specify which file actions you are interested in. Valid values are "create", "modify", "remove", *etc*, as may be observed in the Action column of the *aegis −l pf* listing. The default is to list files with all actions except removed files. You may use this option more than once.

−Not_ACtion *name*
This option may be used to exclude an action from the listing. If no actions are explicitly included or excluded, the default is to exclude removed files. You may use this option more than once.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aelcf* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−USAge *name*
This option may be used to specify which file usages you are interested in. Valid values are "source", "test", *etc*, as may be observed in the Usage column of the *aegis −l pf* listing. The default is to list files with all usages. You may use this option more than once.

−Not_USAge *name*
This option may be used to exclude usages from the listing. The default is to exclude no usages. You may use this option more than once.

−Quote-C
This option is used to request that each file name be quoted as C strings are quoted.

−Quote-COok
This option is used to request that each file name be quoted as *cook*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

−Quote-Shell
This option is used to request that each file name be quoted as *sh*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aelcf* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aelcf* command will exit with a status of 1 on any error. The *aelcf* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

ael(1) list interesting things

aelpf(1) list project files

aenf(1) add new files to be created by a change

aepconf(5)
project configuration file

COPYRIGHT

aelcf version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

In addition to the following license, as a special exception, the copyright holders give permission to link the code of this program with the *OpenSSL* library, and distribute linked combinations including the two. You must obey the GNU General Public License in all respects for all of the code used other than OpenSSL. If you modify file(s) with this exception, you may extend this exception to your version of the file(s), but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version. If you delete this exception statement from all source files in the program, then also delete it here.

The full text of the exception is available in the *LICENSE.openssl* file in the source distribution.

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program − to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product,

and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing

the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to

infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF

SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

> *one line to give the program’s name and a brief idea of what it does.*
> Copyright (C) *year name of author*
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author>
> This program comes with ABSOLUTELY NO WARRANTY; for details type “show w”. This is free software, and you are welcome to redistribute it under certain conditions; type “show c” for details.

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>. # vim: set ts=8 sw=4 et :

NAME

aelock − take a lock while a command runs

SYNOPSIS

aelock [*option*...] *command*
aelock −Help
aelock −VERSion

DESCRIPTION

The *aelock"* command is used to take a project lock while a command runs. This may be used to ensure that the project state is stable while it is being backed up.

The named command is looked for as an attribute called `aelock:`*command* within the *project_specific* field of the project configuration file.

The command is then passed through the usual *aesub*(5) substitutions before being executed. The command is executed as the project owner. If the command returns with a non-zero exit status, the *aelock*(1) command will return an exit status of one.

Security Issues

This command is a potential security problem. Because it takes a read-only lock of all active branches and changes in a project, from the trunk down, misuse of this command is a potential denial of service attack. Thus, this command is limited to project administrators only.

This command could have been designed to take an arbitrary command to execute, like *sudo*(1), but this would have granted users, even project administrators, more privileges than usual. For this reason, the command is held in a source controlled, fully reviewed project configuration file, and is simply indicated by name.

The command is run as the project owner, not the executing user. It has full write access (that's the way Unix permissions work). Like *aeb*(1), this means it can wreak havoc on the project baseline and meta-data. Use with extreme care.

OPTIONS

The following options are understood:

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-Help
This option may be used to obtain more information about how to use the *aelock* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aelock* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aelock* command will exit with a status of 1 on any error. The *aelock* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aelpf − list project files

SYNOPSIS

aelpf [*option*...]
aelpf −Help
aelpf −VERSion

DESCRIPTION

The *aelpf* command is used to list the files which make up a project. The file names are printed one per line on the output.

If there are no files matching your criteria (see below) the output will be empty, and no error will be issued.

This is very similar to the *aegis −l pf* listing, but it only lists file names, it lists no other attributes, and it is considerably faster.

If a change number is supplied on the command line, the files within that change will be excluded from the listing. This allows build tools to distinguish between change files which are changing, from project files which are not, if they care.

If your filenames have newlines in them, you have a problem. You can use any of the *posix_filename_charset*, *dos_filename_required*, *windows_filename_required*, or *shell_safe_filenames* fields in your project configuration file to prevent this. See *aenf*(1) and *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−ACtion *name*
This option may be used to specify which file actions you are interested in. Valid values are "create", "modify", "remove", *etc*, as may be observed in the Action column of the *aegis −l pf* listing. The default is to list files with all actions except removed files. You may use this option more than once.

−Not_ACtion *name*
This option may be used to exclude an action from the listing. If no actions are explicitly included or excluded, the default is to exclude removed files. You may use this option more than once.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aelpf* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−USAge *name*
This option may be used to specify which file usages you are interested in. Valid values are "source", "test", *etc*, as may be observed in the Usage column of the *aegis −l pf* listing. The default is to list files with all usages. You may use this option more than once.

−Not_USAge *name*
This option may be used to exclude usages from the listing. The default is to exclude no usages. You may use this option more than once.

−Quote-C
This option is used to request that each file name be quoted as C strings are quoted.

−Quote-COok
This option is used to request that each file name be quoted as *cook*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

−Quote-Shell
This option is used to request that each file name be quoted as *sh*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aelpf* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aelpf* command will exit with a status of 1 on any error. The *aelpf* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

ael(1) list interesting things

aelcf(1) list change files

aenf(1) add new files to be created by a change

aepconf(5)
project configuration file

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aels − annotated directory listing

SYNOPSIS

aels [*option*...][*filename*...]
aels −Help
aels −VERSion

DESCRIPTION

The *aels* command is used to list information about the files and directories named on the command line. If no files are named, the current directory is listed.

The view presented is from Aegis' perspective. It unifies the development directory with the baseline.

OPTIONS

The following options are understood:

−Recursive

−LOng This option implies the **−Show-Mode**, **−Show-Attributes**, **−Show-User**, **−Show-Group**, **−Show-Size** and **−Show-When** options, unless explicitly overridden.

−Show-Dot-Files
This option may be used to show files starting with a dot (.).

−Hide-Dot-Files
This option may be used to hide files starting with a dot (.). This is the default.

−Show-Removed-Files
This option may be used to include removed files in the listing.

−Hide-Removed-Files
This option may be used to exclude removed files from the listing. This is the default.

−Show-Mode
This option may be used to include the mode column in the listing. The mode column indicates the file type and permissions.

−Hide-Mode
This option may be used to exclude the mode column from the listing. This is the default.

−Show-Attributes
This option may be used to include the attributes column in the listing. The attributes column indicates whether the file is part of the change, the project or neither; whether the file is being created, modified or removed; whether the file is a source file, a test file, or neither. This is the default.

−Hide-Attributes
This option may be used to exclude the attributes column from the listing.

−Show-User
This option may be used to include file owner information in the listing.

−Hide-User
This option may be used to exclude file owner information from the listing. This is the default.

−Show-Group
This option may be used to include file group information in the listing.

−Hide-Group
This option may be used to exclude file group information from the listing. This is the default.

−Show-Size
This option may be used to include file size information in the listing.

−Hide-Size
This option may be used to exclude file size information from the listing. This is the default.

−Show-When
This option may be used to include the file modification time in the listing.

−Hide-When
This option may be used to exclude the file modification time from the listing. This is the default.

-Help
This option may be used to obtain more information about how to use the *aels* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aels* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aels* command will exit with a status of 1 on any error. The *aels* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

ls(1) list directory contents

aefind(1)
search for files in directory hierarchy

ael cf List change files.

ael pf List project files.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aelsf − list source files

SYNOPSIS

aelsf [*option*...]
aelsf −Help
aelsf −VERSion

DESCRIPTION

The *aelsf* command is used to list the source files of a change, including project source files and change source files. The file names are printed one per line on the standard output.

If there are no files matching your criteria (see below) the output will be empty, and no error will be issued.

This is very similar to the *aelpf*(1) and *aelcf*(1) commands, almost as if you ran both, but as a single command.

If your filenames have newlines in them, you have a problem. You can use any of the *posix_filename_charset*, *dos_filename_required*, *windows_filename_required*, or *shell_safe_filenames* fields in your project configuration file to prevent this. See *aenf*(1) and *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−ACtion *name*
This option may be used to specify which file actions you are interested in. Valid values are "create", "modify", "remove", *etc*, as may be observed in the Action column of the *aegis −l pf* listing. The default is to list files with all actions except removed files. You may use this option more than once.

−Not_ACtion *name*
This option may be used to exclude an action from the listing. If no actions are explicitly included or excluded, the default is to exclude removed files. You may use this option more than once.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aelsf* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−USAge *name*
This option may be used to specify which file usages you are interested in. Valid values are "source", "test", *etc*, as may be observed in the Usage column of the *aegis −l pf* listing. The default is to list files with all usages. You may use this option more than once.

−Not_USAge *name*
This option may be used to exclude usages from the listing. The default is to exclude no usages. You may use this option more than once.

−RESOlve
This option may be used to request the absolute path of each of the source files. This is helpful when using *xargs*(1) or *grep*(1).

−Quote-C
This option is used to request that each file name be quoted as C strings are quoted.

−Quote-COok
This option is used to request that each file name be quoted as *cook*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

−Quote-Shell
This option is used to request that each file name be quoted as *sh*(1) strings are quoted. When no quoting is required for individual files, the file name will not be quoted.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aelsf* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aelsf* command will exit with a status of 1 on any error. The *aelsf* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aelcf(1) list change files

aelpf(1) list project files

aenf(1) add new files to be created by a change

aepconf(5)
project configuration file

COPYRIGHT

aelsf version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aemakegen − generate a Makefile.in from file manifest

SYNOPSIS

aemakegen [*option*...][*filename*...]
aemakegen −Help
aemakegen −VERSion

DESCRIPTION

The *aemakegen* command is used to generate a `Makefile.in` file from a file manifest. The search path and file manifest is derived from Aegis meta-data. File names on the command line are considered to be additional files, and will be added to the manifest.

Project Structure

The *aemakegen* command assumes a particular project structure. This is as follows:

`lib/` The *lib* directory contains C++ files to be compiled, and placed into the `lib/lib.a` file, to be linked with the other executables. (You can override this with the `aemakegen:library-directory` project specific attribute.)

`lib`*project*`/`
: An alternative name to *lib*, above.

  If there is a `lib`*project*`/lib`*project*`.h` file, this is installed as `$(prefix)/include/lib`*project*`/lib`*project*`.h` and any project file it includes in turn are also installed below `$(prefix)/include/lib`*project*`/`

prog`/`
: The source for each executable is contained in its own directory. Which directories contain programs are determined by the presence of a `main.c` or `main.cc` file.

  As a special case, files named `test/`*name*`/*` will be linked as an executable `bin/test_`*name*

`bin/` Each program is compiled and linked, with the executable placed in the *bin* directory.

`datadir/`
: These files will be installed into the `$(DATADIR)/`*project-name*`/` directory.

`datarootdir/`
: These files will be installed into the `$(DATAROOTDIR)/` directory. This is usually meta-data to tell other packages about this package's existence.

`libdir/`
: These files will be installed into $(LIBDIR)/

`test_*/`
: These commands are expected to be in support of the `check` target and are compiled but not installed. (Can use a minus rather than an underscore, if you prefer.)

`noinst_*/`
: These commands are expected to be in support of the build, or the `check` target, and are compiled but not installed. (Can use a minus rather than an underscore, if you prefer.)

configure.ac

If the *configure.ac* file contains certain lines, additional features will be added to the file. These include:

AC_CHECK_PROGS(GROFF,
: The project uses the *groff*(1) and the GNU Groff documentation suite.

AC_CHECK_PROGS(LIBTOOL, ...
: This will cause the library to be built as a shared library, and installed so as to make it accessible to the programs linked against it. Note that you can set the project_specific attribute `aemakegen:libtool` to true for the same effect.

AC_CHECK_PROGS(SOELIM,
: One of the programs in the GNU Groff documentation suite.

AC_EXEEXT
: The makefile defines the $(EXEEXT) macro, for executable file extensions.

AC_LANG([C])
: The source files are all assumed to be in C.

AC_LANG([C++])
: The source files are all assumed to be in C++.

AC_LANG_C
: The source files are all assumed to be in C.

AC_LANG_CPLUSPLUS
: The source files are all assumed to be in C++.

AC_LIBEXT
: The makefile defines the $(LIBEXT) macro, for library file extensions.

AC_OBJEXT
: The makefile defines the $(OBJEXT) macro, for object file extensions.

AC_PROG_LIBTOOL
: Synonym for the longer libtool form, above.

AC_PATH_XTRA
: The project uses the X11 window system.

Project Attributes

The following `project_specific` attributes are known:

aemakegen:debian:brief–description:*package*
: Used by the debian target to set the first line of the Description field of each package.

aemakegen:debian:build-depends
: Used by the debian target to set the Build-Depends.

aemakegen:debian:conflicts:*package*
: Used by the debian target to set the Conflicts field of each package.

aemakegen:debian:description:*package*
: Used by the debian target to set the Description field of each package.

aemakegen:debian:dm-upload-allowed
: Used by the debian target. If true, the DM-Upload-Allowed field will be set to yes.

aemakegen:debian:homepage
: Used by the debian target to set the homepage. Omitted of not set.

aemakegen:debian:priority
: Used by the debian target to set the priority. Defaults to "extra" if not set.

aemakegen:debian:maintainer
: Used by the debian target to set the maintainer.

aemakegen:debian:provides:*package*
: Used by the debian target to set the Provides field of each package.

aemakegen:debian:recommends:*package*
: Used by the debian target to set the Recommends field of each package.

aemakegen:debian:replaces:*package*
: Used by the debian target to set the Replaces field of each package.

aemakegen:debian:section
: Used by the debian target to set the section. Defaults to "unknown" if not set.

aemakegen:debian:suggests:*package*
: Used by the debian target to set the Suggests field of each package.

aemakegen:libtool
: Boolean, whether or not to use *libtool*(1) to build the project's libraries. This is of most interest to projects which build shared libraries.

aemakegen:rpm-spec:build-requires
: Additional packages required to build the project.

aemakegen:rpm-spec:description
: A description of the project.

aemakegen:version-info
: String; the shared library's version number (completely different and separate to the project version, see *libtool*(1) for discussion). Three colon-separated decimal numbers. Defaults to 0:0:0 if not set.

Change Set Attributes

The following change set attributes are known:

aemakegen:debian:accepted
: Normally, when the "debian/changelog" file is written, it gathers up all of the Debian "Changed" information, and places it into the change-log entry for the first (*i.e.* most recent) change in the changelog. This ensures it will be transferred into the "*.changes" files. If a change set is marked with `aemakegen:debian:accepted=true`, it drops all of the "Closed" information, as this has already been processed by the Debian bug tracking system.

File Attributes

The following file attributes are known:

aemakegen:noinst
: boolean. If true, *aemakegen*(1) will not cause the program to be installed. Usually attached to the source file containing the *main* function, or to script files. Defaults to false if not defined (*i.e.* do install program).

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Output *filename*
: This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
: This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−SCRipt *pattern*
: This option may be used to nominate file which are scripts. The patterns are normal shell file name globbing patterns, so you may need to quote it. You may use this option more than once. Scripts in the `script/` or `scripts/` directories will be installed. Scripts with a basename starting with `test_` will be build to support the "make check" target.

−TArget *name*

The option may be used to select the desired output format by name. The known names are:

automake
: Generate *automake*(1) input, suitable for use as a top-level `Makefile.am` file.

makefile
: Generate *make*(1) input, suitable for as as a top-level `Makefile.in` file. This is the default.

debian
: Generate the `debian/` directory contents, which will exactly match the Makefile generated by the above two.

pkg-config
: Generate a *pkg-config*(1) configuration (`.pc`) file. It will exactly match the above targets, provided they expect to see this output in a `.pc` file in the manifest, or are given on on the command line.

rpm-spec
: Generate an RPM .spec file, for use with *rpm-build*(1).

-Help

This option may be used to obtain more information about how to use the *aemakegen* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aemakegen* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aemakegen* command will exit with a status of 1 on any error. The *aemakegen* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aemeasure − simple file metrics

SYNOPSIS

aemeasure [*infile* [*outfile*]]

DESCRIPTION

The *aemeasure* command is used to measure a few very simple file statistics: lines of code, lines of comments, blank lines. It is an example of a probgram which produces its output in the *aemetrics*(5) format, suitable for Aegis to read and understand as file metrics.

The language of the file is determined by examining the file suffix.

.c, .h, .y	C language
.cc, .CC, .c++,	C++ language
.man, .mm, .ms, .so	GNU Groff input

METRICS

Aegis is capable of recording metrics as part of the file attributes of a change. This allows various properties of files to be recorded for later trend analysis, or other uses.

The specific metrics are not dictated by Aegis. It is expected that the integration build will create a metrics file for each of the source files the change. These metrics files must be in the format specified by *aemetrics*(5).

The name of the metrics file defaults to "*filename***,S**", however it may be varied, by setting the *metrics_filename_pattern* field of the project *config* file. See *aepconf*(5) for more information.

If such a metrics file exists, for each source file in a change, it will be read and remembered at integrate pass time. If it does not exist, Aegis assumes there are no relevant metrics for that file, and proceeds silently; it is not an error.

OPTIONS

The following option is understood

−LANGuage *name*

This option may be used to specify the input language of the file, rather than have the input language be guessed fromn the file suffix. The name must be one of the following: `C`, `C++`, `roff` or `generic`. Any other name will result in an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aemeasure* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aemeasure* command will exit with a status of 1 on any error. The *aemeasure* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeipass(1)
pass a change integration

aemetrics(5)
metrics values file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis make transparent − make branch file transparent

SYNOPSIS

aegis −Make_Transparent [*option*...] *filename*...
aegis −List [*option*...]
aegis −Help
aegis −VERSion

DESCRIPTION

The *aegis −Make_Transparent* command is used to add a file to the change which will result in the named file being made transparent in the branch when the change is integrated. The version of the project file from the deeper branch will then show through.

You may name a directory to make all files in that directory tree transparent. It is an error if there are no relevant files.

Branch *vs* Change

The *aecpu*(1) command may only be applied to a change. If you wish to perform the same operation on a branch, use the *aemt*(1) command, through the agency of a change.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because changing the change file list logically invalidates them.

Notification

The *make_transparent_command* in the project configuration file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−UNChanged
Examine the named files to see if they are unchanged. Only remove the unchanged files from the branch, and leave the files which have changed. If no files are named on the command line all branch files are checked.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aemt 'aegis −mt \!* −v'
sh$ aemt(){aegis −mt "$@" −v}

This command is dedicated to my wife, Mary-Therese. When I was trying to explain what this command did, she said "I had one of those on my bicycle when I was young, but it fell off".

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is not present in the immediate branch.
It is an error if the file is present in the immediate branch, but is not also present in a deeper branch.
It is an error if the file is only present in the trunk branch.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecpu(1)
uncopy copy files from a change

aemtu(1)
cease making branch files transparent

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis make transparent undo − no longer make branch file transparent

SYNOPSIS

aegis −Make_Transparent_Undo [*option*...] *filename*...
aegis −Make_Transparent_Undo −List
aegis −Make_Transparent_Undo −Help

DESCRIPTION

The *aegis −Make_Transparent_Undo* command is used to reverse the effects of the *aegis −Make_Transparent* command. The named files will be removed from the list of files in the change.

The file is deleted from the development directory unless the **−Keep** option is specified. The **−Keep** option should be used with great care, as you can confuse tools such as *make*(1) by leaving these files in place.

You may name a directory to delete from the change all files in that directory tree previously copied into the change, other files in the tree will be ignored. It is an error if there are no relevant files.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because deleting a file logically invalidates them. If the project *config* file was deleted, any diff registration will also be canceled.

The difference file (`,D`) will also be removed, however any DMT derived files (e.g a `.o` file from a `.c` file) will not be removed. This is because aegis is decoupled from the DMT, and cannot know what these derived file may be called. You may need to delete derived files manually.

Notification

The *make_transparent_undo_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-Interactive
Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links
This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see

aeuconf(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aemtu 'aegis −mtu \!* −v'
sh$     aemtu(){aegis −mtu "$@" −v}
```

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecpu(1)
no longer modify a file

aemt(1) make branch files transparent

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis move file − rename one or more files as part of a change

SYNOPSIS

aegis −MoVe_file [*option*...] *old-file-name new-file-name* [*old1 new1* [*old2 new2* ...]]
aegis −MoVe_file −List [*option*...]
aegis −MoVe_file −Help

DESCRIPTION

The *aegis −MoVe_file* command is used to copy a file into a change and change its name at the same time.

The named files will be copied from the baseline (*old-file-name*) into the development directory (*new-file-name*), and added to the list of files in the change.

Warning: If there is already files in the development directory of either the *old-name* or the *new-name* they will be overwritten.

The *old-file-name* in the development directory will contain 1KB of random text. The random text is sufficiently revolting that most compilers will give error messages, should the file be referenced accidentally. This is often very helpful when moving include files.

You may rename directories. All the files in the *old-name* directory tree will be renamed to be below the *new-name* directory tree.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because adding another file logically invalidates them.

When the change files are listed (*aegis −List Change_Files −TERse*) the new files (*new-file-name*) will appear in the listing, and the removed files (*old-file-name*) will **not** appear in the terse listing. Similarly, when the project files are listed with an explicit change number (*aegis −List Project_Files −TERse −Change* `N`) none of the change's files, including both the new and removed files, will appear in the terse listing. These two features are very helpful when calling aegis from within a DMT to generate the list of source files.

Notification

The *new_file_command* and *remove_file_command* in the project *config* file are run, if set. The *project_-file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

WHITEOUT

Aegis provides you with what is often called a "view path" which indicates to development tools (compilers, build systems, *etc*) look first in the development directory, then in the branch baseline, and so on up to the trunk baseline.

The problem with view paths is that in order to remove files, you need some kind of "whiteout" to say "stop looking, it's been removed."

When you user the *aerm*(1) or *aemv*(1) commands, this means "add information to this change which will remove the file from the baseline when this change is integrated". *I.e.* while the change is in the *being developed* state, the file is only "removed" in the development directory − it's still present in the baseline, and will be until the change is successfully integrated.

When you use the *aerm*(1) or *aemv*(1) commands, Aegis will create a 1K file to act as the whiteout. It's contents are rather ugly so that if you compile or include the "removed" file accidentally, you get a fatal error. This will remind you to remove obsolete references.

When the change in integrated, the removed file is *not* copied/linked from the baseline to the integration directory, and is *not* copied from the development directory. At this time it is physically gone (no whiteout). It is assumed that because of the error inducing whiteout all old references were found and fixed while the change was in the *being developed* state.

File Manifests

When generating list of files to be compiled or linked, it is important that the file manifest be generated from information known by Aegis, rather than from the file system. This is for several reasons:

(a) Aegis knows exactly what (source) files are where, whereas everything else is inferring Aegis' knowledge; and

(b) looking in the file system is hard when the view path is longer that 2 directories (and Aegis' branching method can make it arbitrarily long); and

(c) The whiteout files, and anything else left "lying around", will confuse any method which interrogates the file system.

The easiest way to use Aegis' file knowledge is with something like an *awk*(1) script processing the Aegis file lists. For example, you can do this with *make*(1) as follows:

```
# generate the file manifest
manifest.make.inc: manifest.make.awk
    ( aegis −l cf −ter ; aegis −l pf −ter ) | \
    awk −f manifest.make.awk > manifest.make.inc
# now include the file manifest
include manifest.make.inc
```

Note: this would be inefficient of you did it once per directory, but there is nothing stopping you writing numerous assignments into the *manifest.make.inc* file, all in one pass.

It is possible to do the same thing with Aegis' report generator (see *aer*(1) for more information), but this is more involved than the *awk*(1) script. However, with the information "straight from the horse's mouth" as it were, it can also be much smarter.

This file manifest would become out-of-date without an interlock to Aegis' file operations commands. By using the *project-file_command* and *change_file_command* fields of the project *config* file (see *aepconf*(5) for more information), you can delete this file at strategic times.

```
/* run when the change file manifest is altered */
change_file_command = "rm −f manifest.make.inc";
/* run when the project file manifest is altered */
project_file_command = "rm −f manifest.make.inc";
```

The new file manifest will thus be re-built during the next *aeb*(1) command.

Options and Preferences

There is a **−No-WhiteOut** option, which may be used to suppress whiteout files when you use the *aerm*(1) and *aemv*(1) commands. There is a corresponding **−WhiteOut** option, which is usually the default.

There is a *whiteout_preference* field in the user preferences file (see *aeuconf*(5) for more information) if you want to set this option more permanently.

Whiteout File Templates

The *whiteout_template* field of the project *config* file may be used to produce language-specific error files. If no whiteout template entry matches, a very ugly 1KB file will be produced − it should induce compiler errors for just about any language.

If you want a more human-readable error message, entries such as

```
whiteout_template =
[
    {
        pattern = [ "*.[ch]" ];
        body = "#error This file has been removed.";
    }
];
```

can be very effective (this example assumes *gcc*(1) is being used).

If it is essential that *no* whiteout file be produced, say for C source files, you could use a whiteout template such as

```
whiteout_template =
[
    { pattern = [ "*.c" ]; }
];
```

because an absent *body* sub-field means generate no whiteout file at all.

You may have more than one whiteout template entry, but note that the order of the entries is important. The first entry which matches will be used.

Notification

On successful completion of this command, the notifications usually performed by the *aerm*(1), *aenf*(1) and *aent*(1) commands are run, as appropriate. These include the *project_file_command*, *new_file_command*, *new_test_command* and *remove_file_command* fields of the project *config* file. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging

This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−WhiteOut
This option may be used to request that deleted files be replaced by a "whiteout" file in the development directory. The idea is that compiling such a file will result in a fatal error, in order that all references may be found. This is usually the default.

−No_WhiteOut
This option may be used to request that no "whiteout" file be placed in the development directory.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aemv 'aegis −mv \!* −v'
sh$ aemv(){aegis −mv "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if either file is already in the change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy files into a change

aedb(1) begin development of a change

aemvu(1)
undo the rename files as part of a change

aenf(1) add files to be created by a change

aenfu(1) remove files to be created by a change

aerm(1) add files to be deleted by a change

aermu(1)
remove files to be deleted by a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis move file undo − undo the rename a file as part of a change

SYNOPSIS

aegis −MoVe_file_Undo [*option*...] *filename*...
aegis −MoVe_file_Undo −List [*option*...]
aegis −MoVe_file_Undo −Help

DESCRIPTION

The *aegis −MoVe_file_Undo* command is used to reverse the effects of the *aegis −MoVe_file* command. You only need to name one half of the rename, the other half will be determined automatically. You may apply this command to whole directories.

The named files will be removed from the development directory, and removed from the list of files in the change.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because adding another file logically invalidates them.

Notification

The *new_file_undo_command* and *remove_file_undo_command* in the project *config* file are run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

WHITEOUT

Aegis provides you with what is often called a “view path” which indicates to development tools (compilers, build systems, *etc*) look first in the development directory, then in the branch baseline, and so on up to the trunk baseline.

The problem with view paths is that in order to remove files, you need some kind of "whiteout" to say “stop looking, it’s been removed.”

When you user the *aerm*(1) or *aemv*(1) commands, this means "add information to this change which will remove the file from the baseline when this change is integrated". *I.e.* while the change is in the *being developed* state, the file is only "removed" in the development directory − it’s still present in the baseline, and will be until the change is successfully integrated.

When you use the *aerm*(1) or *aemv*(1) commands, Aegis will create a 1K file to act as the whiteout. It’s contents are rather ugly so that if you compile or include the "removed" file accidentally, you get a fatal error. This will remind you to remove obsolete references.

When the change in integrated, the removed file is *not* copied/linked from the baseline to the integration directory, and is *not* copied from the development directory. At this time it is physically gone (no whiteout). It is assumed that because of the error inducing whiteout all old references were found and fixed while the change was in the *being developed* state.

File Manifests

When generating list of files to be compiled or linked, it is important that the file manifest be generated from information known by Aegis, rather than from the file system. This is for several reasons:

(a) Aegis knows exactly what (source) files are where, whereas everything else is inferring Aegis' knowledge; and

(b) looking in the file system is hard when the view path is longer that 2 directories (and Aegis' branching method can make it arbitrarily long); and

(c) The whiteout files, and anything else left "lying around", will confuse any method which interrogates the file system.

The easiest way to use Aegis' file knowledge is with something like an *awk*(1) script processing the Aegis file lists. For example, you can do this with *make*(1) as follows:

```
# generate the file manifest
manifest.make.inc: manifest.make.awk
    ( aegis −l cf −ter ; aegis −l pf −ter ) | \
    awk −f manifest.make.awk > manifest.make.inc
# now include the file manifest
include manifest.make.inc
```

Note: this would be inefficient of you did it once per directory, but there is nothing stopping you writing numerous assignments into the *manifest.make.inc* file, all in one pass.

It is possible to do the same thing with Aegis' report generator (see *aer*(1) for more information), but this is more involved than the *awk*(1) script. However, with the information "straight from the horse's mouth" as it were, it can also be much smarter.

This file manifest would become out-of-date without an interlock to Aegis' file operations commands. By using the *project-file_command* and *change_file_command* fields of the project *config* file (see *aepconf*(5) for more information), you can delete this file at strategic times.

```
/* run when the change file manifest is altered */
change_file_command = "rm −f manifest.make.inc";
/* run when the project file manifest is altered */
project_file_command = "rm −f manifest.make.inc";
```

The new file manifest will thus be re-built during the next *aeb*(1) command.

Options and Preferences

There is a **−No-WhiteOut** option, which may be used to suppress whiteout files when you use the *aerm*(1) and *aemv*(1) commands. There is a corresponding **−WhiteOut** option, which is usually the default.

There is a *whiteout_preference* field in the user preferences file (see *aeuconf*(5) for more information) if you want to set this option more permanently.

Whiteout File Templates

The *whiteout_template* field of the project *config* file may be used to produce language-specific error files. If no whiteout template entry matches, a very ugly 1KB file will be produced − it should induce compiler errors for just about any language.

If you want a more human-readable error message, entries such as

```
whiteout_template =
[
    {
        pattern = [ "*.[ch]" ];
        body = "#error This file has been removed.";
    }
];
```

can be very effective (this example assumes *gcc*(1) is being used).

If it is essential that *no* whiteout file be produced, say for C source files, you could use a whiteout template such as

```
whiteout_template =
[
    { pattern = [ "*.c" ]; }
];
```

because an absent *body* sub-field means generate no whiteout file at all.

You may have more than one whiteout template entry, but note that the order of the entries is important. The first entry which matches will be used.

Notification

On successful completion of this command, the notifications usually performed by the *aermu*(1), *aenfu*(1) and *aentu*(1) commands are run, as appropriate. These include the *project_file_command*, *new_file_undo_command*, *new_test_undo_command* and *remove_file_undo_command* fields of the project *config* file. See *aepconf*(5) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging

This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links

This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links

This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for

your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−WhiteOut
This option may be used to request that deleted files be replaced by a "whiteout" file in the development directory. The idea is that compiling such a file will result in a fatal error, in order that all references may be found. This is usually the default.

−No_WhiteOut
This option may be used to request that no "whiteout" file be placed in the development directory.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aemvu 'aegis −mvu \!* −v'
sh$ aemvu(){aegis −mvu "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is not being moved by the change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy files into a change

aedb(1) begin development of a change

aemv(1) rename files as part of a change

aenf(1) add files to be created by a change

aenfu(1) remove files to be created by a change

aerm(1) add files to be deleted by a change

aermu(1)
remove files to be deleted by a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new administrator − add a new administrator to a project

SYNOPSIS

aegis −New_Administrator *user-name*... [*option*...]
aegis −New_Administrator −List [*option*...]
aegis −New_Administrator −Help

DESCRIPTION

The *aegis −New_Administrator* command is used to add a new administrator to a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aena 'aegis −na \!* −v'
sh$     aena(){aegis −na "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.
It is an error if any of the named users have a uid of less than 100.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aepa(1) modify the attributes of a project

aera(1) remove administrators from a project

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new branch − create a new branch

SYNOPSIS

aegis −New_BRanch [*number*][*option*...]
aegis −New_BRanch −Help

DESCRIPTION

The *aegis −New_BRanch* command is used to create a new branch. A branch is very similar to a *change*, except that a branch may have changes (or branches) of its own, and a change may not.

You may choose your own branch number, if you want. Zero and positives are legal, but negatives are not. It is an error if that number has already been used for a change or another branch. If you do not specify a change number, the lowest available positive number (1 or more) will be used.

The new branch will be a special sort of change. It will be in the '*being developed*' state, but the usual commands in that stat (build, diff, etc) will not work. Instead, you must create changes on the branch, and when those changes are integrated into the branch, this is the equivalent of build, diff, etc, on the branch. Once the branch is completed, the *aede*(1) command may be used to advance it to the *being reviewed* state, and from then on it becomes a normal change. Should it be returned to the *being developed* state for any reason, it will once again require sub-changes to alter anything.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined branch number. Useful for writing scripts.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aenbr 'aegis −nbr \!* −v'
sh$     aenbr(){aegis −nbr "$@" −v}
```

SEE ALSO

aenbru(1)
: remove a branch

aenc(1)
: create a new change

aede(1)
: develop end

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new branch undo − remove a branch

SYNOPSIS

aegis −New_BRanch_Undo *number* [*option*...]
aegis −New_BRanch_Undo −Help

DESCRIPTION

The *aegis −New_BRanch_Undo* command is used to remove a branch created with the *aenbr*(1) command.

Note: This command will completely remove all trace of the branch from Aegis' database. This includes all changes performed on the branch and all of its sub-branches. (This history remains in the history files, but is inaccessible.)

If you wish to finish development of a branch, and commit all of its changes to the parent branch, use the *aede*(1) command, instead.

If you wish to stop anyone from developing more changes on the branch, use the *aerd*(1) command to remove all the developers.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenbr(1)
create a new branch

aencu(1)
new change undo

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new change − add a new change to a project

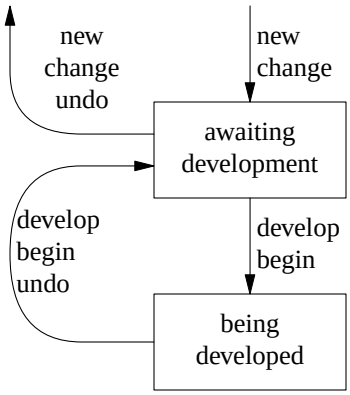
SYNOPSIS

aegis −New_Change [*number*] **−File** *attr-file* [*option*...]
aegis −New_Change [*number*] **−Edit** [*option*...]
aegis −New_Change −List [*option*...]
aegis −New_Change −Help

DESCRIPTION

The *aegis −New_Change* command is used to add a new change to a project. See *aecattr*(5) for information on the format of the *attr-file*.

The change is created in the *awaiting development* state. The change is not assigned to any user. The change has no development directory.



You may choose your own change number if you want, provided that it has not been used already. If you do not specify a change number, aegis will allocate the lowest unused change number. The first few change numbers are reserved for branches later in the project, and so automatically allocated change numbers will usually not start from 1. See *aepa*(1) and *aepattr*(5) for more information.

You **must** give the **−Project** option, see below.

Notification

This is one of the rare "state transitions" which does not have a notification command. The assumption is this command is invoked by the system which usually receives notifications.

OPTIONS

The following options are understood:

−Edit

Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround

Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_-command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*
Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined change number. Useful for writing scripts.

−Project *name*
This option is used to select the project for the new change.

You **must** supply the **−Project** option to this command. Experience has shown that when a site has a number of active projects or several active branches on a project, new changes are frequently created against the wrong project or the wrong branch. Making the project explicit reduces this problem.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "**−−***option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aenc 'aegis −nc \!* −v'
sh$ aenc(){aegis −nc "$@" −v}

ERRORS

It is an error if the current user is not an administrator of the project. (In some cases it is possible for developers of a project to create changes, see *aepattr*(5) for more information.)

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

tkaenc(1)
GUI interface to the *aenc*(1) command.

aeca(1) modify the attributes of a change

aedb(1) begin development of a change

aena(1) add a new administrator to a project

aencu(1)
remove a new change from a project

aenpr(1)
create a new project

aepa(1) modify the attributes of a project

aecattr(5)
change attributes file format

aepattr(5)
project attributes file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new change undo − remove a new change from a project

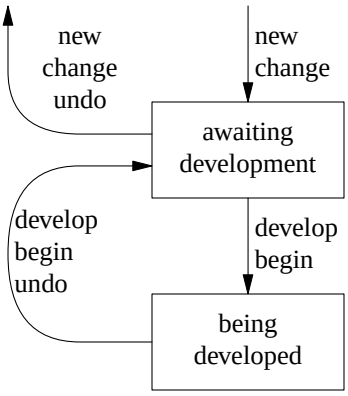
SYNOPSIS

aegis −New_Change_Undo [*option*...]
aegis −New_Change_Undo −List [*option*...]
aegis −New_Change_Undo −Help

DESCRIPTION

The *aegis −New_Change_Undo* command is used to remove a new change from a project.

It wasn't called *'aegis −Remove_Change'* in order to emphasize that fact the the change must be in the *awaiting development* state. In practice it is possible, with a combination of commands, to remove any change which has not reached the *completed*



state.

In general, only project administrators may destroy changes. However, if the project *developers_may_-create_changes* attribute is true, and you are a developer and you created a particular change, you may also destroy it.

Notification

This is one of the rare "state transitions" which does not have a notification command. The assumption is this command is invoked by the system which usually receives notifications. It's probably a bad assumption.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aencu 'aegis −ncu \!* −v'
sh$ aencu(){aegis −ncu "$@" −v}

ERRORS

It is an error if the change is not in the *awaiting development* state.
It is an error if any use other than a project administrator or the creator of the change attempts to run this command.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenc(1) add a new change to a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new developer − add new developers to a project

SYNOPSIS

aegis −New_Developer *user-name*... [*option*...]
aegis −New_Developer −List [*option*...]
aegis −New_Developer −Help

DESCRIPTION

The *aegis −New_Developer* command is used to add new developers to a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aend 'aegis −nd \!* −v'
sh$     aend(){aegis −nd "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerd(1) remove developers from a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new file − add new files to be created by a change

SYNOPSIS

aegis −New_File *file-name*... [*option*...]
aegis −New_File −List [*option*...]
aegis −New_File −Help

DESCRIPTION

The *aegis −New_File* command is used to add new files to a change. The named files will be added to the list of files in the change.

For each file named, a new file is created in the development directory, if it does not exist already. If the file already exists, it will not be altered.

If you want a new source file to be executable (shell scripts, for example) then you simply use the normal *chmod*(1) command. If any of the file's executable bits are set at *aede*(1) time the file is remembered as executable and all execute bits (minus the project's umask) will be set by subsequent *aecp*(1) commands.

If you name a directory on the command line, the entire directory tree will be searched for new files. (Note: absolutely everything will be added, including dot files and binary files, so you will need to clean out any junk first.) Files below this named directory which are already in the change, or in the project, will be ignored. The *file_name_accept* and *file_name_reject* patterns in the project *aegis.conf* file will also be applied, see *aepconf*(5) for more information.

Directory Example

There are times when a command such as

```
$ aenf fubar/*
aegis: project "example": change 42: "fubar/glorp" already in
change
aegis: project "example": change 42: found 1 fatal error, no new
files added
$
```

will fail as shown. There are several ways to deal with this, the easiest being to simply name the directory:

```
$ aenf fubar
aegis: project "example": change 42: file "fubar/smiley" added
aegis: project "example": change 42: file "fubar/frownie" added
$
```

You could also use the *find*(1) command for arbitrarily complex file selection, but you must first exclude files that the above command excludes automatically:

```
$ aelcf > exclude
$ aelpf >> exclude
$ find fubar −type f | \
    grep −v −f exclude | \
    xargs aegis −−new-file −v
aegis: project "example": change 42: file "fubar/smiley" added
aegis: project "example": change 42: file "fubar/frownie" added
$
```

If you aren't using the exclude list, the *find*(1) command will need fine tuning for your development directory style. If you are using the symlink-style, you will need to add the *find −nlink 1* option in addition to the *find −type f* option.

```
$ find fubar −type f −nlinks 1 | \
    xargs aegis −−new-file −v
aegis: project "example": change 42: file "fubar/smiley" added
aegis: project "example": change 42: file "fubar/frownie" added
$
```

If you are using the full-copy development directory style, you will have to use the exclude list method, above.

File Templates

When a new file is created in the development directory the project *config* file is searched for a template for the new file. If a template is found, the new file will be initialized to the template, otherwise it will be created empty. See *aepconf*(5) for more information.

The simplest form is to use template files, such as

```
file_template =
[
    {
        pattern = [ "*.c" ];
        body = "${read_file ${source template/c abs}}";
    },
    {
        pattern = [ "test/*/.sh" ];
        body = "${read_file ${source template/test abs}}";
    },
];
```

As you can see, the template files are part of the project source, so you can add the appropriate copyright notices, and wrappers, *etc*. The *$source* substitution locates them, if they are not part of the current change (and they usually are not).

The template files themselves contain substitutions. The *$filename* substitution is available, and contains the name of the file being created. This can be manipulated in various ways when constructing the appropriate file contents. See *aesub*(5) for more information about substitutions.

It is also possible to run a command to create the new file. You can do this instead of specifying a body string, *viz:*

```
file_template =
[
    {
        pattern = [ "*" ];
        body_command = "perl ${source template.pl abs} $filename";
    },
];
```

The command is run with a current directory set to the top of the development directory. It is an error if the command fails to create the file. You can mix-and-match the two techniques, *body* string and *body_command*, if you want.

File Name Limitations

There are a number of controls available to limit the form of project file names. All of these controls may be found in the project configuration file, see *aepconf* (5) for more information. The most significant are briefly described here:

maximum_filename_length = integer;

This field is used to limit the length of filenames. All new files may not have path components longer than this. Defaults to 255 if not set. For maximum portability you should set this to 14.

posix_filename_charset = boolean;

This field may be used to limit the characters allowed in filenames to only those explicitly allowed by POSIX. Defaults to *false* if not set, meaning whatever your operating system will tolerate, except white space and high-bit-on characters. For maximum portability you should set this to *true*.

dos_filename_required = boolean;

This field may be used to limit filenames so that they conform to the DOS 8+3 filename limits and to the DOS filename character set. Defaults to *false* if not set.

windows_filename_required = boolean;
: This field may be used to limit filenames so that they conform to the Windows98 and WindowsNT filename limits and character set. Defaults to *false* if not set.

shell_safe_filenames = boolean;
: This field may be used to limit filenames so that they do not contain shell special characters. Defaults to *true* if not set. If this field is set to *false*, you will need to use the *${quote}* substitution around filenames in commands, to ensure that filenames containing shell special characters do not have unintended side effects. Weird characters in filenames may also confuse your dependency maintenance tool.

allow_white_space_in_filenames = boolean;
: This field may be used to allow white space characters in file names. This will allow the following characters to appear in file names: backspace (BS, \b, 0x08), horizontal tab (HT, \t, 0x09), new line (NL, \n, 0x0A), vertical tab (VT, \v, 0x0B), form feed (FF, \f, 0x0C), and carriage return (CR, \r, 0x0D). Defaults to false if not set.

  Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as well.

  The user must take great care to use the ${quote} substitution around all file names in commands in the project configuration. And even then, substitutions which expect a space separated list of file names will have undefined results.

allow_non_ascii_filenames = boolean;
: This field may be used to allow file names with non-ascii-printable characters in them. Usually this would mean a UTF8 or international charset of some kind. Defaults to false if not set.

  Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as well.

filename_pattern_accept = [string];
: This field is used to specify a list of patterns of acceptable filenames. Defaults to "*" if not set.

filename_pattern_reject = [string];
: This field is used to specify a list of patterns of unacceptable filenames.

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory

path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Changing the Type of a File

If you want to change the type of a file (say, from a test to a source file, or *vice versa*) you could do it as two changes, by first using *aerm*(1) in one change and then using *aenf*(1) or *aent*(1) in a second change, or you can combine both steps in the same change. Remember to use the *aerm −nowhiteout* option or you will get a most peculiar new file template.

File Action Adjustment

When this command runs, it first checks the change files against the projects files. If there are inconsistencies, the file actions will be adjusted as follows:

create If a file is being created, but another change set is integrated which also creates the file, the file action in the change set still being developed will be adjusted to "modify".

modify If a file is being modified, but another change set is integrated which removes the file, the file action in the change set still being developed will be adjusted to "create".

remove If a file is being removed, but another change set is integrated which removes the file, the file will be dropped from the change set still being developed.

Notification

The *new_file_command* in the project configuration file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

TEST CORRELATIONS

The “aegis −Test −SUGgest” command may be used to have aegis suggest suitable regression tests for your change, based on the source files in your change. This automatically focuses testing effort to relevant tests, reducing the number of regression tests necessary to be confident that you have not introduced a bug.

The test correlations are generated by the “aegis −Integrate_Pass” command, which associates each test in the change with each source file in the change. Thus, each source file accumulates a list of tests which have been associated with it in the past. This is not as exact as code coverage analysis, but is a reasonable approximation in practice.

The *aecp*(1) and *aenf*(1) commands are used to associate files with a change. While they do not actively perform the association, these are the files used by *aeipass*(1) and *aet*(1) to determine which source files are associated with which tests.

Test Correlation Accuracy

Assuming that the testing correlations are accurate and that the tests are evenly distributed across the function space, there will be a less than *1/number* chance that a relevant test has not been run by the “aegis −Test −SUGgest *number*” command. A small amount of noise is added to the test weighting, so that unexpected things are sometimes tested, and the same tests are not run every time.

Test correlation accuracy can be improved by ensuring that:

- Each change should be strongly focused, with no gratuitous file inclusions. This avoids spurious correlations.
- Each item of new functionality should be added in an individual change, rather than several together. This strongly correlates tests with functionality.
- Each bug should be fixed in an individual change, rather than several together. This strongly correlates tests with functionality.

- Test correlations will be lost if files are moved. This is because correlations are by name.

The best way for tests to correlate accurately with source files is when a change contains a test and exactly those files relating to the functionality under test. Too many spurious files will weaken the usefulness of the testing correlations.

OPTIONS

The following options are understood

−as-needed
Usually it is an error if a file is already in a change set, and is redundantly added to the change set again. This option says to ignore such files.

−Build
This option may be used to specify that the file is constructed during a build (often only an integrate build), so that history of it may be kept. This is useful for generating patch files, where a history of generated files is important. Files created in this way may not be copied into a change, though they may be deleted. Avoid using files of this type, if at all possible.

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−CONFIGured
This option may be used to specify that the file is an Aegis project configuration file. The default project configuration file is called *aegis.conf*, however any file name may be used. You may also use more than one file, splitting the content across several files, all of which must be of this type.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−INDependent
The option may be used to request all the necessary actions, but not to actually add the new file to the change set.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−TEMplate
This option may be used to specify that a new file template should be used, even if the file already exists.

−No_TEMplate
This option may be used to specify that a new file template should not be used, even if the file does not exist (any empty file will be created).

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−Universal_Unique_IDentifier *string*
This option may be used to set the UUID of a file.

−Not_Universal_Unique_IDentifier
This option may be used to require that the file is created without an UUID. The **aeipass-option:assign-file-uuid** is set to false for the file to avoid automatic UUID assignment when **aeipass**(1) is invoked.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aenf 'aegis −nf \!* −v'
sh$ aenf(){aegis −nf "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is already part of the change.
It is an error if the file is already part of the baseline.
It is an error if the files named on the command line are not normal files and not directories. (If you need symbolic links or special files, create them at build time.)

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy files into a change

aedb(1) begin development of a change

aemv(1) rename a file as part of a change

aenfu(1) remove new files from a change

aent(1) add new tests to a change

aerm(1) add files to be deleted by a change

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new file undo − remove new files from a change

SYNOPSIS

aegis −New_File_Undo *file-name*... [*option*...]
aegis −New_File_Undo −List [*option*...]
aegis −New_File_Undo −Help

DESCRIPTION

The *aegis −New_File_Undo* command is used to remove new files from a change (reverse the actions of the 'aegis −New_File' command). The file is removed from the list of files in the change.

The file is removed from the development directory unless the **−Keep** option is used. The **−Keep** option should be used with great care, as you can confuse tools such as *make*(1) by leaving these files in place.

You may specify a directory name to remove all new files in the named directory tree, other files in the tree will be ignored. It is an error if there are no relevant files.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Notification

The *new_file_undo_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because deleting a file logically invalidates them.

The difference file (`,D`) will also be removed, however any DMT derived files (e.g a `.o` file from a `.c` file) will not be removed. This is because aegis is decoupled from the DMT, and cannot know what these derived file may be called. You may need to delete derived files manually.

OPTIONS

The following options are understood:

−BAse_RElative
: This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
: This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

-Interactive
Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links
This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see

aeuconf(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aenfu ’aegis −nfu \!$ −v’
sh$ aenfu(){aegis −nfu "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is not in the change.
It is an error if the file was not added to the change with the ’aegis −New_File’ command.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenf(1) add new files to a change

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new integrator − add new integrators to a project

SYNOPSIS

aegis −New_Integrator *user-name*... [*option*...]
aegis −New_Integrator −List [*option*...]
aegis −New_Integrator −Help

DESCRIPTION

The *aegis −New_Integrator* command is used to add new integrators to a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aeni 'aegis −ni \!* −v'
sh$     aeni(){aegis −ni "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeri(1) remove integrators from a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new project alias − create a new project alias

SYNOPSIS

aegis −New_Project_Alias [*option*...] *project-name alias-name*
aegis −Help
aegis −VERSion

DESCRIPTION

The *aegis −New_Project_Alias* command is used to create a projects alias, so that branches of projects may be referred by a shorter or more specific name.

The project name *must* be given on the command line; the default project is not sufficient. The project named may be a top-level project, or it may be a branch (to any depth of branch).

The new alias name must also be given on the command line, and it must be the *second* name. Project aliases have fewer limits than project names: they must not need shell quoting, that's all.

Example

Aliases may be used in may ways. The most common is to give a particular release a code name. You would do this by saying

```
aenpa example.4.2 sydney
```

This would make "sydney" an alias for the "example.4.2" branch.

Another use for aliases is to have a fixed alias for your active branch, so that your developer team does not need to change their default project, even though the branch number moves on for each release. You could say

```
aenpa example.4.2 example.cur
```

This would make "example.cur" an alias for the "example.4.2" branch. When this was finished, and 4.3 started, a project administrator could say

```
aerpa example.cur
aenpa example.4.3 example.cur
```

Now "example.cur" is an alias for the "example.4.3" branch, but the developers need only reference "example.cur" to always work on the right branch.

OPTIONS

The following options are understood:

-Help

This option may be used to obtain more information about how to use the *aegis* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

ERRORS

It is an error if the old project does not exist.
It is an error if the current user is not a project administrator.
It is an error if the new alias name look like a branch name.
It is an error if the new alias contains unprintable characters.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerpa(1) Remove project alias.

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new project − create a new project

SYNOPSIS

aegis −New_PRoject *project-name* [*option*...]
aegis −New_PRoject −List [*option*...]
aegis −New_PRoject −Help

DESCRIPTION

The *aegis −New_PRoject* command is used to create a new project. The project is created as an empty directory structure with no staff except the administrator, no changes, and branches to implement the version specified.

Please note: unless you specify a version (see the **−version** option, below) this command will default to creating branches to support version 1.0. If you discovered this too late, all is not lost: you can use the *aenbru*(1) command to get rid of the branches you didn't want.

Directory

The project directory, under which the project baseline and history and state and change data are kept, will be created at this time. If the **−DIRectory** option is not given, the project directory will be created in the directory specified by the default_project_directory field of *aeuconf*(5), or if not set in current user's home directory; in either case with the same name as the project.

Staff

The project is created with the current user and group as the owning user and group. The current user is an administrator for the project. The project has no developers, reviewers, integrators or other administrators. The project's umask is derived from the current user's umask, but guaranteeing that group members will have access and that only the project owner will have write access.

Pointer

The project pointer will be added to the first element of the search path, or */usr/local/com* if no path is set. If this is inappropriate, use the **−LIBrary** option to explicitly set the desired location. See the **−LIBrary** option for more information.

Version

You may specify the project version in two ways:

1. The version number may be implicit in the project name, in which case the version numbers will be stripped off. For example, "aenpr example.1.2" will create a project called "example" with branch number 1 created, and sub-branch 2 of branch 1 created.
2. The version number may be stated explicitly, in which case it will be subdivided for branch numbers. For example, "aenpr example −version 1.2" will create a project called "example" with branch number 1 created, and sub-branch 2 of branch 1 created.

In each case, these branches may be named wherever a project name may be given, such as "−p example.1" and "−p example-1.2". The actual punctuation character is unimportant.

You may have any depth of version numbers you like. Both methods of specifying version numbers may be used, and they will be combined. If you want no version numbers at all, use **−version** with a single dash as the argument, as in "`−version −`"

If no version number is given, either explicitly or implicitly, version 1.0 is used.

Project Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are

(using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

OPTIONS

The following options are understood:

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use 'pwd' to make an absolute path, it usually gives the wrong answer.

−Edit

Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround

Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*

Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

−Keep This option may be used to re-attach a project detached using *aermpr −keep* and possibly moved by the system administrator.

−LIBrary *abspath*

This option may be used to specify a directory to be searched for global state files and user state files. (See *aegstate*(5) and *aeustate*(5) for more information.) Several library options may be present on the command line, and are search in the order given. Appended to this explicit search path are the directories specified by the *AEGIS_PATH* environment variable (colon separated), and finally, */usr/local/lib/aegis* is always searched. All paths specified, either on the command line or in the *AEGIS_PATH* environment variable, must be absolute.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−VERSion *number*
This option may be used to specify the version number for the project. Version numbers are implemented as branches. Use a single dash (“−”) as the argument if you want no version branches created.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aenpr ’aegis −npr \!* −v’
sh$ aenpr(){aegis −npr "$@" −v}

ERRORS

It is an error if the project name already exists.
It is an error if the project directory already exists.
It is an error if the current user does not have sufficient permissions to create the directory specified with the **−DIRectory** option.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aena(1) add a new administrator to a project

aenbru(1)
Remove a new branch. This can often be useful if *aenpr*(1) created some default branches for you, and now you want to get rid of them.

aenc(1) add a new change to a project

aend(1) add a new developer to a project

aenrls(1)
create a new project from an existing project

aenrv(1) add a new reviewer to a project

aermpr(1)
remove project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new release − create a new project from an old-style project.

SYNOPSIS

aegis −New_ReLeaSe project-name [*new-project-name*][*option*...]
aegis −New_ReLeaSe −List [*option*...]
aegis −New_ReLeaSe −Help

DESCRIPTION

The *aegis −New_ReLeaSe* command is used to create a new project from an existing project. *It creates a new post-3.0 project from an old pre-3.0 project.*

Please Note: If your old-style project does not have a version number in the project name, you *must* supply a new project name, otherwise you will get an error. (If you want to re-use the old project name, you need to rename the old project, and then use *aenrls* to create a new new-style project with the old name. See the HOWTO for how to change a project's name.)

This command was essential before the introduction of branches into the Aegis model. It is more useful to create a new release of a project by ending development on the branch of the previous release and starting development of a new branch numbered for the desired release.

Once you have a new-style project, use the *aenbr*(1) command to create new branches on this project. This provides more efficient release management, and allows historical versions to be reproduced more simply.

If no *new-project-name* is specified, it will be derived from the project given as follows: any minor version dot suffix will be removed from the name, then any major version dot suffix will be removed from the name. A major version dot suffix will be appended, and then a minor version dot suffix will be appended. As an example, "foo.1.0" would become "foo.1.1" assuming the default minor version increment, and "foo" would become "foo.1.1" assuming the same minor version increment.

The entire project baseline will be copied. The project state will be as if change 1 had already been integrated, naming every file (in the old project) as a new file. The history files will reflect this. No build will be necessary; it is assumed that the old baseline was built successfully. Change numbers will commence at 2, as will build numbers. Test numbers will commence where the old project left off (because all the earlier test numbers were used by the old project).

The default is for the minor version number to be incremented. If the major version number is incremented or set, the minor version number will be set to zero if it is not explicitly given.

The pointer to the new project will be added to the first element of the search path, or */usr/local/com* if none is set. If this is inappropriate, use the **−LIBrary** option to explicitly set the desired location. See the **−LIBrary** option for more information.

The project directory, under which the project baseline and history and state and change data are kept, will be created at this time. If the **−DIRectory** option is not given, the project directory will be created in the directory specified by the default_project_directory field of the project user's *aeuconf*(5), or if not set in project user's home directory; in either case with the same name as the project.

All staff will be copied from the old project to the new project without change, as will all of the project attributes.

THE BASELINE LOCK

The baseline lock is used to ensure that the baseline remains in a consistent state for the duration of commands which need to read the contents of files in the baseline.

The commands which require the baseline to be consistent (these include the *aeb*(1), *aecp*(1) and *aed*(1) commands) take a baseline *read* lock. This is a non-exclusive lock, so the concurrent development of changes is not hindered.

The command which modifies the baseline, *aeipass*(1), takes a baseline *write* lock. This is an exclusive lock, forcing *aeipass*(1) to block until there are no active baseline read locks.

It is possible that one of the above development commands will block until an in-progress *aegis*

−*Integrate_PASS* completes. This is usually of short duration while the project history is updated. The delay is essential so that these commands receive a consistent view of the baseline. No other integration command will cause the above development commands to block.

When aegis' branch functionality is in use, a read (non-exclusive) lock is taken on the branch baseline and also each of the "parent" baselines. However, a baseline write (exclusive) lock is only taken on the branch baseline; the "parent" baselines are only read (non-exclusive) locked.

Project Directory Location

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

OPTIONS

The following options are understood:

−DIRectory *path*

This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

−LIBrary *abspath*

This option may be used to specify a directory to be searched for global state files and user state files. (See *aegstate*(5) and *aeustate*(5) for more information.) Several library options may be present on the command line, and are search in the order given. Appended to this explicit search path are the directories specified by the *AEGIS_PATH* environment variable (colon separated), and finally, */usr/local/lib/aegis* is always searched. All paths specified, either on the command line or in the *AEGIS_PATH* environment variable, must be absolute.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging

This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose

This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−VERSion *number*
This option may be used to specify the version number for the project. Version number are implemented as branches. Use the empty string as the argument if you want no version branches created.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aenrls 'aegis −nrls \!* −v'
sh$ aenrls(){aegis −nrls "$@" −v}

ERRORS

It is an error if the old project named does not exist.

It is an error if the old project named has not yet had any changes integrated.

It is an error if the old project named has any changes not in the *completed* state.

It is an error if the current user is not an administrator of the old project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenpr(1)
create a new project

aermpr(1)
remove project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

Aegis User Guide

The chapter on *Branching* has useful information about releases and branching.

NAME

aegis new reviewer − add new reviewers to a project

SYNOPSIS

aegis −New_ReViewer *user-name*... [*option*...]
aegis −New_ReViewer −List [*option*...]
aegis −New_ReViewer −Help

DESCRIPTION

The *aegis −New_ReViewer* command is used to add new reviewers to a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aenrv 'aegis −nrv \!* −v'
sh$     aenrv(){aegis −nrv "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerrv(1) remove reviewers from a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new test − add a new test to a change

SYNOPSIS

aegis −New_Test [*option*...][*filename*...]
aegis −New_Test −List [*option*...]
aegis −New_Test −Help

DESCRIPTION

The *aegis −New_Test* command is used to add a new test to a change. A new file is created in the development directory.

New tests default to “automatic” unless otherwise specified.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Test Filename Generation

You may choose your own filename for a test, by specifying it on the command line.

If no filename is specified on the command line, a test filename is automatically generated. This is controlled by the *new_test_filename* field of the project configuration file (see *aepconf*(5) for more information. All automatically generated test filenames within a project are numbered uniquely. The default pattern for new test filenames is "*test/XX/tXXXX[am].sh*", where *XX* is the first 2 digits of the test number, *XXXX* is the whole test number, and *[am]* is `a` for automatic tests and `m` for manual tests.

Modifying Tests

Tests may be modified in future by adding them to a change with the *aecp*(1) command. Tests are treated just like any other source file, and are subject to the same process.

File Templates

When a new file is created in the development directory the project *config* file is searched for a template for the new file. If a template is found, the new file will be initialized to the template, otherwise it will be created empty. See *aepconf*(5) for more information.

The simplest form is to use template files, such as

```
file_template =
[
    {
        pattern = [ "*.c" ];
        body = "${read_file ${source template/c abs}}";
    },
    {
        pattern = [ "test/*/.sh" ];
        body = "${read_file ${source template/test abs}}";
    },
];
```

As you can see, the template files are part of the project source, so you can add the appropriate copyright

notices, and wrappers, *etc*. The *$source* substitution locates them, if they are not part of the current change (and they usually are not).

The template files themselves contain substitutions. The *$filename* substitution is available, and contains the name of the file being created. This can be manipulated in various ways when constructing the appropriate file contents. See *aesub*(5) for more information about substitutions.

It is also possible to run a command to create the new file. You can do this instead of specifying a body string, *viz:*

```
file_template =
[
    {
        pattern = [ "*" ];
        body_command = "perl ${source template.pl abs} $filename";
    },
];
```

The command is run with a current directory set to the top of the development directory. It is an error if the command fails to create the file. You can mix-and-match the two techniques, *body* string and *body_command*, if you want.

Be careful to make sure that the test filename template pattern matches the *new_test_filename* field.

File Name Limitations

There are a number of controls available to limit the form of project file names. All of these controls may be found in the project configuration file, see *aepconf*(5) for more information. The most significant are briefly described here:

maximum_filename_length = integer;

This field is used to limit the length of filenames. All new files may not have path components longer than this. Defaults to 255 if not set. For maximum portability you should set this to 14.

posix_filename_charset = boolean;

This field may be used to limit the characters allowed in filenames to only those explicitly allowed by POSIX. Defaults to *false* if not set, meaning whatever your operating system will tolerate, except white space and high-bit-on characters. For maximum portability you should set this to *true*.

dos_filename_required = boolean;

This field may be used to limit filenames so that they conform to the DOS 8+3 filename limits and to the DOS filename character set. Defaults to *false* if not set.

windows_filename_required = boolean;

This field may be used to limit filenames so that they conform to the Windows98 and WindowsNT filename limits and character set. Defaults to *false* if not set.

shell_safe_filenames = boolean;

This field may be used to limit filenames so that they do not contain shell special characters. Defaults to *true* if not set. If this field is set to *false*, you will need to use the *${quote}* substitution around filenames in commands, to ensure that filenames containing shell special characters do not have unintended side effects. Weird characters in filenames may also confuse your dependency maintenance tool.

allow_white_space_in_filenames = boolean;

This field may be used to allow white space characters in file names. This will allow the following characters to appear in file names: backspace (BS, \b, 0x08), horizontal tab (HT, \t, 0x09), new line (NL, \n, 0x0A), vertical tab (VT, \v, 0x0B), form feed (FF, \f, 0x0C), and carriage return (CR, \r, 0x0D). Defaults to false if not set.

Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as

well.

The user must take great care to use the ${quote} substitution around all file names in commands in the project configuration. And even then, substitutions which expect a space separated list of file names will have undefined results.

allow_non_ascii_filenames = boolean;
: This field may be used to allow file names with non-ascii-printable characters in them. Usually this would mean a UTF8 or international charset of some kind. Defaults to false if not set.

  Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as well.

filename_pattern_accept = [string];
: This field is used to specify a list of patterns of acceptable filenames. Defaults to "*" if not set.

filename_pattern_reject = [string];
: This field is used to specify a list of patterns of unacceptable filenames.

Please Note: Aegis also consults the underlying file system, to determine its notion of maximum file size. Where the file system's maximum file size is less than *maximum_filename_length*, the filesystem wins. This can happen, for example, when you are using the Linux UMSDOS file system, or when you have an NFS mounted an ancient V7 filesystem. Setting *maximum_filename_length* to 255 in these cases does not alter the fact that the underlying file systems limits are far smaller (12 and 14, respectively).

If your development directories (or your whole project) is on filesystems with filename limitations, or a portion of the heterogeneous builds take place in such an environment, it helps to tell Aegis what they are (using the project *config* file's fields) so that you don't run into the situation where the project builds on the more permissive environments, but fails with mysterious errors in the more limited environments.

If your development directories are routinely on a Linux UMSDOS filesystem, you would probably be better off setting *dos_filename_required = true*, and also changing the *development_directory_template* field. Heterogeneous development with various Windows environments may also require this.

Changing the Type of a File

If you want to change the type of a file (say, from a test to a source file, or *vice versa*) you could do it as two changes, by first using *aerm*(1) in one change and then using *aenf*(1) or *aent*(1) in a second change, or you can combine both steps in the same change. Remember to use the *aerm −nowhiteout* option or you will get a most peculiar new file template.

Notification

The *new_test_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

TEST PROCESS

Each change is required to be accompanied by tests, and those tests are required to be run against the built development directory, and they must pass. This ensures that new functionality is accompanied by tests to verify its correctness, and bug fixes are accompanied by tests which confirm that the bug has been fixed.

Regression Tests

Tests are treated as any other source file, and are maintained in the baseline and history with all other source files. The tests which must accompany every change accumulate in the project baseline, providing a definition of correct function for the baseline. These accumulated tests may be executed using an "aegis −REGression" command, to verify that the project will not "regress" as a result of a change.

Baseline Tests

Bug fixes are required to have their tests *fail* against the project baseline (in contrast to the development directory). This ensures that the test actually demonstrates the bug in the baseline, as well as demonstrating that it is fixed by the change. New functionality trivially fails against the baseline, and so aegis does not attempt to guess if a test is a bug fix test or new functionality test, it simply requires tests to fail against the

baseline.

This requirement applies both to new tests being created by a change and also to tests which have been copied into a change for modification.

Reviewing Tests

Reviewers may be confident that aegis has enforced the test requirements; that a change must have tests, that the change must build, that the tests pass against the development directory, and that the tests fail against the baseline. These conditions are enforced by *aede*(1) and the change will not be advanced to the *being reviewed* state until these conditions are met. Reviewers should thus review tests for *completeness* of coverage of the code in the change, and insensitivity to changes in the execution environment (e.g. not date sensitive). Reviewers should also use "aegis −list change_details" to verify that a change does or does not have testing exemptions.

Exemptions

Various test exemptions may be granted by project administrators, see *aepa*(1) and *aepattr*(5) for more information. Copying tests into a change, or adding new tests to a change, may cancel those exemptions.

TEST CORRELATIONS

The "aegis −Test −SUGgest" command may be used to have aegis suggest suitable regression tests for your change, based on the source files in your change. This automatically focuses testing effort to relevant tests, reducing the number of regression tests necessary to be confident that you have not introduced a bug.

The test correlations are generated by the "aegis −Integrate_Pass" command, which associates each test in the change with each source file in the change. Thus, each source file accumulates a list of tests which have been associated with it in the past. This is not as exact as code coverage analysis, but is a reasonable approximation in practice.

The *aecp*(1) and *aenf*(1) commands are used to associate files with a change. While they do not actively perform the association, these are the files used by *aeipass*(1) and *aet*(1) to determine which source files are associated with which tests.

Test Correlation Accuracy

Assuming that the testing correlations are accurate and that the tests are evenly distributed across the function space, there will be a less than *1/number* chance that a relevant test has not been run by the "aegis −Test −SUGgest *number*" command. A small amount of noise is added to the test weighting, so that unexpected things are sometimes tested, and the same tests are not run every time.

Test correlation accuracy can be improved by ensuring that:

- Each change should be strongly focused, with no gratuitous file inclusions. This avoids spurious correlations.
- Each item of new functionality should be added in an individual change, rather than several together. This strongly correlates tests with functionality.
- Each bug should be fixed in an individual change, rather than several together. This strongly correlates tests with functionality.
- Test correlations will be lost if files are moved. This is because correlations are by name.

The best way for tests to correlate accurately with source files is when a change contains a test and exactly those files relating to the functionality under test. Too many spurious files will weaken the usefulness of the testing correlations.

OPTIONS

The following options are understood;

-AUTOmatic

This option may be used to specify automatic tests. Automatic tests require no human assistance.

−BAse_RElative

This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Edit Edit the new test files one they have been created. (This avoids the copy-and-paste step required to edit the new test script when it has an automatically generated file name.)

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-MANual
This option may be used to specify manual tests. Manual tests require some human intervention, e.g.: confirmation of some screen behavior (X11, for instance), or some user action, "unplug ethernet cable now".

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined test file name. Useful for writing scripts.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−TEMplate
This option may be used to specify that a new file template should be used, even if the file already exists.

−No_TEMplate
This option may be used to specify that a new file template should not be used, even if the file does not exist (any empty file will be created).

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

−Universal_Unique_IDentifier *string*
This option may be used to set the UUID of a file.

−Not_Universal_Unique_IDentifier
This option may be used to require that the file is created without an UUID. The **aeipass-option:assign-file-uuid** is set to false for the file to avoid automatic UUID assignment when **aeipass**(1) is invoked.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aent 'aegis −nt \!* −v'
sh$ aent(){aegis −nt "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy an existing test into a change

aedb(1) begin development of a change

aentu(1) remove a new test from a change

aerm(1) remove an existing test as part of a change

aet(1) run tests

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis new test undo − remove new tests from a change

SYNOPSIS

aegis −New_Test_Undo *file-name*... [*option*...]
aegis −New_Test_Undo −List [*option*...]
aegis −New_Test_Undo −Help

DESCRIPTION

The *aegis −New_Test_Undo* command is used to remove new tests from a change (reverse the actions of the 'aegis −New_Test' command). The file is removed from the development directory.

You may specify a directory name to remove all new tests in the named directory tree, other files in the tree will be ignored. It is an error if there are no relevant files.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Notification

The *new_test_undo_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because deleting a file logically invalidates them.

OPTIONS

The following options are understood:

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-Interactive
Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep
This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-No_Keep
This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf* (5) for more information.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf* (5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links
This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files' up-to-date-ness isn't important right now; incorrect use of this option may have unanticipated build side-effects. See also the "symbolic_links_preference" field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aentu ’aegis −ntu \!$ −v’
sh$ aentu(){aegis −ntu "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is not in the change.
It is an error if the file was not added to the change with the ’aegis −New_Test’ command.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aent(5) add a new test to a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis project attributes − modify the attributes of a project

SYNOPSIS

aegis −Project_Attributes *attr-file* [*option*...]
aegis −Project_Attributes −Edit [*option*...]
aegis −Project_Attributes −List [*option*...]
aegis −Project_Attributes −Help

DESCRIPTION

The *aegis −Project_Attributes* command is used to set, edit or list the attributes of a project.

The output of the **−List** variant is suitable for use as input at a later time.

See *aepattr*(5) for a description of the file format.

OPTIONS

The following options are understood:

−Edit

Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround

Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_-command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*

Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree

This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aepa 'aegis −pa \!* −v'
sh$ aepa(){aegis −pa "$@" −v}

ERRORS

It is an error if the current user is not an administrator of the specified project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

tkaepa(1)
Graphical interface to the *aepa*(1) command.

aeca(1) modify the attributes of a change

aepattr(5)
project attribute file format

aepstate(5)
project state file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aepatch − send and receive changes as patches

SYNOPSIS

aepatch −send [*option*...]
aepatch −receive [*option*...]
aepatch −list [*option*...]
aepatch −Help
aepatch −VERSion

DESCRIPTION

The *aepatch* command is used to send Aegis changes as patches, or receive patches and turn them into Aegis changes.

Please note that this only works for text files. If your project uses binary files, the *aepatch* program will not be useful because the *diff*(1) and *patch*(1) commands only work on text files. Also, this only works for files with names which do not contain white space.

If you need to merge matches together, you could use the GNU patch utils, which include a tool to merge patches together.

SEND

The send variant takes a specified change and constructs a patch containing all of the changes to all of the files in that change. The result is compressed, and encoded into a text format which can be sent as e-mail without being corrupted by the mail transfer agents along the way.

The output of the *aepatch −send* command is a normal Unix patch, as you would produce using *diff*(1), *bzip2*(1) and a MIME encoder such as *mpack*(1). There are no special formats. The output can be uncompressed with the normal *bunzip2*(1) command and applied with the normal *patch*(1) command.

The compression algorithm is selectable via the **−compression-algorithm** option, see the OPTIONS section, below, for details. The **−compatibility** option also understands compression needs.

Generating Traditional Patches

If you wish to send "traditional" patches to developers who are not using Aegis to manage the sources at their end, you can use the following options:

```
aepatch −send −cte=none −comp-alg=none
```

This says to use no Content Transfer Encoding, and no compression. If you wish to also omit the Aegis meta data, you can use the following options:

```
aepatch −send −cte=none −nocomp −compat=4.16
```

This setting for the −compatibility option omits all Aegis extensions.

By default, a context diff is generated. Some projects prefer to use the unified diff format. This is controlled by the *patch_diff_command* field of the project configuration file (see *aepconf*(5) for more information). If you have GNU diff, use the following command:

```
patch_diff_command = "set +e; "
  "diff −u −−text "
  "−L ${quote $index} −L ${quote $index} "
  "${quote $original} ${quote $input} > ${quote $output}; "
  "test $? −le 1"";
```

This setting will cause the *aepatch*(1) command to produce unified diff patches instead of context diff patches. As you can see from this command, the *aepatch*(1) command is onlu of use if you have text source files; it produces less than ideal results for binary files.

Options

The following options are understood by the send variant:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−COMPATibility *version-number*
This option may be used to specify the version of *aepatch*(1) which will be *receiving* this change set. This information is used to select which features to include in the data, and which to omit. By default, the latest feature set will be used.

−compression-algorithm *name*
This option may be used to specify the compression to be used. They are listed on order of compression effeciency.

none Use no compression (not always meaningful for all commands).

gzip Use the compression used by the *gzip*(1) program.

bzip2 Use the compression used by the *bzip2*(1) program.

More compression algorithms may be added in the future.

−COMPress
This option is deprecated in favour of the **−comp-alg=gzip** or **−comp-alg=bzip2** options.

−No_COMPress
This options is deprecated in favour of the **−comp-alg=none** option.

−Content_Transfer_Encoding *name*
This option may be used to specify the content transfer encoding to be used. It may take one of the following values:

None No content transfer encoding is to be performed.

Base64 The MIME base 64 encoding is to be used. This is the default.

Quoted_Printable
The MIME quoted printable encoding is to be used.

Unix_to_Unix_encode
The ancient unix-to-unix encoding is to be used.

These encodings may be abbreviated in the same way as comment line options.

−Ascii_Armor
This means the same as the “−cte=base64” option above.

−No_Ascii_Armor
This means the same as the “−cte=none” option above.

−DELta *number*
This option may be used to specify a particular delta in the project’s history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning “when the branch started”.

−DELta_Date *string*
This option may be used to specify a particular date and time in the project’s history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Signed_Off_By
This option may be used to have a `Signed-off-by:` line appended to the change set description.

−No_Signed_Off_By
This option may be used to prevent a `Signed-off-by:` line from being appended to the change set description.

RECEIVE

The receive variant takes a patch and creates an Aegis change (see *aenc*(1)) to implement the change within. Files are added to the change (see *aenf*(1), *aecp*(1), *aerm*(1), *aent*(1)) and then the patch contents are unpackaged into the development directory, and the changes applied to the files.

The patch does not have to be produced by the *aepatch*(1) command. Normal patches produced by *diff*(1) command are also valid input. The intent is that you can particicate in normal open source development, and also use Aegis, even if your fellow developers are not.

Once unpacked, the change is then built (see *aeb*(1)), differenced (see *aed*(1)), and tested (see *aet*(1)). The automatic process stops at this point, so that you can confirm that the change is desired.

File Names

It is common for patch files generated using the usual *diff −r* mechanism to contain extra path prefixes. The *aepatch*(1) command attempts to remove these automagically. This is usually possible because patches usually modify files within the project, so the patch file names are compared with project file names to guess which and how much path prefixes to remove.

−Remove_Path_Prefix *string*
This option may be used to explicitly specify path prefixes to be removed, if present. It may be specified more than once.

If you have a complex project directory structure, from time to time people may send you patches relative to a sub-directory, rather than relative to the project root. The *aepatch*(1) program can't guess this by itself.

−Add_Path_Prefix *string*
This option may be used to specify the path of a project sub-directory in which to apply the patch.

Notification

The *aepatch* command invokes various other Aegis commands. The usual notifications that these commands would issue are issued.

Options

The following options are understood by the receive variant:

−Change *number*
This option may be used to choose the change number to be used, otherwise the change number in the patch (if present) will be used if it is available, otherwise one will be chosen automatically.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, just as for the *aecp*(1) command. You may also use a delta name instead of a delta number.

−DIRectory *path*
This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename ‘−’ is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Project *name*
This option may be used to set the project name. If not specified the project name in the input package will be used (if present), otherwise the usual project name default will be used.

−Trojan This option may be used to treat the change set as if it had a Trojan horse attack in it.

−No_Trojan
This option may be used to treat the change set as if it definitely does not have a Trojan horse attack in it. *Use with extreme care.* You need to have authenticated the message with something like PGP first **and** know the the author well.

−Output *filename*
This option may be used to specify a filename which is to be written with the automatically determined change number. Useful for writing scripts.

Security
Receiving changes by e-mail, and automatically committing them to the baseline without checking them, would be a recipe for disaster. A number of safeguards are provided:

- The format of the package is confirmed to be correct, and the package verified for internal consistency, before it is unpacked and acted upon.
- The automatic portion of the process stops before development ends. This ensures that the receiver validates the change before it is committed, and then it must also be reviewed, preventing accidental or malicious damage.
- The more you use Aegis’ test management facilities (see *aent*(1) and *aet*(1)) the harder it is for an inadequate change to get into the baseline.

LIST

The list variant can be used to list the contents of a package without actually unpacking it first. The output is reminiscent of the *aegis −list change-details* output.

Options
The following options are understood by the list variant:

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename ‘−’ is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default. Only useful with the −List option.

OPTIONS

The following options to this command haven’t been mentioned yet:

-Help

This option may be used to obtain more information about how to use the *aepatch* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aepatch* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

FILE FORMAT

The file format re-uses existing formats, rather than introduce anything new. This means it is possible to extract the contents of a package even when aepatch is unavailable.

- On sending, the source files are generated using the *diff*(1) program, in the same way a normal Unix patch is generated.
  On receiving, the differences are applied to the source files, in the same manner as the normal *patch*(1) program.
- On sending, the patch is compressed using the GNU gzip format. Typically primary source files are ASCII text, resulting in significant compression. (This is optional.)
  On receiving, if the patch is compressed it will be automagically uncompressed, detection is automatic, you do not need to do this yourself.
- On sending, the compressed patch is encoded using the MIME base64 encoding. This makes the result approximately 33% larger than the compressed binary would be, but still smaller than the primary sources. (This is optional.)
  On receiving, if the patch is MIME64 encoded it will be automatically decoded, detetcion is automatic, you do not need to do this yourself.

EXIT STATUS

The *aepatch* command will exit with a status of 1 on any error. The *aepatch* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aepatch version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aepromptcmd − change prompt color by change state

SYNOPSIS

PROMPT_COMMAND="aepromptcmd"

DESCRIPTION

The *bash*(1) shell has an interesting property: If the PROMPT_COMMAND vaiable is set, the value is executed as a command prior to issuing each primary prompt. (Actually, it can be a seties of semicolon separated commands.)

In order to change the text back to normal, the PS1 variable needs to have "\33[0m" somewhere near the end, otherwise things can get a little difficult to read. If you are using *bash*(1), you need to let it know these are unprintable (like this: "\[\33[0m\]") or it messes up command line editing.

The *aepromptcmd* command is used to set the color of the prompt, based on the state of the current change. This is an idea taken from Kent Beck's *Test Driven Development* book. If the change is in the *being developed* or *being integrated* state and it needs to be built, the prompt is red; if it is built but it needs to be tested, the prompt is magenta, otherwise it is green.

Example

Here is a short script you can put in your `.bashrc` file to turn on prompt coloring:

```
if [ "$PS1" ] then
  case "$PROMPT_COMMAND" in
  "" )
    PROMPT_COMMAND="aepromptcmd"
    PS1="$PS1^[[0m"
    ;;
  *aepromptcmd*)
    ;;
  *)
    PROMPT_COMMAND="$PROMPT_COMMAND;aepromptcmd"
    PS1="$PS1\[\33[0m\]"
    ;;
  esac
  export PROMPT_COMMAND
  export PS1 fi
```

Note that this usually leaves your prompt default (black) when you are not somewhere inside a development directory.

Limitations

The *aepromptcmd* command uses the ANSI color escape sequences. It really should to use the *tigetstr*(3) function from *terminfo*(3) to do this in a terminal independent way. Code contributions welcome.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-Help

This option may be used to obtain more information about how to use the *aepromptcmd* program.

−Verbose

By default error messages are supressed, so that the prompt will be normal when you are outside an Aegis work area. Use this option to tuen error messages back on.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aepromptcmd* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aepromptcmd* command will exit with a status of 1 on any error. The *aepromptcmd* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aepromptcmd version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis report − report generator

SYNOPSIS

aereport [*option*...] *report-name*
aereport [*option*...] **−File** *filename*
aereport −RePorT −List
aereport −Help
aereport −VERSion

DESCRIPTION

The *aereport* command is used to generate reports from aereport' database. Reports are specified in a C-like language described in the *aer*(5) manual entry.

For a list of the reports available on your system, use the '*aer −list*' command. These reports live in the */usr/local/share/report* directory, and it initially contains the reports distributed with aereport, however sites are free to add their own here.

WRITING REPORT SCRIPTS

Getting started writing report scripts can be difficult. You are best to have a look at the reports distributes with Aegis, and try to adapt them. The report script files are kept in the */usr/local/share/report* directory.

For information about the data structures which may be accessed from a report script, you need to see the relevant manual entries:

the projects list
: See *aegstate*(1) for the member fields.

a specific project
: See *aepstate*(1) for the member fields.

a specific change
: See *aecstate*(1) for the member fields.

a specific file
: See *aefstate*(1) for the member fields.

Each of the above man pages also contains a section towards the end which specifically addresses report generator use, usually with code fragments.

OPTIONS

The following options are understood:

-BaseLine
: This option may be used to specify that the project baseline is the subject of the command.

−BRanch *number*
: This option may be used to specify a different branch for the origin file, rather than the baseline. (See also **−TRunk** option. Please Note: the **−BRanch** option does not take a project name, just the branch number suffix.

−GrandParent
: This option may be used to specify the grandparent branch (one up from the current branch) for the origin file, rather than the baseline. (The −grandparent option is the same as the "−branch .." option.)

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−DELta *number*
: This option may be used to specify a particular delta in the project's history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not

exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning “when the branch started”.

−DELta_Date *string*
This option may be used to specify a particular date and time in the project’s history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−File *filename*
Take the report script from the specified file, rather than looking for the named report in the library of reports distributed with Aegis. The filename “−” is understood to mean the standard input.

-Help
This option may be used to obtain more information about how to use the *aereport* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user’s *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-TRunk
This option may be used to specify the project trunk for the origin file, rather than the baseline. (See also **−BRanch** option, the −trunk option is the same as the “−branch −” option.)

-UNFormatted
This option may be used with most listings to specify that the column formatting is not to be performed. This is useful for shell scripts.

-Page-Header
This option requests that page headings be present in listings and reports. This is the default.

-No-Page-Header
This option requests that page headings be omitted from listings and reports.

-Verbose
This option may be used to cause aereport to produce more output. By default aereport only produces output on errors. When used with the **-List** option this option causes column headings to be added.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option.

The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aereport* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aer ’aereport \!* −v’
sh$ aer(){aereport "$@" −v}

SEE ALSO

ael(1) list (possibly) interesting things

aer(5) report script language definition

EXIT STATUS

The *aereport* command will exit with a status of 1 on any error. The *aereport* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aereport version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove administrator − remove administrators from a project

SYNOPSIS

aegis −Remove_Administrator *user-name ...* [*option...*]
aegis −Remove_Administrator −List [*option...*]
aegis −Remove_Administrator −Help

DESCRIPTION

The *aegis −Remove_Administrator* command is used to remove administrators from a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aera 'aegis −ra \!* −v'
sh$     aera(){aegis −ra "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

It is an error if an attempt is made to remove the last administrator from the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aena(1) add new administrators to a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis review begin − begin a change review

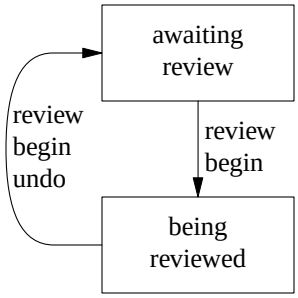
SYNOPSIS

aegis −Review_Begin [*option*...]
aegis −Review_Begin −List [*option*...]
aegis −Review_Begin −Help

DESCRIPTION

The *aegis −Review_Begin* command is used to notify aegis that you have begun to review a change.

The change will be advanced from the *awaiting review* state to the *being reviewed* state.



Notification

If the *review_begin_notify_command* has been set in the project attributes, this command will be run. This is usually used to tell other reviewers that you have started review, and they need not. See *aepattr*(5) and *aepa*(1) for more information.

If used when the *develop_end_action* project attribute is set to *goto_being_reviewed*, then only the notification message is sent.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aerb 'aegis −rb \!* −v'
sh$ aerb(){aegis −rb "$@" −v}

ERRORS

It is an error if the change is not in the *awaiting review* state.
It is an error if the current user is not a reviewer of the project.
Its is an error if the current user developed the change and the project is configured to not permit developers to review their own changes (default).

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecd(1) change directory

aede(1) complete development of a change

aedeu(1)
recall a change for further development

aerpass(1)
pass review of a change

aeib(1) begin integrating a change

aenrv(1) add a reviewer to a project

aerfail(1)
fail review of a change

aerpu(1)
rescind a change review pass

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

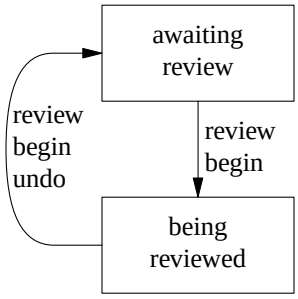
aegis review begin undo − stop reviewing a change

SYNOPSIS

aegis −Review_Begin_Undo [*option*...]
aegis −Help
aegis −VERSion

DESCRIPTION

The *aegis −Review_Begin_Undo* command is used to stop reviewing a change. It is moved from the *being reviewed* state back to the *awaiting review* state.



Notification

If the *review_begin_undo_notify_command* has been set in the project attributes, this command will be run. This is usually used to tell other reviewers that you have stopped reviewing, and they may like to do so instead. See *aepattr*(5) and *aepa*(1) for more information.

If used when the *develop_end_action* project attribute is set to *goto_being_reviewed*, then only the notification message is sent.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aerbu 'aegis −rbu \!* −v'
sh$ aerbu(){aegis −rbu "$@" −v}

ERRORS

It is an error if the change is not in the *being reviewed* state.
It is an error if the current user is not the reviewer of the change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecd(1) change directory

aede(1) complete development of a change

aedeu(1)
recall a change for further development

aerb(1) begin review of a change

aenrv(1) add a reviewer to a project

aerfail(1)
fail review of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove developer − remove developers from a project

SYNOPSIS

aegis −Remove_Developer *user-name*... [*option*...]
aegis −Remove_Developer −List [*option*...]
aegis −Remove_Developer −Help

DESCRIPTION

The *aegis −Remove_Developer* command is used to remove developers from a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aerd 'aegis −rd \!* −v'
sh$     aerd(){aegis −rd "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aend(1) add a new developer to a project

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aerect − draw a rectangle

SYNOPSIS

aerect [*option*...] *width height*
aegis -Help
aegis -VERSion

DESCRIPTION

The *aerect* command is used to draw rectangles for use with the intranet interface.

OPTIONS

The following options are understood:

−Bevel *size*
This option may be used to specify the bevel size. A size of 0 may be use to specify no bevel. Defaults to 3 if not specified.

−Color *red green blue*
This option may be used to specify the color of the rectangle. The components are specified in a range from 0 to 255. If not specified, the color will be based on the size of the rectangle.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “−−*option*=*value*” convention is also understood.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aerevml − send and receive RevML change sets

SYNOPSIS

aerevml −Send [*option*...]
aerevml −Receive [*option*...]
aerevml −Help
aerevml −VERSion

DESCRIPTION

The *aerevml* command is used to send and receive change sets using the RevML format. This format is independent of any particular VC/SCM tool or vendor. It allows export from any RevML capable VC/SCM system and import into any other RevML capable VC/SCM system.

The basic function is to reproduce a change, so a command like

```
aerevml −send | aerevml −receive
```

may be used to clone a change, though less efficiently than *aeclone*(1). The file format used is designed to withstand mail servers, so activities such as

```
aerevml −send | e-mail | aerevml −receive
```

(where *e-mail* represents sending, transporting and receiving your e-mail) will reproduce the change on a remote system. With suitable tools (such as PGP) is it possible to

```
aerevml −send | encrypt | e-mail | decrypt | aerevml −receive
```

The mechanism is also designed to allow web-based distribution such as

```
aerevml −send | web-server → web-browser | aerevml −receive
```

by the use of appropriate CGI scripts and mailcap entries.

It is possible to support both a “push” model and a “pull” model using this command. For suggestions and ideas for various ways to do this, see the Aegis Users Guide.

RevML Project

The RevML format is used for copying revision controlled files and change sets between various SCM repositories. The RevML project may be found at `http://public.perforce.com/public/-revml/index.html`

The latest RevML DTD may be found at `http://public.perforce.com/public/revml/-revml.dtd`

SEND

The send variant takes a specified change, or baseline, and constructs a distribution package containing all of the change attributes and source file attributes and source file contents. The result is compressed, and encoded into a text format which can be sent as e-mail without being corrupted by the mail transfer agents along the way.

Options

The following options are understood by the send variant:

−BaseLine
: This option may be used to specify the source of a project, rather than a change. Implies the *−Entire_Source* option, unless over-ridden.

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−COMPATibility *version-number*
: This option may be used to specify the version of *aerevml*(1) which will be *receiving* this change set. This information is used to select which features to include in the data, and which to omit. By default, the latest feature set will be used.

−compression-algorithm *name*
: This option may be used to specify the compression to be used. They are listed on order of compression effeciency.

none Use no compression (not always meaningful for all commands).

gzip Use the compression used by the *gzip*(1) program.

bzip2 Use the compression used by the *bzip2*(1) program.

More compression algorithms may be added in the future.

−COMPress
This option is deprecated in favour of the **−comp-alg=gzip** or **−comp-alg=bzip2** options.

−No_COMPress
This options is deprecated in favour of the **−comp-alg=none** option.

−Content_Transfer_Encoding *name*
This option may be used to specify the content transfer encoding to be used. It may take one of the following values:

None No content transfer encoding is to be performed.

Base64 The MIME base 64 encoding is to be used. This is the default.

Quoted_Printable
The MIME quoted printable encoding is to be used.

Unix_to_Unix_encode
The ancient unix-to-unix encoding is to be used.

These encodings may be abbreviated in the same way as comment line options.

−Ascii_Armor
This means the same as the "−cte=base64" option above.

−No_Ascii_Armor
This means the same as the "−cte=none" option above.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning "when the branch started".

−DELta_Date *string*
This option may be used to specify a particular date and time in the project's history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−Description_Header
This option may be used to add an RFC 822 style header to the change description being sent, with a From and Date line. This is the default.

−No_Description_Header
This option suppresses the description header.

−Entire_Source
This option may be used to send the entire source of the project, as well as the change source files.

−Mime_Headers
This option may be use to force the presence of mime headers in the output, in circumstances they would usually be absent.

−No_Mime_Headers
This option may be use to force the absence of mime headers in the output, in circumstances where they would usually be present.

−Partial_Source
This option may be used to send only source files of a change. This is the default, except for the *−BaseLine* option.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Signed_Off_By
This option may be used to have a `Signed-off-by:` line appended to the change set description.

−No_Signed_Off_By
This option may be used to prevent a `Signed-off-by:` line from being appended to the change set description.

RECEIVE

The receive variant takes a change package created by the send variant and creates an Aegis change (see *aenc*(1)) to implement the change within. Files are added to the change (see *aerm*(1), *aecp*(1), *aenf*(1) and *aent*(1)) and then the file contents are unpackaged into the development directory.

The change is then built (see *aeb*(1)), differenced (see *aed*(1)), and tested (see *aet*(1)). If all of this is successful, development of the change is ended (see *aed*(1)). The automatic process stops at this point, so that a local reviewer can confirm that the change is desired.

Notification

The *aerevml* command invokes various other Aegis commands. The usual notifications that these commands would issue are issued.

Options

The following options are understood by the receive variant:

−Change *number*
This option may be used to choose the change number to be used, otherwise one will be chosen automatically.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, just as for the *aecp*(1) command. You may also use a delta name instead of a delta number.

−DIRectory *path*
This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename ‘−’ is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Ignore_UUID
This option may be used to ignore the UUID, if present, of the incoming change set.

−No_Ignore_UUID
This option force the *aerevml* command to use the change set's UUID. This is the default.

−Project *name*
This option may be used to set the project name. If not specified, the project name in the input package will be used, rather than the usual project name defaulting mechanism.

−Trojan This option may be used to treat the change set as if it had a Trojan horse attack in it.

−No_Trojan
This option may be used to treat the change set as if it definitely does not have a Trojan horse attack in it. *Use with extreme care.* You need to have authenticated the message with something like PGP first **and** know the the author well.

Security

Receiving changes by e-mail, and automatically committing them to the baseline without checking them, would be a recipe for disaster. A number of safeguards are provided:

- The format of the package is confirmed to be correct, and the package verified for internal consistency, before it is unpacked and acted upon.
- The automatic portion of the process stops when development ends. This ensures that a local reviewer validates the change before it is committed, preventing accidental or malicious damage.
- If the change seeks to update the project *config* file, the automatic process terminates before the build or difference occurs. This is because this file could contain trojans for these operations, so a human must examine the file before the change proceeds any further.
- There is a *potential_trojan_horse = [string];* field in the project*config* file. Nominate build configuration files, shell scripts, code generators, *etc* here to specify files in addition to the project configuration file which should cause the automatic processing to halt.
- The use of e-mail authentication and encryption systems, such as PGP and GPG, are encouraged. However, it is expected that this processing will occur after *aerevml −send* has constructed the package and before *aerevml −receive* examines and acts on the package. Verification of the sender is the surest defense against trojan horses.
- Automatic sending and receiving of packages is supported, but not implemented within the aerevml command. It is expected that the aerevml command will be used within shell scripts customized for your site and its unique security requirements. See the Aegis User Guide for several different ways to do this.
- The more you use Aegis' test management facilities (see *aent*(1) and *aet*(1)) the harder it is for an inadequate change to get into the baseline.

Duplicate Storms

In a distributed development environment, it is common for change sets to eventually be propagated back to the originator. There are situations (particularly in some star topologies) where several copies of the package will return to the originator.

If these change sets are not detected at the review stage, and are propagated out yet again, there is the possibility of an exponential explosion of redundant change sets being distributed again and again.

To combat this, changes are checked after the files are unpacked, but before and build or difference or test is performed. The "*aecpu −unchanged*" command is used to exclude all files that the local repository already has in the desired form. If no change files remain after this, the change is dropped entirely (see *aedbu*(1) and *aencu*(1)).

LIST

The list variant can be used to list the contents of a package without actually unpacking it first. The output is reminiscent of the *aegis −list change-details* output.

Options

The following options are understood by the list variant:

−File *filename*

Read the change set from the specified file. The default is to read it from the standard input. The filename '−' is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Output *filename*

This option may be used to specify the output file. The output is sent to the standard output by default. Only useful with the −List option.

OPTIONS

The following options to this command haven't been mentioned yet:

-Help

This option may be used to obtain more information about how to use the *aerevml* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aerevml* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

EXIT STATUS

The *aerevml* command will exit with a status of 1 on any error. The *aerevml* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis review fail − fail a change review

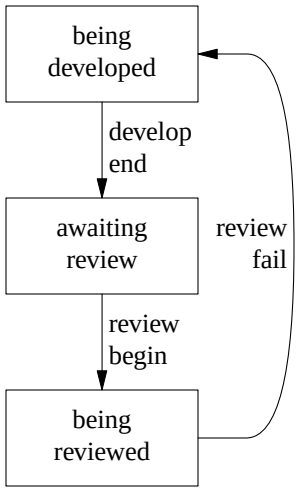
SYNOPSIS

aegis −Review_FAIL −File *reason-file* [*option*...]
aegis −Review_FAIL −REAson *'reason-text'* [*option*...]
aegis −Review_FAIL −Edit [*option*...]
aegis −Review_FAIL −List [*option*...]
aegis −Review_FAIL −Help

DESCRIPTION

The *aegis −Review_FAIL* command is used to inform aegis that a change has failed review.

The change will be returned from the *being reviewed* state to the *being developed* state. The change will cease to be assigned to the current user, and will be reassigned to the originating developer.



The developer will be notified by mail. See the review_fail_notify_command in *aepattr*(5) for more information.

The *reason-file* will contain a description of why the change was failed. The file is in plain text. It is recommended that you only use newline to terminate paragraphs, (rather than to terminate lines) as this will result in better formatting in the various listings.

Notification

On successful completion of this command, the *review_fail_notify_command* field of the project attributes is run, if set. See *aepattr*(5) and *aepa*(1) for more information.

OPTIONS

The following options are understood:

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Edit

Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround
Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*
Take the attributes from the specified file. The filename '−' is understood to mean the standard input.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to provide the failure reason on the command line, rather than in a file. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aerfail 'aegis −rfail \!* −v'
sh$     aerfail(){aegis −rfail "$@" −v}
```

ERRORS

It is an error if the change is not in the *being reviewed* state.

It is an error if the current user is not a reviewer for the project.

It is an error if the current user developed the change and the project is configured to disallow developers to review their own changes (default).

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecd(1) change directory

aede(1) complete development of a change

aedeu(1)
recall a change for further development

aenrv(1) add a reviewer to a project

aerpass(1)
pass review of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove integrator − remove integrators from a project

SYNOPSIS

aegis −Remove_Integrator *user-name*... [*option*...]
aegis −Remove_Integrator −List [*option*...]
aegis −Remove_Integrator −Help

DESCRIPTION

The *aegis −Remove_Integrator* command is used to remove integrators from a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aeri 'aegis −ri \!* −v'
sh$     aeri(){aegis −ri "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeni(1) add a new administrator to a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove file − add files to be deleted to a change

SYNOPSIS

aegis −ReMove_file *file-name*... [*option*...]
aegis −ReMove_file −List [*option*...]
aegis −ReMove_file −Help

DESCRIPTION

The *aegis −ReMove_file* command is used to add files to be deleted to a change. The file will be added to the list of files in the change, and will be removed from the baseline at integration time.

This command may be used to remove tests, not just source files. Tests are treated just like any other source file, and are subject to the same process.

A file will be created in the development directory containing 1KB of random text. The random text is sufficiently revolting that most compilers will give error messages, should the file be referenced accidentally. This is often very helpful when removing include files.

You may specify a directory name to remove all files in the named directory tree. It is an error if there are no relevant files.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because adding a file logically invalidates them.

When the change files are listed (*aegis −List Change_Files −TERse*) the removed files will not appear in the terse listing. Similarly, when the project files are listed with an explicit change number (*aegis −List Project_Files −TERse −Change* `N`) none of the change's files, including the the removed files, will not appear in the terse listing. These two features are very helpful when calling aegis from within a DMT to generate the list of source files.

Changing the Type of a File

If you want to change the type of a file (say, from a test to a source file, or *vice versa*) you could do it as two changes, by first using *aerm*(1) in one change and then using *aenf*(1) or *aent*(1) in a second change, or you can combine both steps in the same change. Remember to use the *aerm −nowhiteout* option or you will get a most peculiar new file template.

Notification

The *remove_file_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

WHITEOUT

Aegis provides you with what is often called a "view path" which indicates to development tools (compilers, build systems, *etc*) look first in the development directory, then in the branch baseline, and so on up to the trunk baseline.

The problem with view paths is that in order to remove files, you need some kind of "whiteout" to say "stop

looking, it’s been removed.”

When you user the *aerm*(1) or *aemv*(1) commands, this means "add information to this change which will remove the file from the baseline when this change is integrated". *I.e.* while the change is in the *being developed* state, the file is only "removed" in the development directory − it’s still present in the baseline, and will be until the change is successfully integrated.

When you use the *aerm*(1) or *aemv*(1) commands, Aegis will create a 1K file to act as the whiteout. It’s contents are rather ugly so that if you compile or include the "removed" file accidentally, you get a fatal error. This will remind you to remove obsolete references.

When the change in integrated, the removed file is *not* copied/linked from the baseline to the integration directory, and is *not* copied from the development directory. At this time it is physically gone (no whiteout). It is assumed that because of the error inducing whiteout all old references were found and fixed while the change was in the *being developed* state.

File Manifests

When generating list of files to be compiled or linked, it is important that the file manifest be generated from information known by Aegis, rather than from the file system. This is for several reasons:

(a) Aegis knows exactly what (source) files are where, whereas everything else is inferring Aegis’ knowledge; and

(b) looking in the file system is hard when the view path is longer that 2 directories (and Aegis’ branching method can make it arbitrarily long); and

(c) The whiteout files, and anything else left “lying around”, will confuse any method which interrogates the file system.

The easiest way to use Aegis’ file knowledge is with something like an *awk*(1) script processing the Aegis file lists. For example, you can do this with *make*(1) as follows:

```
# generate the file manifest
manifest.make.inc: manifest.make.awk
    ( aegis −l cf −ter ; aegis −l pf −ter ) | \
    awk −f manifest.make.awk > manifest.make.inc
# now include the file manifest
include manifest.make.inc
```

Note: this would be inefficient of you did it once per directory, but there is nothing stopping you writing numerous assignments into the *manifest.make.inc* file, all in one pass.

It is possible to do the same thing with Aegis’ report generator (see *aer*(1) for more information), but this is more involved than the *awk*(1) script. However, with the information "straight from the horse’s mouth" as it were, it can also be much smarter.

This file manifest would become out-of-date without an interlock to Aegis’ file operations commands. By using the *project-file_command* and *change_file_command* fields of the project *config* file (see *aepconf*(5) for more information), you can delete this file at strategic times.

```
/* run when the change file manifest is altered */
change_file_command = "rm −f manifest.make.inc";
/* run when the project file manifest is altered */
project_file_command = "rm −f manifest.make.inc";
```

The new file manifest will thus be re-built during the next *aeb*(1) command.

Options and Preferences

There is a **−No-WhiteOut** option, which may be used to suppress whiteout files when you use the *aerm*(1) and *aemv*(1) commands. There is a corresponding **−WhiteOut** option, which is usually the default.

There is a *whiteout_preference* field in the user preferences file (see *aeuconf*(5) for more information) if you want to set this option more permanently.

Whiteout File Templates

The *whiteout_template* field of the project *config* file may be used to produce language-specific error files. If no whiteout template entry matches, a very ugly 1KB file will be produced − it should induce compiler errors for just about any language.

If you want a more human-readable error message, entries such as

```
whiteout_template =
[
    {
        pattern = [ "*.[ch]" ];
        body = "#error This file has been removed.";
    }
];
```

can be very effective (this example assumes *gcc*(1) is being used).

If it is essential that *no* whiteout file be produced, say for C source files, you could use a whiteout template such as

```
whiteout_template =
[
    { pattern = [ "*.c" ]; }
];
```

because an absent *body* sub-field means generate no whiteout file at all.

You may have more than one whiteout template entry, but note that the order of the entries is important. The first entry which matches will be used.

File Action Adjustment

When this command runs, it first checks the change files against the projects files. If there are inconsistencies, the file actions will be adjusted as follows:

create — If a file is being created, but another change set is integrated which also creates the file, the file action in the change set still being developed will be adjusted to "modify".

modify — If a file is being modified, but another change set is integrated which removes the file, the file action in the change set still being developed will be adjusted to "create".

remove — If a file is being removed, but another change set is integrated which removes the file, the file will be dropped from the change set still being developed.

OPTIONS

The following options are understood:

−as-needed

Usually it is an error if a file is already in a change set, and is redundantly added to the change set again. This option says to ignore such files.

−BAse_RElative

This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative

This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be

more general than expected.

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−WhiteOut
This option may be used to request that deleted files be replaced by a "whiteout" file in the development directory. The idea is that compiling such a file will result in a fatal error, in order that all references may be found. This is usually the default.

−No_WhiteOut
This option may be used to request that no "whiteout" file be placed in the development directory.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aerm 'aegis −rm \!* −v'
sh$ aerm(){aegis −rm "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file does not exist in the baseline.

It is an error if the file is already part of the change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecp(1) copy files into a change

aedb(1) begin development of a change

aemv(1) rename a file as part of a change

aenf(1) add files to be created to a change

aermu(1)
remove files to be deleted from a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove project − remove project

SYNOPSIS

aegis −ReMove_Project *project-name* [*option*...]
aegis −ReMove_Project −List [*option*...]
aegis −ReMove_Project −Help

DESCRIPTION

The *aegis −ReMove_PRoject* command is used to remove a project, either entirely, or just from aegis' supervision.

Project aliases to the removed project are also removed.

OPTIONS

The following options are understood:

-Help

This option may be used to obtain more information about how to use the *aegis* program.

-Interactive

Specify that aegis should ask the user for confirmation before deleting each file. Answer the question *yes* to delete the file, or *no* to keep the file. You can also answer *all* to delete the file and all that follow, or *none* to keep the file and all that follow.

Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

If aegis is running in the background, the question will not be asked, and the files will be deleted.

-Keep

This option may be used to retain files and/or directories usually deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

-No_Keep

This option may be used to ensure that the files and/or directories are deleted or replaced by the command. Defaults to the user's *delete_file_preference* if not specified, see *aeuconf*(5) for more information.

−LIBrary *abspath*

This option may be used to specify a directory to be searched for global state files and user state files. (See *aegstate*(5) and *aeustate*(5) for more information.) Several library options may be present on the command line, and are search in the order given. Appended to this explicit search path are the directories specified by the *AEGIS_PATH* environment variable (colon separated), and finally, */usr/local/lib/aegis* is always searched. All paths specified, either on the command line or in the *AEGIS_PATH* environment variable, must be absolute.

-List

This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aermpr 'aegis −rmpr \!* −v'
sh$ aermpr(){aegis −rmpr "$@" −v}

ERRORS

It is an error if the project has any changes between the *being developed* and *being integrated* states, inclusive.
It is an error if the current user is not an administrator.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenpr(1)
create a new project

aenrls(1)
create a new project from an existing project

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove file undo − remove files to be deleted from a change

SYNOPSIS

aegis −ReMove_file_Undo *file-name*... [*option*...]
aegis −ReMove_file_Undo −List [*option*...]
aegis −ReMove_file_Undo −Help

DESCRIPTION

The *aegis −ReMove_file_Undo* command is used to remove files to be deleted from a change. The files is removed from the list of files in the change.

You may specify a directory name to delete from the change all files being removed in the named directory tree, other files in the tree will be ignored. It is an error if there are no relevant files.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

Notification

The *remove_file_undo_command* in the project *config* file is run, if set. The *project_file_command* is also run, if set, and if there has been an integration recently. See *aepconf*(5) for more information.

Process Side Effects

This command will cancel any build or test registrations, because deleting a file logically invalidates them.

OPTIONS

The following options are understood:

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's

$HOME/.aegisrc file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Verify_Symbolic_Links
This option may be used to request that the symbolic links, or hard links, or file copies, in the work area be updated to reflect the current state of the baseline. This is controlled by the *development_directory_style* field of the project configuration file. Only files which are not involved in the change are updated. See also the “symbolic_links_preference” field of *aeuconf*(5). This option is the default, if meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Assume_Symbolic_Links
This option may be used to request that no update of baseline mirror files take place. This options is useful when you *definitely know* the files’ up-to-date-ness isn’t important right now; incorrect use of this option may have unanticipated build side-effects. See also the “symbolic_links_preference” field of *aeuconf*(5). This option is the default, if not meaningful for your configuration. The name is an historical accident, hard links and file copies are included.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user’s *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aermu ’aegis −rmu \!* −v’
sh$ aermu(){aegis −rmu "$@" −v}

ERRORS

It is an error if the change is not in the *being developed* state.
It is an error if the change is not assigned to the current user.
It is an error if the file is not in the change.
It is an error if the was not added to the change using the *aegis −ReMove_file* command.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerm(1) add files to be deleted to a change

aeuconf(5)
user configuration file format

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove project alias − remove a project alias

SYNOPSIS

aegis −Remove_Project_Alias [*option*...] *project-alias*
aegis -Help
aegis -VERSion

DESCRIPTION

The *aegis -Remove_Project_Alias* command is used to remove a project alias.

The project alias *must* be given on the command line, the default project is not sufficient.

Example

Aliases may be used in may ways. The most common is to give a particular release a code name. You would do this by saying

```
aenpa example.4.2 sydney
```

This would make “sydney” an alias for the “example.4.2” branch.

Another use for aliases is to have a fixed alias for your active branch, so that your developer team does not need to change their default project, even though the branch number moves on for each release. You could say

```
aenpa example.4.2 example.cur
```

This would make “example.cur” an alias for the “example.4.2” branch. When this was finished, and 4.3 started, a project administrator could say

```
aerpa example.cur
aenpa example.4.3 example.cur
```

Now “example.cur” is an alias for the “example.4.3” branch, but the developers need only reference “example.cur” to always work on the right branch.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “−−*option*=*value*” convention is also understood.

ERRORS

It is an error if the current user is not a project administrator.
It is an error if the given name is not a project alias.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenpa(1)
Create a new project alias.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis review pass − pass a change review

SYNOPSIS

aegis −Review_PASS [*option*...]
aegis −Review_PASS −List [*option*...]
aegis −Review_PASS −Help

DESCRIPTION

The *aegis −Review_PASS* command is used to notify aegis that a change has passed review.

The default configuration requires only a single reviewer for each change set. It is possible to have more than one reviewer, and/or project specific policies about who may review certain files, by configuring Aegis to use an external review policy command.

The state transition performed depends on the settings of the `review_policy_command` field of the project configuration file and the `develop_end_action` field of the project attributes.

`review_policy_command` not set:
The change will be advanced from the *being reviewed* state to the *awaiting integration* state.

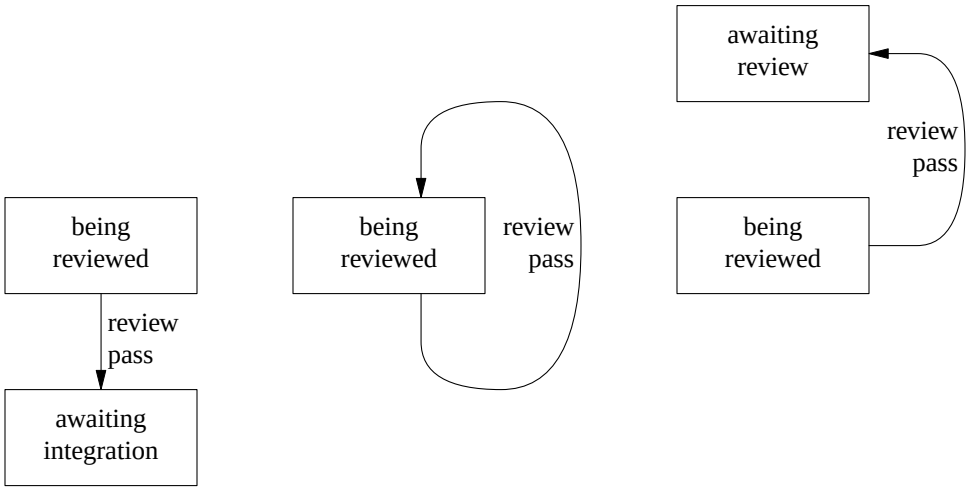
`review_policy_command` set:
The command will be executed, and the exit status examined.

Zero:
The change will be advanced from the *being reviewed* state to the *awaiting integration* state.

Non-Zero:
The setting of the `develop_end_action` of the project attributes is examined:

goto_awaiting_review:
The change will be advanced from the *being reviewed* state to the *awaiting integration* state.

Otherwise:
The change will remain in the *being reviewed* state. It is expected that a future `review_policy_command` execution will satisfy the project criteria and exit zero.



It is possible to avoid the *being reviewed* state altogether by setting the *develop_end_action* field of the project confituration file to *goto_awaiting_integration*.

If the project configuration file has specified the presence of `Signed-off-by:` lines, a suitable line containing the current user's email address will be appended to the change description.

If you use one of the −File, −Edit or −Reason options to add comments, the file is to be in plain text, and it

is recommended that you only use a newline to terminate paragraphs (rather than to terminate lines) as this will result in better formatting in the various listings.

Notification
On successful completion of this command, the *review_pass_notify_command* field of the project attributes is run, if set. See *aepattr*(5) and *aepa*(1) for more information.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Edit
Edit the attributes with a text editor, this is usually more convenient than supplying a text file. The *VISUAL* and then *EDITOR* environment variables are consulted for the name of the editor to use; defaults to *vi*(1) if neither is set. See the *visual_command* and *editor_command* fields in *aeuconf*(1) for how to override this specifically for Aegis.

Warning: Aegis tries to be well behaved when faced with errors, so the temporary file is left in your home directory where you can edit it further and re-use it with a **−file** option.

The **−edit** option may not be used in the background, or when the standard input is not a terminal.

−Edit_BackGround
Edit the attributes with a dumb text editor, this is most often desired when edit commands are being piped into the editor via the standard input. Only the **EDITOR** environment variable is consulted for the name of the editor to use; it is a fatal error if it is not set. See the *editor_command* field in *aeuconf*(1) for how to override this specifically for Aegis.

−File *filename*
Take the attributes from the specified file. The filename ‘−’ is understood to mean the standard input.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user’s *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

−Signed_Off_By
This option may be used to have a `Signed-off-by:` line appended to the change set description.

−No_Signed_Off_By
This option may be used to prevent a `Signed-off-by:` line from being appended to the change set description.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is

usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aerpass 'aegis −rpass \!* −v'
sh$ aerpass(){aegis −rpass "$@" −v}

ERRORS

It is an error if the change is not in the *being reviewed* state.
It is an error if the current user is not a reviewer of the project.
Its is an error if the current user developed the change and the project is configured to disallow developers to review their own changes (default).

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aecd(1) change directory

aede(1) complete development of a change

aedeu(1)
recall a change for further development

aeib(1) begin integrating a change

aenrv(1) add a reviewer to a project

aerfail(1)
fail review of a change

aerpu(1)
rescind a change review pass

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis review pass undo − rescind a change review pass

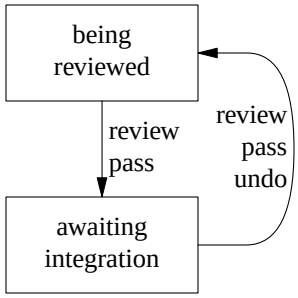
SYNOPSIS

aegis −Review_Pass_Undo [*option*...]
aegis −Review_Pass_Undo −List [*option*...]
aegis −Review_Pass_Undo −Help

DESCRIPTION

The *aegis −Review_Pass_Undo* command is used to notify aegis that a change review pass has been rescinded.

The change will be moved from the *awaiting integration* state to the *being reviewed* state.



Notification

On successful completion of this command, the *review_pass_undo_notify_command* field of the project attributes is run, if set. See *aepattr*(5) and *aepa*(1) for more information.

OPTIONS

The following options are understood:

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−REAson *text*
This option may be used to attach a comment to the change history generated by this command. You will need to use quotes to insulate the spaces from the shell.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aerpu 'aegis −rp \!* −v'
sh$ aerpu(){aegis −rp "$@" −v}

ERRORS

It is an error if the change is not in the *awaiting integration* state.
It is an error if the current user is not the reviewer of the change.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerpass(1)
pass review of a change

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis remove reviewer − remove reviewers from a project

SYNOPSIS

aegis −Remove_ReViewer *user-name*... [*option*...]
aegis −Remove_ReViewer −List [*option*...]
aegis −Remove_ReViewer −Help

DESCRIPTION

The *aegis −Remove_ReViewer* command is used to remove reviewers from a project.

OPTIONS

The following options are understood:

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Descend_Project_Tree
This option may be used to request that the command should be applied to the project and all its branches and sub-branches.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf*(5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is

```
csh%    alias aerrv 'aegis −rrv \!* −v'
sh$     aerrv(){aegis −rrv "$@" −v}
```

ERRORS

It is an error if the current user is not an administrator of the project.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenrv(1) add a new reviewer to a project

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aesub − substitute and echo strings

SYNOPSIS

aesub [*option*...] string ...
aesub −Help
aesub −VERSion

DESCRIPTION

The *aesub* command is used to perform the usual *aesub*(5) substitutions on its command line arguments, and then echo them to the standard output.

Shell Script Quoting

The *aesub*(1) command is often used in shell scripts. It is important to remember that the shell will do its own substitutions on the command line argument *before* it invokes the *aesub*(1) command. Usually, you don't want this to happen, so you need to use *single* (**'**) quotes to do this. (The shell continues to substitute inside double (**"**) quotes.)

Quote *aesub*(1) arguments using **'** *single* **'** quotes.

OPTIONS

The following options are understood:

-BaseLine
This option may be used to specify that the project baseline is the subject of the command.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−File *filename*
Take the text to be substituted from the specified file. The filename '−' is understood to mean the standard input.

-Help
This option may be used to obtain more information about how to use the *aesub* program.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aesub* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aesub* command will exit with a status of 1 on any error. The *aesub* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aesub(5)
Available string substitutions.

COPYRIGHT

aesub version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aesubunit − run SubUnit tests

SYNOPSIS

aesubunit [*option*...] *filename*...
aesubunit −Help
aesubunit −VERSion

DESCRIPTION

The **aesubunit** command is used to invoke tests via the http://www.robertcollins.net/unittest/subunit unit testing interface.

The shape of the external unittest should not need to be known *a priori*. After the test has run, tests should still exist as discrete objects, so that anything taking a reference to them doesn't get 50 copies of the same object.

The *aesubunit* command may be used to replace the *test_command* or *batch_test_command* fields of the project configuration file.

Control Protocol

The results of the test are obtained by examining the standard output of the tests as they run. The text is meant to be human readable, so that tests may run stand-alone.

Tests should ideally print a header of the form

`test`
`testing`
`test:` *test label*
`testing:` *test label*

A successful test will produce lines of the form

`success`
`success:`
`successful` *test label*
`successful:` *test label*

A test failure will produce text of the form

`failure` *test label* `failure:` *test label* `failure` *test label* `[ ]` `failure:` *test label* `[ ]`

The square brackets indicate text which may describe the test in more detail. It will be printed on the standartd output by the *aesubunit* program.

A test which produces no result (neither succes nor failure) uses the following forms

`error:` *test label* `error:` *test label* `[ ]`

In general, unexpected output from the test will be sent through to the *aesubunit* standard output.

If a subunit test terminates with an exit status other than zero, this is taken to be a no result indication for that test.

OPTIONS

The following options are understood:

−Batch This option may be useed to specify that a batch test should be performed, and produce results in the appropriate form.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-Help

This option may be used to obtain more information about how to use the *aesubunit* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aesubunit* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aesubunit* command will exit with a status of 1 on any error. The *aesubunit* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aesubunit version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aesvt − simple version tool

SYNOPSIS

aesvt −CHeck_Out −HIstory *file* **−File** *output-file* [**−e** *edit*]
aesvt −CHeck_In −HIstory *file* **−File** *input-file* [**−e** *edit*] [*name*=*value* ...]
aesvt −List −HIstory *file*
aesvt −Query −HIstory *file*
aesvt −Version

DESCRIPTION

The *aesvt* program may be used to manage history version files. This is a minimalist history tool, which makes no provision for managing a work area.

It is able to cope with binary files, and with reasonable efficiently if they are not too large.

It has good end-to-end properties because it keeps a checksum for each file version, and a checksum for the whole history file.

There is no provision for keyword substitution of any kind. A check-out will exactly reproduce the input file. A check-in will never alter the input file.

OPTIONS

The following options are understood:

−History *history-file*
This option is used to specify the name of the history file.

−File *file-name*
This option is used to specify the name of the input or output file. On check-out, the file name "−" is understood to mean the standard output. There is **no** equivalent for check-in.

−Edit *edit-number*
This option is used to specify the edit number (version number). On check-out, if no version number is specified, the most recent version is given. On check-in, if no version number is specifiued (and it usually isn't), the previous version will have one added to it, or version 1 will be used if this is the first check-in.

−CHeck_In
This option is used to check a file into the history.

−CHeck_Out
This option is used to check-out a file from the history.

−compression-algorithm *name*
This option may be used to specify the compression to be used. They are listed on order of compression effeciency.

none Use no compression (not always meaningful for all commands).

gzip Use the compression used by the *gzip*(1) program.

bzip2 Use the compression used by the *bzip2*(1) program.

More compression algorithms may be added in the future.

−COMPress
This option is deprecated in favour of the **−comp-alg=gzip** or **−comp-alg=bzip2** options.

−No_COMPress
This options is deprecated in favour of the **−comp-alg=none** option.

−List This option is used to list the file's history.

−Query This option is used to query edit number of most recent check-in.

−Version

This option is used to print version number.

All other options will produce a diagnostic error.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aesvt* are long, this means ignoring the extra leading '−'. The "−−*option=value*" convention is also understood.

EXIT STATUS

The *aesvt* command will exit with a status of 1 on any error. The *aesvt* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

FILE FORMAT

Each version in the history file consists of an RFC822 header, plus the file contents. The header includes (at least) the Content-Length, used to remember the length of the file data in bytes; the Checksum, used to remember the Adler32 checksum of the file data; and Version, used to remember the version number. The file data can be text or binary, because its length is determined by the header. There is no quoting mechanism of any kind for the data. Except for the mandatory fields, additional user-defined us-ascii meta-data may also be stored in the header. There is no diff or delta of any kind for any version.

This combination of header and data has good end-to-end behaviour, because there is a checksum to validate the file data against. Bad blocks in the data will be detected then next time a check-in or check-out is attempted.

The format of the history file consists of one or more file versions with the above layout, joined head-to-tail with no separators or boundary indicators of any kind. The versions are in descending order, from most recent (greatest edit number) to least recent (version number one). To determine where one version stops and the next version starts, use the Content-Length field in the header. The entire history file is then compressed using the bunzip2 algorithm (via libbz2). There is no diff or delta of any kind in the history file.

The advantage of compressing the file is that there is usually a very high redundancy between file versions. For example, if two identical versions are checked in (not necessarily sequentially) the second copy will compress to only a few bytes. Unlike *diff*(1) style deltas, this also copes very will with moving blocks of data within the file. The use of bunzip2 formatting means there is also a checksum for the whole history file, which allows you to detect bad blocks in the header portions; it also means there is a simple way to extract the data from a history file even without the *aesvt* program, or for testing, or because you are curious.

You can actually choose from a number of compression algorithms, including GNU Zip and bunzip2, via the `−compression-algorithm` option. More copmpresison algoritthms may be added in the future. The best available comression is used, because this results in the most compact history files. Future versions will always be able to access the compression used by earlier versions.

End-To-End Issues

See also Saltzer, J.H. *et al* (1981) *End-to-end arguments in system design*, http://web.mit.edu/Saltzer/-www/publications/endtoend/endtoend.pdf

Xdelta

This style of history file was inspired by RFC 3284 − *The VCDIFF Generic Differencing and Compression Data Format*. While the *aesvt* format does not use RFC3284 internally, the arguments for compression across file versions are just as relevant.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis test − run tests

SYNOPSIS

aegis −Test [*option*...][*name=value*][*file-name*...]
aegis −Test −INDependent [*option*...][*name=value*][*file-name*...]
aegis −Test −List [*option*...]
aegis −Test −Help

DESCRIPTION

The *aegis −Test* command is used to run tests. If no files are named, all relevant tests are run. By default both automatic and manual tests are run.

You may name directories on the command line, and all relevant tests in that directory tree in the change will be run. It is an error if there are no relevant tests.

Each architecture must be tested separately. This is because there may be subtle problems that are only revealed on some architectures. Some projects may also have different code for different architectures.

The status of the last test run is remembered so that tests are not run if there is no need. (This does not apply to *−REGression* tests, unfortunately.) Tests must be re-run if the test previously failed, if the test file has changed, if there has been a build, and for each architecture.

name=value

You can add *name=value* pairs to the command line, these will be passed unchanged to the test command. Usually on the end of the command line, but this can be changed in the project configuration file.

The **−force** option results in an implicit `force=1` variable being added to the list of variable assignments, and thus added to the end of the command. This is of most use when using the *batch_test_command* filed of the project configuration file.

This may initially look like a development process end-run, allowing test scripts to be written so that they give all the right answers without actually doing anything. You have always been able to do this with environment variables, so this isn't anything new.

It is possible to get all of the variable assignments to turn into environment variables by putting `$var` at the *start* of the command, before the name of the shell, rather than at the default location at the end of the command.

File Name Interpretation

The aegis program will attempt to determine the project file names from the file names given on the command line. All file names are stored within aegis projects as relative to the root of the baseline directory tree. The development directory and the integration directory are shadows of this baseline directory, and so these relative names apply here, too. Files named on the command line are first converted to absolute paths if necessary. They are then compared with the baseline path, the development directory path, and the integration directory path, to determine a baseline-relative name. It is an error if the file named is outside one of these directory trees.

The **−BAse_RElative** option may be used to cause relative filenames to be interpreted as relative to the baseline path; absolute filenames will still be compared with the various paths in order to determine a baseline-relative name.

The *relative_filename_preference* in the user configuration file may be used to modify this default behavior. See *aeuconf*(5) for more information.

TEST PROCESS

Each change is required to be accompanied by tests, and those tests are required to be run against the built development directory, and they must pass. This ensures that new functionality is accompanied by tests to verify its correctness, and bug fixes are accompanied by tests which confirm that the bug has been fixed.

Regression Tests

Tests are treated as any other source file, and are maintained in the baseline and history with all other source files. The tests which must accompany every change accumulate in the project baseline, providing a

definition of correct function for the baseline. These accumulated tests may be executed using an “aegis −REGression” command, to verify that the project will not “regress” as a result of a change.

Baseline Tests

Bug fixes are required to have their tests *fail* against the project baseline (in contrast to the development directory). This ensures that the test actually demonstrates the bug in the baseline, as well as demonstrating that it is fixed by the change. New functionality trivially fails against the baseline, and so aegis does not attempt to guess if a test is a bug fix test or new functionality test, it simply requires tests to fail against the baseline.

This requirement applies both to new tests being created by a change and also to tests which have been copied into a change for modification.

Reviewing Tests

Reviewers may be confident that aegis has enforced the test requirements; that a change must have tests, that the change must build, that the tests pass against the development directory, and that the tests fail against the baseline. These conditions are enforced by *aede*(1) and the change will not be advanced to the *being reviewed* state until these conditions are met. Reviewers should thus review tests for *completeness* of coverage of the code in the change, and insensitivity to changes in the execution environment (e.g. not date sensitive). Reviewers should also use “aegis −list change_details” to verify that a change does or does not have testing exemptions.

Exemptions

Various test exemptions may be granted by project administrators, see *aepa*(1) and *aepattr*(5) for more information. Copying tests into a change, or adding new tests to a change, may cancel those exemptions.

TEST COMMAND CONFIGURATION

The command used to execute tests is defined by the *test_command* field in the project configuration file (see *aepconf*(5) for more information), this defaults to using the Bourne shell if not set. The current directory will be the top of the appropriate directory tree. If tests require temporary files, they should create them in */tmp*, as a test cannot expect to have write permission in the current directory.

If you want to use a more sophisticated test engine, rather than a simple shell script, but this test engine does not return result codes suitable for use with aegis, you could wrap it in a shell script which re-writes the exit status into the values aegis expects. You could also achieve the same results by writing a more complex *test_command* in the project *config* file.

It is also possible to write test commands which are able to test more than one file at once. This is controlled by the *batch_test_command* field of the project *config* file. In this case, the ${output} substitution indicates the name of a file the test command must create, in *aetest*(5) format, to contain the results of the tests run. This is often used on systems with multiple CPUs or the ability to distribute jobs across several computers on a network.

Substitutions

All of the *aesub*(5) substitutions are available in the test commands. Some of them are of particular note:

ARCHitecture
: This substitution is replaced by the name of the architecture to be tested.

Search_Path
: This substitution is replaced by a colon separated list of absolute paths to search when looking for test support files.

Search_Path_Executable
: This substitution is replaced by a colon separated list of absolute paths to search when looking for executable support files (library files and sub-commands).

Most of the time *$Search_Path_Executable* are exactly the same. However, during “aegis −t −bl” they will be different, with *$Seach_Path* starting at the development directory (the test being run) and *$Seach_Path_Executable* starting at the baseline (the executable being run).

Test Result Codes

As each test is run (via the *test_command* field in the project *config* file), aegis determines whether the test succeeded or failed by looking at its exit status. This exit status is mostly as expected for UNIX commands.

Success
: A test should exit 0 to indicate success, i.e. that the specific function under test worked as expected.

Failure
: A test should exit 1 to indicate failure, i.e. that the specific function under test did not work as expected.

No Result
: A test should exit 2 to indicate no result, *i.e.* that the specific function under test could not be exercised because something else went wrong. For example, running out of disk space when creating the test input files in the */tmp* directory.

Skipped
: A test should exit 77 to indicate that it was skipped. This is usually to do with the current architecture not being meaningful. Whenever possible, use “No Result” instead. (The value was chosen for compatibility with other test systems.)

Actually, any exit code other than 0, 1 or 77 will be interpreted as “no result”. However, always using 0, 1, 2 or 77 means that if a new result code is required by a later release of Aegis your existing tests will continue to work.

TEST CORRELATIONS

The “aegis −Test −SUGgest” command may be used to have aegis suggest suitable regression tests for your change, based on the source files in your change. This automatically focuses testing effort to relevant tests, reducing the number of regression tests necessary to be confident that you have not introduced a bug.

The test correlations are generated by the “aegis −Integrate_Pass” command, which associates each test in the change with each source file in the change. Thus, each source file accumulates a list of tests which have been associated with it in the past. This is not as exact as code coverage analysis, but is a reasonable approximation in practice.

The *aecp*(1) and *aenf*(1) commands are used to associate files with a change. While they do not actively perform the association, these are the files used by *aeipass*(1) and *aet*(1) to determine which source files are associated with which tests.

Test Correlation Accuracy

Assuming that the testing correlations are accurate and that the tests are evenly distributed across the function space, there will be a less than *1/number* chance that a relevant test has not been run by the “aegis −Test −SUGgest *number*” command. A small amount of noise is added to the test weighting, so that unexpected things are sometimes tested, and the same tests are not run every time.

Test correlation accuracy can be improved by ensuring that:

- Each change should be strongly focused, with no gratuitous file inclusions. This avoids spurious correlations.
- Each item of new functionality should be added in an individual change, rather than several together. This strongly correlates tests with functionality.
- Each bug should be fixed in an individual change, rather than several together. This strongly correlates tests with functionality.
- Test correlations will be lost if files are moved. This is because correlations are by name.

The best way for tests to correlate accurately with source files is when a change contains a test and exactly those files relating to the functionality under test. Too many spurious files will weaken the usefulness of the testing correlations.

OPTIONS

The following options are understood:

-AUTOmatic
This option may be used to specify automatic tests. Automatic tests require no human assistance.

-BaseLine
This option may be used to specify that the project baseline is the subject of the command.

−BAse_RElative
This option may be used to cause relative filenames to be considered relative to the base of the source tree. See *aeuconf*(5) for the corresponding user preference.

−CUrrent_RElative
This option may be used to cause relative filenames to be considered relative to the current directory. This is usually the default. See *aeuconf*(5) for the corresponding user preference.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−FOrce This option may be used to specify that all tests should be run, even if the status of the last test run indicates that there is no need to run a specific test.

-Help
This option may be used to obtain more information about how to use the *aegis* program.

-INDependent
This option is used to specify that the test is to be run independent of any particular change. If no tests are named, all tests in the baseline will be run.

-List
This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

-MANual
This option may be used to specify manual tests. Manual tests require some human intervention, e.g.: confirmation of some screen behavior (X11, for instance), or some user action, "unplug ethernet cable now".

-Not_Logging
This option may be used to disable the automatic logging of output and errors to a file. This is often useful when several aegis commands are combined in a shell script.

−PErsevere
This option may be used to specify that all tests should be run, even if some fail. Defaults to the user's *persevere_preference* if not specified, see *aeuconf* (5) for more information.

−No_PErsevere
This option may be used to specify that the test run should stop after the first failure. Defaults to the user's *persevere_preference* if not specified, see *aeuconf* (5) for more information.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf* (5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−PROGress
This option may be used to specify that progress messages should be issued before each test run or before each batch test run in case *batch_test_command* field specified in project *config* file (see *aeuconf* (5) for more information).

−No_PROGress
This option may be used to specify that progress messages should be suppressed. This is the default.

−REGression
This option is used to specify that the regression test suite is to be run. The regression test suite consists of all tests in the baseline which do not appear in the change. It is an error if there are no regression tests. You may not name tests on the command line when using the −REGression option. You may name individual tests to be run on the command line, without using the −REGression option; if they are not part of the change, the tests of the same name in the baseline will be run.

−SUGgest [*number*]
The "*aegis −Integrate_Pass*" command collects test correlation statistics when changes are integrated. This option may be used to request that aegis suggest which tests should be run, using these testing correlations. If no number is specified, 10 tests will be suggested. This option implies the **−REGression** option.

−SUGgest_Limit *minutes*
This option may be used to limit the number of tests to a certain number of minutes. They will be run from most relevant to least relevant.

−SUGgest_Noise *number*
This option may be used to control the amount of noise injected into the test selection performed by the **−SUGgest** option. The number is a percentage of noise to be injected. Defaults to 10 if not specified. The injection of noise ensures that a variety of tests are run on subsequent runs, and also some from left-field as a sanity check.

-TERse
This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts.

-Verbose
This option may be used to cause aegis to produce more output. By default aegis only produces output on errors. When used with the **-List** option this option causes column headings to be added.

−Wait This option may be used to require Aegis commands to wait for access locks, if they cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

−No_Wait
This option may be used to require Aegis commands to emit a fatal error if access locks cannot be obtained immediately. Defaults to the user's *lock_wait_preference* if not specified, see *aeuconf* (5) for more information.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading '−'. The "*−−option=value*" convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aet 'aegis −t \!* −v'
sh$ aet(){aegis −t "$@" −v}

ERRORS

It is an error if the change is not in one of the *being developed* or *being integrated* states.
It is an error if the change is not assigned to the current user.
It is an error if your have no relevant tests and no relevant exemption.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeb(1) build a change

aeca(1) modify the attributes of a change

aedb(1) begin development of a change

aeib(1) begin integration of a change

aent(1) add a new test to a change

aecp(1) copy an existing test into a change

aepconf(5)
project configuration file format

aeuconf(5)
user configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aetar − remotely distribute a change via tar

SYNOPSIS

aetar −Send [*option*...]
aetar −Receive [*option*...]
aetar −List [*option*...]
aetar −Help
aetar −VERSion

DESCRIPTION

The *aetar* command is used to send and receive change sets via *tar*(1) to facilitate geographically distributed development.

The basic function is to reproduce a change, so a command like

```
aetar −send | aetar −receive
```

may be used to clone a change, though less efficiently than *aeclone*(1). The file format used is an ordinary *gzip*(1) compressed *tar*(1) archive.

SEND

The send variant takes a specified change, or baseline, and constructs a distribution package containing all of the source file contents. No change meta-data is included.

It is not necessary for the recipient to have the *aetar*(1) command. It is possible to use the regular *tar xzf* command to extract the files from the archive.

Options

The following options are understood by the send variant:

−BaseLine
This option may be used to specify the source of a project, rather than a change.

−Add_Path_Prefix *string*
This option may be used to specify a path prefix to be added to every filename in the archive. This means that when the archive is unpacked, it will all be placed in the one directory.

−Change *number*
This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−COMPATibility *version-number*
This option may be used to specify the version of *aetar*(1) which will be *receiving* this change set. This information is used to select which features to include in the data, and which to omit. By default, the latest feature set will be used.

−compression-algorithm *name*
This option may be used to specify the compression to be used. They are listed on order of compression effeciency.

none Use no compression (not always meaningful for all commands).

gzip Use the compression used by the *gzip*(1) program.

bzip2 Use the compression used by the *bzip2*(1) program.

More compression algorithms may be added in the future.

−COMPress
This option is deprecated in favour of the **−comp-alg=gzip** or **−comp-alg=bzip2** options.

−No_COMPress
This options is deprecated in favour of the **−comp-alg=none** option.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, rather than the most current version. If the delta has been given a name (see *aedn*(1) for how) you may use a delta name instead of a delta number. It is an error if the delta specified does not exist. Delta numbers start from 1 and increase; delta 0 is a special case meaning "when the branch started".

−DELta_Date *string*
This option may be used to specify a particular date and time in the project's history to copy the file from, rather than the most current version. It is an error if the string specified cannot be interpreted as a valid date and time. Quote the string if you need to use spaces.

−DELta_From_Change *number*
This option may be used to specify a particular project delta from its change number.

−Entire_Source
This option may be used to send the entire source of the project, as well as the change source files. This is the default.

−Partial_Source
This option may be used to send only source files of a change.

−Include_Build
This option may be used to send also build files.

−Not_Include_Build
This option may be used to send only source (source, test, config but not build) files. This is the default.

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

RECEIVE

The receive variant takes a tarball and creates an Aegis change (see *aenc*(1)) to implement the change within. Files are added to the change (see *aenf*(1), *aecp*(1), *aerm*(1), *aent*(1)) and then the file contents are unpackaged into the development directory.

It is not necessary for the sender to have the *aetar*(1) command. It is possible to use the regular *tar czf* command to create the the tarball. You may want to use the *tardy*(1) command to manipulate the filenames before extraction.

File Names

It is common for tar files generated to distribute open source projects to contain a path prefix.

−Remove_Path_Prefix *string*
This option may be used to explicitly specify path prefixes to be removed, if present. It may be specified more than once.

−Remove_Path_Prefix *number*
Strip the smallest prefix containing num leading slashes from each file name found in the patch file. A sequence of one or more adjacent slashes is counted as a single slash.

If you have a complex project directory structure, from time to time people may send you tarballs relative to a sub-directory, rather than relative to the project root.

−Add_Path_Prefix *string*
This option may be used to specify the path of a project sub-directory in which to apply the tarball.

Notification
The *aetar* command invokes various other Aegis commands. The usual notifications that these commands would issue are issued.

Options
The following options are understood by the receive variant:

−Change *number*
This option may be used to choose the change number to be used, otherwise one will be chosen automatically.

−DELta *number*
This option may be used to specify a particular delta in the project's history to copy the file from, just as for the *aecp*(1) command. You may also use a delta name instead of a delta number.

−DIRectory *path*
This option may be used to specify which directory is to be used. It is an error if the current user does not have appropriate permissions to create the directory path given. This must be an absolute path.

Caution: If you are using an automounter do not use ‘pwd‘ to make an absolute path, it usually gives the wrong answer.

−EXCLude
This option may be used to exclude certain files in the tarball from consideration.

You can also add more exclusions using the *project_specific* field of the project configuration, using the `aetar:exclude` attribute listing file names to exclude separated by spaces.

−Exclude_Auto_Tools
This option may be used to exclude files common to tarballs of open source projects which used GNU Autoconf or GNU Automake. This is triggered by the presence of *configure.ac*, *configure.in* or *Makefile.am* files. This only works for simple projects, more complex projects will need to use the project exclude attributes.

You can set this automatically using the boolean `aetar:exclude-auto-tools` attribute in the *project_specific* field of the project configuration file.

−Exclude_CVS
This option may be used to exclude files common to CVS repositories, which implement the repository functions, rather than contain source code. It will also look inside `.cvsignore` files for additional files to ignore.

You can set this automatically using the boolean `aetar:exclude-cvs` attribute in the *project_specific* field of the project configuration file.

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename ‘−’ is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Project *name*
This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more

information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

−Trojan This option may be used to treat the change set as if it had a Trojan horse attack in it.

−No_Trojan
This option may be used to treat the change set as if it definitely does not have a Trojan horse attack in it. *Use with extreme care.* You need to have authenticated the message with something like PGP first **and** know the the author well.

Security

Downloading a tarball and automatically committing it to the baseline without checking it would be a recipe for disaster. A number of safeguards are provided:

- The file sare unpacked into a new change. You need to edit the change description. You need to uncopy unchanged files. You need to difference the change. You need to build and test the change. This ensures that a local reviewer validates the change before it is committed, preventing accidental or malicious damage.
- The use of authentication and encryption systems, such as PGP and GPG, are encouraged. However, it is expected that this processing will occur after *aetar −send* has constructed the package and before *aetar −receive* examines and acts on the package. Verification of the sender is the surest defense against trojan horses.
- Automatic sending and receiving of packages is supported, but not implemented within the aetar command. It is expected that the aetar command will be used within shell scripts customized for your site and its unique security requirements. See the Aegis User Guide for several different ways to do this.
- The more you use Aegis' test management facilities (see *aent*(1) and *aet*(1)) the harder it is for an inadequate change to get into the baseline.

LIST

The list variant can be used to list the contents of a tarball without actually unpacking it first.

Options

The following options are understood by the list variant:

−File *filename*
Read the change set from the specified file. The default is to read it from the standard input. The filename '−' is understood to mean the standard input.

If your system has *libcurl*(3), and Aegis was configured to use it at compile time (this is the default if it is available) you will also be able to specify a Uniform Resource Locator (URL) in place of the file name. The relevant data will be downloaded. (The **−Verbose** option will provide a progress bar.)

−Output *filename*
This option may be used to specify the output file. The output is sent to the standard output by default. Only useful with the −List option.

OPTIONS

The following options to this command haven't been mentioned yet:

-Help
This option may be used to obtain more information about how to use the *aetar* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aetar* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

FILE FORMAT

The file format re-uses existing formats, rather than introduce anything new. This means it is possible to extract the contents of a package even when aetar is unavailable.

- The source files and other information is stored as a normal Unix *tar*(1) archive.
- On sending, the tarball is compressed using the GNU gzip format. Typically primary source files are ASCII text, resulting in significant compression. (This is optional.)
  On receiving, if the tarball is compressed it will be automagically uncompressed, detection is automatic, you do not need to do this yourself.

EXIT STATUS

The *aetar* command will exit with a status of 1 on any error. The *aetar* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aetar version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis version − give version information

SYNOPSIS

aegis −VERSion [*info-name*]
aegis −VERSion −Help

DESCRIPTION

The *aegis −VERSion* command is used to give version information and conditions of use.

There are a number of possible *info-name*s, as follow (abbreviations as for command line options):

Copyright
: The copyright notice for the aegis program. Version information will also be printed. This is the default.

Redistribution
: Print the conditions of use and redistribution.

Warranty
: Print the limited warranty.

OPTIONS

The following options are understood:

-Help
: This option may be used to obtain more information about how to use the *aegis* program.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments “−project”, “−PROJ” and “−p” are all interpreted to mean the **−Project** option. The argument “−prj” will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aegis* are long, this means ignoring the extra leading ’−’. The “*−−option=value*” convention is also understood.

RECOMMENDED ALIAS

The recommended alias for this command is
csh% alias aev ’aegis −vers \!*’
sh$ aev(){aegis −vers "$@"}

ERRORS

It is an error if the *info-name* given is unknown.

EXIT STATUS

The *aegis* command will exit with a status of 1 on any error. The *aegis* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aexml − Aegis database to XML

SYNOPSIS

aexml [*option*...] *listname*
aexml -Help
aexml -List
aexml -VERSion

DESCRIPTION

The *aexml* command is used to extract portions of Aegis' database in XML format.

List Names

The following list names may be given

Change_Files
: Internal change file state. See *aefstate*(5) for structure.

Change_State
: Internal change state. See *aecstate*(5) for structure.

Projects List of projects. See *aegstate*(5) for structure.

Project_Change_State
: Internal project change state. See *aecstate*(5) for structure.

Project_Config_File
: The project configuration file. See *aepconf*(5) for structure.

Project_Files
: Internal project file state. See *aefstate*(5) for structure.

Project_Files_By_Delta
: Historical project file state as it appeared immediately after the integrate pass of the specified change or delta. See *aefstate*(5) for structure.

Project_State
: Internal project state. See *aepstate*(5) for structure.

User_Config_File
: User configuration file. See *aeuconf*(5) for structure.

The abbreviations foe the list names follows the same rules as for command line options.

OPTIONS

The following options are understood:

−Change *number*
: This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

-Help
: This option may be used to obtain more information about how to use the *aexml* program.

-List
: This option may be used to obtain a list of suitable subjects for this command. The list may be more general than expected.

−Output *filename*
: This option may be used to specify the output file. The output is sent to the standard output by default.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

-TERse

This option may be used to cause listings to produce the bare minimum of information. It is usually useful for shell scripts. If the file name ends with ".gz" it will be compressed with the *gzip*(1) algorithm, if it ends with ".bz" or ".bz2" it will be compressed with the *bzip2*(1) algorithm.

All options may be abbreviated; the abbreviation is documented as the upper case letters, all lower case letters and underscores (_) are optional. You must use consecutive sequences of optional letters.

All options are case insensitive, you may type them in upper case or lower case or a combination of both, case is not important.

For example: the arguments "−project", "−PROJ" and "−p" are all interpreted to mean the **−Project** option. The argument "−prj" will not be understood, because consecutive optional characters were not supplied.

Options and other command line arguments may be mixed arbitrarily on the command line, after the function selectors.

The GNU long option names are understood. Since all option names for *aexml* are long, this means ignoring the extra leading '−'. The "−−*option*=*value*" convention is also understood.

EXIT STATUS

The *aexml* command will exit with a status of 1 on any error. The *aexml* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aexml version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aexver − graphical file history

SYNOPSIS

aexver

DESCRIPTION

The *aexver* command is used to to view historical versions of files in an Aegis repository.

A list box is displayed with all project filenames in it. When ther user double-clicks on a file, another box is displayed listing all the revisions of that file. Selecting any two revisions will bring up a diff comparing those revisions to each other.

ENVIRONMENT VARIABLES

AE2DIFF

The name of the program to perform the 2-way diff. Default to `xidff` if not set.

EXIT STATUS

The *aexver* command will exit with a status of 1 on any error. The *aexver* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

COPYRIGHT

aexver version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

tkaeca − GUI interface for aeca, using TCL/TK

SYNOPSIS

tkaeca

DESCRIPTION

The *tkaeca* command is used to provide a GUI interface to *aeca*(1). Its use should be self-evident to anyone familiar with Aegis.

The top line of the screen contains button for selecting the project and the change. They will be defaulted whenever possible, using the usual Aegis defaulting rules. Click on the buttons to obtain a pick list if you want to change; then double-click an item to select it. (I would have used tk_optionMenu, but it doesn't have a scroll bar, even when all the items don't fit on the screen.)

The middle section contains text entry areas, for editing the *brief_description* and *description* fields of the change attributes. The text wraps in a natural way, both here and in (say) the "*tkaeca −list chand details*" listing, so only use newlines to indicate end-of-paragraph.

The lower section contains two boxes. The first is the testing required for the change − select as many or as few as is appropriate. The second a set of radio buttons to select the change cause − pick one.

The "OK" and "Cancel" buttons do what you expect. The cancel button simply quits. The OK button runs the *aeca*(1) command − if anything goes wrong (*e.g.* asking for testing exemptions you can't have) then the error message will be displayed in the message area at the bottom of the window.

OPTIONS

There are no command line options. It is best to run this command in the background.

EXIT STATUS

The *tkaeca* command will exit with a status of 1 on any error. The *tkaeca* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aeca(1) View and edit change attributes, from the command line.

tkaegis(1)
GUI interface for Aegis, using TCL/TK

COPYRIGHT

tkaeca version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

tkaegis − GUI interface for Aegis, using TCL/TK

SYNOPSIS

tkaegis

DESCRIPTION

The *tkaegis* command is used to provide a GUI interface to Aegis. Its use should be self-evident to anyone familiar with Aegis.

There are some areas where tkaegis is still missing functionality; these are primarily related to project and change attributes that are not yet included in the dialogs, and also issues such as configuring the change and history tools, the architectures, and so on.

INSTALLATION

First, you will need Tcl/Tk installed, and will need to modify the path in the first line of tkaegis to reflect the path of your Tk wish interpreter. Hopefully, the *./configure* script took care of this.

Next, you may need to modify some important variables that occur immediately below these comments, to specify the architecture, project base directory, and the editor you are using (if you leave that blank, tkaegis will try to determine the editor to use from the EDITOR environment variable; if that fails, it will fall back to emacs or vi).

NAVIGATION

When you run tkaegis, a window will appear with a menu at the top. The window is used to display the output of aegis commands and some other feedback. The menu will initially have only two items, Project and Help. At this stage the Help menu only has an About dialog box.

The Project menu will allow you to create new projects, select from your existing projects, clear the contents of the feedback window, or exit the program. If you create a new project, a dialog box will appear allowing you to enter the project name, directory, and initial branch number. When you press OK the project will be created and should then appear in the Project menu.

If you select an existing project, a new option will be added to the Project menu, allowing you to delete the project. A Branch menu will also appear. This is similar to the Project menu, but allows you to create, delete, or select project branches.

If you select a branch in the branch menu, a Role menu will appear. This will allow you to choose the role that you will be playing, namely one of administrator, developer, reviewer, or integrator. tkaegis uses your UNIX login name and the names of the roleplayers associated with the project and branch, to determine which of the roles it will allow you to choose. If you create a new project, only the administrator role will appear.

Selecting a role will put you in a ‘mode’, which will determine what other menus appear and what you can do next. Each mode will now be described in turn, by giving a brief description of the role-specific menu hierarchy. Following the name of each menu item is the corresponding aegis command, where applicable.

ADMINISTRATOR MODE

In this mode, you can modify the staff and roles associated with the branch, and create, remove, and change the attributes of change requests, and view all the change requests.

Admin:

Edit Branch Attributes (aepa) −
Change the attributes for the branch

Staff:

Administrators:

Add (aena) −
Add an administrator for the branch

View (ael a) −
View the administrators for the branch

Remove (aera) −
Remove an administrator for the branch

Developers:

Add (aend) −
Add a developer for the branch

View (ael d) −
View the developers for the branch

Remove (aerd) −
Remove a developer for the branch

Reviewers:

Add (aenrv) −
Add a reviewer for the branch

View (ael r) −
View the reviewers for the branch

Remove (aerrv) −
Remove a reviewer for the branch

Integrators:

Add (aeni) −
Add an integrator for the branch

View (ael i) −
View the integrators for the branch

Remove (aeri) −
Remove an integrator for the branch

Change:

Add New Change (aenc) −
Add a new change request

New Change Undo (aencu) −
Undo the addition of a change request

Edit Change Attributes (aeca) −
Modify the attributes of a change request

View Changes (ael c) −
View the set of changes

DEVELOPER MODE

This mode is used by developers. When entering this mode, the Develop menu will appear, but no others. A change must be selected after which the other menus will appear. If there is only one change awaiting development, this will be auto-selected.

Develop:

View Changes (ael c) −
View all the change requests

Begin Change (aedb) −
Start work on a new change

Continue Change −
Continue work on a change in development

View Differences (aediff) −
Show all the diffs for this change

Abort Change (aedbu) −
Abort working on the change

End Change (aede) −
(Attempt to) end working on the change

Resume Change (aedeu) −
Resume work on a change awaiting review

File:

Edit Files −
Allow files to be loaded into an editor

Add New File (aenf) −
Add a new file to the project

Discard New File (aenfu) −
Discard a newly added file

Remove Existing File (aerm) −
Discard a previously existing file

Restore Existing File (aermu) −
Undo discard of a previously existing file

Change Existing File (aecp) −
Allow an existing file to be edited

Undo Changes to Existing File (aecpu) −
Lose changes to an existing file

Build:

Build Project (aeb) −
Attempt to build the project

Test:

Add New Test Script (aent) −
Add a new test script to the project

Discard New Test Script (aentu) −
Remove a new test script

Run New Tests (aet) −
Run the new tests

Run Regression Tests (aet −reg) −
Run the old tests

Run Baseline Test (aet −bl) −
Run the baseline test

REVIEWER MODE

In this mode you are able to review changes.

Review:

View Changes (ael c) −
View all the changes

Begin Review (aerb) −
Start reviewing a change

Abort Review (aerbu) −
Abort reviewing a change

Pass (aerpass) −
Pass a change review

Fail (aerfail) −
Fail a change review

Undo Pass (aerpu) −
Undo a previously passed review

INTEGRATOR MODE

In this mode you can perform integration activities.

Integrate:

View Changes (ael c) −
View all the changes

Start Integration (aeib) −
Start integrating a change

Resume Integration −
Resume an integration in progress

Cancel Integration (aeibu) −
Cancel an integration

View Differences (aediff) −
Show the file differences for the change

Build (aeb) −
Build the project

New Tests (aet) −
Run the new tests

Baseline Test (aet −bl) −
Run the baseline test

Regression Test (aet −reg) −
Run the regression tests

Pass (aeipass) −
Pass the integration

Fail (aeifail) −
Fail the integration

SEE ALSO

tkaeca(1)
GUI interface for the *aeca*(1) command.

tkaenc(1)
GUI interface for the *aenc*(1) command.

tkaepa(1)
GUI interface for the *aepa*(1) command.

COPYRIGHT

AUTHOR

Graham Wheeler <gram@cdsec.com>
Citadel Data Security

NAME

tkaenc − GUI interface for aenc, using TCL/TK

SYNOPSIS

tkaenc

DESCRIPTION

The *tkaenc* command is used to provide a GUI interface to *aenc*(1). Its use should be self-evident to anyone familiar with Aegis.

The top line of the screen contains button for selecting the project. It will be defaulted using the usual Aegis defaulting rules. Click on the buttons to obtain a pick list if you want to change it; then double-click an item to select it. (I would have used tk_optionMenu, but it doesn't have a scroll bar, even when all the items don't fit on the screen.)

The middle section contains text entry areas, for editing the *brief_description* and *description* fields of the change attributes. The text wraps in a natural way, both here and in (say) the "*tkaenc −list change_details*" listing, so only use newlines to indicate end-of-paragraph.

The lower section contains three boxes. The first is the testing required for the change − select as many or as few as is appropriate. The second a set of radio buttons to select the change cause − pick one.

The third box is whether to begin development immediately (it will also run the *aedb*(1) command), or whether to leave the change in *awaiting development*.

The "OK" and "Cancel" buttons do what you expect. The cancel button simply quits. The OK button runs the *aenc*(1) command − if anything goes wrong (*e.g.* asking for testing exemptions you can't have) then the error message will be displayed in the message area at the bottom of the window.

The number of the change will be printed on a message on the standard output, just as for a normal *aenc*(1) command.

OPTIONS

There are no command line options.

EXIT STATUS

The *tkaenc* command will exit with a status of 1 on any error. The *tkaenc* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aenc(1) Create new changes from the command line.

aedb(1) Begin development of changes from the command line.

tkaegis(1)
GUI interface for Aegis, using TCL/TK

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

tkaepa − GUI interface for aeca, using TCL/TK

SYNOPSIS

tkaepa

DESCRIPTION

The *tkaepa* command is used to provide a GUI interface to *aepa*(1). Its use should be self-evident to anyone familiar with project administration under Aegis.

The top line of the screen contains button for selecting the project The project will be defaulted whenever possible, using the usual Aegis. defaulting rules. Click on the buttons to obtain a pick list if you want to change; then double-click an item to select it.

There is a text entry areas for editing the *description* attribute. The lower section contains a number of checks boxes for the various attributes.

The “OK” and “Cancel” buttons do what you expect. The cancel button simply quits. The OK button runs the *aepa*(1) command − if anything goes wrong the error message will be displayed in the message area at the bottom of the window.

OPTIONS

There are no command line options. It is best to run this command in the background.

EXIT STATUS

The *tkaepa* command will exit with a status of 1 on any error. The *tkaepa* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file’s *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aepa(1) View and edit project attributes, from the command line.

tkaegis(1)
GUI interface for Aegis, using TCL/TK

COPYRIGHT

tkaepa version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

tkaer − GUI tool for reviewing Aegis change sets, using TCL/TK

SYNOPSIS

tkaer

DESCRIPTION

The *tkaer* command is used to provide an easy and convenient way to review Aegis change sets. It provides a front-end to other tools which are used to view the modifications.

Files in the change set are shown in one of four lists. The choice of list is based on the operation performed on the file by the change (create, modify, move or remove). Empty lists are not shown. The change details, as provided by *ael*(1) may be displayed by clicking on the “Details” button.

MODIFIED AND MOVED FILES

tkdiff is used to show the differenceces between the change and baseline versions of modified and moved files. (In the case of moved files, the original name is used to access the baseline version.)

Double-clicking button 1 on a filename (or pressing the space key when the filename is highlighted) will show the differences between the change and the current branch baseline. Holding down button 3 (or pressing the “a” key) will invoke a pop-up menu presenting the reviewer with a list of grandparent branches which also contain the file. This is particularly useful when rolling in branches.

NEW AND REMOVED FILES

New files are viewed by opening a new *xterm* and using *vi* (in read-only mode) to display its contents. This method is also used for removed files, however it is the baseline version that is displayed (so that the reviewer can see what has been removed).

ADDITIONAL FEATURES FOR REVIEWERS.

If the change is in the *being reviewed* state, the reviewer may open the comments editor by clicking on the “Comments” button. These comments will be submitted should the reviewer decide that the review has failed. An outline of the files included in the change is automatically created.

Once the review is complete, clicking the “Finished” button results in a dialog box which will allow the reviewer to pass or fail (via the *aerpass*(1) and *aerfail*(1) commands) the change. Alternatively, the reviewer may resume reviewing or quit, leaving the change state unmodified. If the change was not in the *being reviewed* state, the “Finished” button simply causes *tkaer* to exit.

CONFIGURING TKAER

tkaer may be customised by the .tkaer file. This file is created by *tkaer* in the users home directory when it is first run. This newly created file contains the default configuration as described above, such as the choice of tools used in reviewing. The configuration file itself is a tcl script which is executed by the *tkaer* script using the *tcl* “source” command. Each entry takes the form of a “set” statement which adds an item to the pref array. Items currently supported are:

pref(diff_command)

This is the tool used to visually display the difference between a changes modified or moved file and the baseline version. The default setting is

set pref(diff_command) "tkdiff"

You can change it to

set pref(diff_command) "mgdiff"

If you have the *mgdiff*(1) command installed.

pref(view_command)

This is the tool used to visually display a new file or a removed files, prior contents. The default is *vi*(1)

pref(view_edit_font)

This is the font used by both the change details viewer and the review comments editor. Any available X11 font may be used.

OPTIONS

−Change *number*

This option may be used to specify a particular change within a project. See *aegis*(1) for a complete description of this option.

−Project *name*

This option may be used to select the project of interest. When no **−Project** option is specified, the *AEGIS_PROJECT* environment variable is consulted. If that does not exist, the user's *$HOME/.aegisrc* file is examined for a default project field (see *aeuconf*(5) for more information). If that does not exist, when the user is only working on changes within a single project, the project name defaults to that project. Otherwise, it is an error.

EXIT STATUS

The *tkaer* command will exit with a status of 1 on any error. The *tkaer* command will only exit with a status of 0 if there are no errors.

ENVIRONMENT VARIABLES

See *aegis*(1) for a list of environment variables which may affect this command. See *aepconf*(5) for the project configuration file's *project_specific* field for how to set environment variables for all commands executed by Aegis.

SEE ALSO

aerpass(1)

pass review of a change

aerfail(1)

fail review of a change

tkaegis(1)

GUI interface for Aegis, using TCL/TK

tkdiff by John M. Klassa. TkDiff Home Page http://www.accurev.com/free/tkdiff

AUTHOR

tkaer contributed by Scott Finneran <sfinneran@lucent.com>

NAME

aecattr − aegis change attributes file

DESCRIPTION

A change attributes file is used to describe the modifiable portion of a change.

CONTENTS

A change attributes file contains the following fields:

description = string;
: This field contains a detailed description of the change.

brief_description = string;
: This field contains a brief description of the change.

cause = (...);
: This field describes the cause which motivated the change.

  external_bug
  : The change was created in response to a bug report from outside the development team. This repairs existing functionality.

  external_enhancement
  : The change was created in response to an enhancement request from outside the development team. This adds new functionality.

  external_improvement
  : The change was created in response to an improvement request from outside the development team. This improves existing functionality.

  internal_bug
  : The change was created in response to a bug report from inside the development team. This repairs existing functionality.

  internal_enhancement
  : The change was created in response to an enhancement request from inside the development team. This adds new functionality.

  internal_improvement
  : The change was created in response to an improvement request from inside the development team. This improves existing functionality.

  chain
  : This cause is where you have a fix to fix a fix; tracking these is an interesting quality metric.

test_exempt = boolean;
: This field is true if it is not necessary to test the change. It is, in general, desirable to test all changes, whether new functionality or a bug fix. This is, however, a project management issue.

test_baseline_exempt = boolean;
: This field is true if it is not necessary to test the change against the baseline before it is changed. The test of the baseline is required to fail; this is to establish that the test has isolated the bug, and that the change has fixed that isolated bug.

regression_test_exempt = boolean;
: This field is true if it is not necessary to perform a full regression test on the change. If absent, defaults to true for all causes except improvements.

architecture = [string];
: This field is a list of names of system and machine architectures on which the change must successfully build and test.

copyright_years = [integer];
: This field details the years in which the change was worked on. This field is present in trunk, branch and leaf nodes.

  As a change is edited, years in which the change was worked on accumulate in this field automatically. Branches accumulate years as integrations occur. You may need to manually edit this, though it should be rare.

version_previous = string;
: This field records the "previous" version, mostly to simplify patch generation. It is only meaningful for trunks and branches. It is set automatically when a branch is started or integrated.

attribute = [{ ... }];
: This is a list of *(name,value)* pairs, defining user specified attributes.

  name = string;
  : The name of the attribute. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not. Attribute names are case-insensitive.

  value = string;
  : The value of the attribute.

  Arguably, most change attributes which may be altered by the user (and some that can't) should be of this form. Due to an accident of history, this is not the case.

SEE ALSO

aenc(1) create a new change

aeca(1) modify the attributes of a change

aegis(5) aegis file format syntax

aecstate(5)
: change state file format

aepattr(5)
: project attributes file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aecstate − aegis change state file

SYNOPSIS

project`/info/change/`*[0−9]/[0−9][0−9][0−9]*

DESCRIPTION

A change state file is used to store information about a change. These files are created and maintained by aegis. These files should not be edited by humans. These files is owned by the project owner and group.

The change number is at least 3 digits, zero padded if necessary. (More digits will be used if a project has a thousand or more changes in any one release, although this is rare.) The files are spread across a directory tree, 100 per subdirectory, to improve the directory search times, and to avoid various systems' directory length limitations.

CONTENTS

description = string;

This field contains a detailed description of the change.

brief_description = string;

This field contains a brief description of the change.

cause = (...);

This field describes the cause which motivated the change.

external_bug

The change was created in response to a bug report from outside the development team. This repairs existing functionality.

external_enhancement

The change was created in response to an enhancement request from outside the development team. This adds new functionality.

external_improvement

The change was created in response to an improvement request from outside the development team. This improves existing functionality.

internal_bug

The change was created in response to a bug report from inside the development team. This repairs existing functionality.

internal_enhancement

The change was created in response to an enhancement request from inside the development team. This adds new functionality.

internal_improvement

The change was created in response to an improvement request from inside the development team. This improves existing functionality.

chain

This cause is where you have a fix to fix a fix; tracking these is an interesting quality metric.

test_exempt = boolean;

This field is true if it is not necessary to test the change. It is, in general, desirable to test all changes, whether new functionality or a bug fix. This is, however, a project management issue.

test_baseline_exempt = boolean;

This field is true if it is not necessary to test the change against the baseline before it is changed. The test of the baseline is required to fail; this is to establish that the test has isolated the bug, and that the change has fixed that isolated bug.

regression_test_exempt = boolean;
: This field is true if it is not necessary to perform a full regression test on the change. If absent, defaults to true for all causes except improvements.

architecture = [string];
: This field is a list of names of system and machine architectures on which the change must successfully build and test.

copyright_years = [integer];
: This field details the years in which the change was worked on. This field is present in trunk, branch and leaf nodes.

  As a change is edited, years in which the change was worked on accumulate in this field automatically. Branches accumulate years as integrations occur. You may need to manually edit this, though it should be rare.

version_previous = string;
: This field records the "previous" version, mostly to simplify patch generation. It is only meaningful for trunks and branches. It is set automatically when a branch is started or integrated.

attribute = [{ ... }];
: This is a list of *(name,value)* pairs, defining user specified attributes.

  name = string;
  : The name of the attribute. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not. Attribute names are case-insensitive.

  value = string;
  : The value of the attribute.

  Arguably, most change attributes which may be altered by the user (and some that can’t) should be of this form. Due to an accident of history, this is not the case.

state = (...);
: This field is used to describe what state the change is in. The state determines what operations may be performed on the change.

  awaiting_development
  : The change has been created, but has yet to be worked on.

  being_developed
  : The change is being developed.

  awaiting_review
  : The change has been developed, and is waiting to be review. (Optional, controlled by the *develop end action* project attribute.)

  being_reviewed
  : The change has been developed, and is being reviewed. (Optional, controlled by the *develop end action* project attribute.)

  awaiting_integration
  : The change has passed review, and is queued ready for integration.

  being_integrated
  : The change is being integrated.

  completed
  : The change has been completed and is now part of the baseline. Changes in this state can not be reversed.

given_test_exemption = boolean;
: This field is the value of test_exemption (see *aecattr*(5)) when the change was created.

given_regression_test_exemption = boolean;
: This field is the value of regression_test_exemption (see *aecattr*(5)) when the change was created.

delta_number = integer;
: This field records the delta number for this change. It is only present if the change is in one of the *being_integrated* or *completed* states.

delta_uuid = string;
: This field records a universally unique identifier for this configuration. It is supplements the *delta_number* field in that it is unique across all replicas of the project, whereas the delta number is ambiguous across replicas. It is only present in the *being_integrated* and *completed* states.

minimum_integration = boolean;
: This field records whether the change was placed into the *being_integrated* state using the −minimum option (or that option was implicitly set due to a file being removed). It is only present if the change is in the *being_integrated* state.

project_file_command_sync = integer;
: This field records the last change integrated into the project. If it disagrees with the project, a 'project_file_command' (from pconf) needs to be executed at the next build.

test_time = time;
: This field records the time the last successful *aegis −Test* command was run for all architectures. It is only present in the *being_developed* and *being_integrated* states.

test_baseline_time = time;
: This field records the time the last successful *aegis −Test −BaseLine* command was run for all architectures. It is only present in the *being_developed* and *being_integrated* states.

regression_test_time = time;
: This field records the time the last successful *aegis −Test −REGression* command was run for all architectures. It is only present in the *being_developed* and *being_integrated* states.

build_time = time;
: This field records the last time the last successful *aegis −Build* command was run for all architectures. It is only present in the *being_developed* and *being_integrated* states.

architecture_times = [{ ... }];
: This field records the time of various operations for each variant named in the *architecture* field. It is only present in the *being_developed* and *being_integrated* states.

  variant = string;
  : This field is one of the patterns named in the *architecture* field.

  node = string;
  : This field is the computer on which the command was run which last changed this structure.

  test_time = time;
  : This field records the last time the last successful *aegis −Test* command was run for this specific pattern instance.

  test_baseline_time = time;
  : This field records the last time the last successful *aegis −Test −BaseLine* command was run for this specific pattern instance.

  regression_test_time = time;
  : This field records the last time the last successful *aegis −Test −REGression* command was run for this specific pattern instance.

build_time = time;
This field records the last time the last successful *aegis −Build* command was run for this specific pattern instance.

development_directory = string;
This field is the absolute path of a change's development directory. It is only present of the change is in a state between *being_developed* and *being_integrated* inclusive.

However, branches are treated slightly differently to changes. The directory is relative to the root of the project tree, in order to facilitate moving the project without rewriting any of the database. Note that its doesn't point to the branch baseline, but one level up; just as the project root doesn't point to the trunk baseline, but rather one level up.

integration_directory = string;
This field is the absolute path of the change's integration directory. It is only present of the change is in the *being_integrated* state.

history = [{ ... }, ...];
This field records the history of the change, in the form of state transitions. The history records have the form

when = time;
This field records the time the state transition occurred.

what = (...);
This field records what happened. Valid value names echo the various aegis functions.

who = string;
This field records the user name of the user who caused the state transition.

why = string;
This field is optional. It is a comment of some sort. In the cases of *review_fail* and *integrate_fail*, this field will contain why the change failed.

uuid = string;
This field provides a globally unique identifier for the change set, even when geographically distributed development is happening.

branch = { ... };
This field is only present for branches (long transactions).

umask = integer;
File permission mode mask. See *umask*(2) for more information. This value will always be OR'ed with 022, because *aegis* is paranoid.

developer_may_review = boolean;
If this field is true, then a developer may review her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

Note that the *develop_end_action* field may not contradict the *developer_may_review* field. If developers may not review their own work, then their changes may not goto directly to the *being integrated* state (as this means much the same thing).

developer_may_integrate = boolean;
If this field is true, then a developer may integrate her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

reviewer_may_integrate = boolean;
If this field is true, then a reviewer may integrate a change she reviewed. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

developers_may_create_changes = boolean;
: This field is true if developers may created changes, in addition to administrators. This tends to be a very useful thing, since developers find most of the bugs.

forced_develop_begin_notify_command = string;
: This command is used to notify a developer that a change requires developing; it is issued when a project administrator uses an *aedb −User* command to force development of a change by a specific user. All of the substitutions described in *aesub*(5) are available. This field is optional.

  Executed as: the new developer. Current directory: the development directory of the change for the new developer. Exit status: ignored.

develop_end_notify_command = string;
: This command is used to notify that a change is ready for review. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

develop_end_undo_notify_command = string;
: This command is used to notify that a change had been withdrawn from review for further development. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

review_begin_notify_command = string;
: This command is used to notify that a review has begun. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_begin_undo_notify_command = string;
: This command is used to notify that a review is no longer in progress, the reviewer has withdrawn. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_notify_command = string;
: This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_undo_notify_command = string;
: This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. Defaults to the same action as the *develop_end_notify_command* field. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_fail_notify_command = string;
: This command is used to notify that a review has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

integrate_pass_notify_command = string;
: This command is used to notify that an integration has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Some compilers bury absolute path names into object files and executables. The renaming of the integration directory to become the new baseline breaks these paths. This command is passed an environment variable called AEGIS_INTEGRATION_-DIRECTORY so that the appropriate symlink may be placed, if desired.

  Executed as: the project owner. Current directory: the new project baseline. Exit status: ignored.

integrate_fail_notify_command = string;
: This command is used to notify that an integration has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the integrator. Current directory: the development directory of the change. Exit status: ignored.

default_test_exemption = boolean;
: This field contains what to do when a change is created with no test exemption specified.

default_test_regression_exemption = boolean;
: This field contains what to do when a change is created with no regression test exemption specified.

history = [{ ... }];
: This field contains a history of integrations for the project. Updated by each successful ’aegis −Integrate_Pass’ command.

  delta_number = integer;
  : The delta number of the integration.

change_number = integer;
: The number of the change which was integrated.

name = [string];
: The names by which this delta is known.

uuid = string;
: The uuid assigned to the change.

when = time;
: This field record the time of the change integration.

is_a_branch = (...)
: This field is used to remember if the completed change was a branch.

  unknown
  : It is unknown if the change is a branch, this is the default value usually associated with change integrated with an older version of aegis.

  no
  : The change is not a branch.

  yes
  : The change is a branch.

change = [integer];
: The list of changes which have been created on this branch to date.

sub_branch = [integer];
: The list of branches which have been created on this branch to date. This will be a subset of the above (possibly empty, possibly complete, never larger).

administrator = [string];
: The list of administrators of the branch.

developer = [string];
: The list of developers of the branch.

reviewer = [string];
: The list of reviewers of the branch.

integrator = [string];
: The list of integrators of the branch.

currently_integrating_change = integer;
: The change currently being integrated. Only one change (within a branch) may be integrated at a time. Only set when an integration is in progress.

default_development_directory = string;
: The pathname of where to place new development directories. The pathname must be absolute. This field is only consulted if the field of the same name in the user configuration file is not set.

minimum_change_number = integer;
: The minimum change number for *aenc(1)*, if no change number is specified. This allows the low-numbered change numbers to be used for branches later in the project. Defaults to 10 if not set, may not be less than 1.

reuse_change_numbers = boolean;
: This controls whether the automatically selected *aenc*(1) change numbers “fill in” any gaps. Defaults to true if not set.

minimum_branch_number = integer;
: The minimum branch number for *aenbr(1)*, if no branch number is specified. Defaults to 1 if not set.

skip_unlucky = boolean;
: This field may be set to true if you want to skip various unlucky numbers for changes, branches and tests. Various traditions are avoided, both Eastern and Western. Defaults to false if not set.

compress_database = boolean;
: This field may be set to true if you want to compress the database on writing. (It is always uncompress on reading if necessary.) Defaults to false if not set.

  Unless you have an exceptionally large project, coupled with fast CPUs and high network latency, there is probably very little benefit in using this feature. (The database is usually less than 5% of the size of the repository.) On slow networks, however, this can improve the performance of file-related commands.

develop_end_action = (...);
: This field controls the state the change enters after a successful *aede*(1) action.

  goto_being_reviewed
  : This means that the change goes from the *being_developed* state to the *being_reviewed* state. The *aerb*(1) command only sends informative email.

  goto_awaiting_review
  : This means that the change goes from the *being_developed* state to the *awaiting_review* state. The *aerb*(1) command is now mandatory.

  goto_awaiting_integration
  : This means that the change goes from the *being_developed* state into the *awaiting_integration* state. Code review is skipped entirely.

  Note that the *develop_end_action* field may not contradict the *developer_may_review* field. If developers may not review their own work, then their changes may not goto directly to the *being integrated* state (as this means much the same thing). A contradictory setting will be replaced with *goto_being_reviewed*.

Obsolete Fields

The following fields are only present is old projects. They will be moved to an appropriate file state when the change is next modified. See *aefstate*(5) for more information.

src = [{ ... }, ...];
: This field is a list of all the files in the change. The records have the form

  file_name = string;
  : This file names the file. The name is relative to the root of the baseline directory tree.

  uuid = string;
  : This field uniquely identifies the file for its entire lifetime. This field remains constant across file renames. The value of this field shall be formatted as a valid UUID, all in lower case.

  action = (create, modify, remove);
  : This field describes what is being done with the file.

edit_number = string;
: This field records the edit number of the file when it was added to the change (or updated using the *aegis −DIFFerence* command). This field is not present for new files.

usage = (source, config, build, test, manual_test);
: This field describes what function the file serves.

diff_time = time;
: This field records the last time modified of the change file when the last *aegis −DIFFerence* command was run. It is only present between the *being_developed* and *being_integrated* states, inclusive. It is not present for files which are being deleted. This field is used to determine if a difference has been done, and if the file has been tampered with before state transitions.

diff_file_time = time;
: This field records the last time modified of the difference file when the last *aegis −DIFFerence* command was run. It is only present between the *being_developed* and *being_integrated* states, inclusive. This field is used to determine if a difference has been done, and if the difference file has been tampered with before state transitions.

move = string;
: To change the name of a file, a combination of deleting the old name and creating the new name is used. With deleted files, this field is used to say where it went. With new files, this field is used to say where it came from.

WRITING REPORT SCRIPTS

When attempting to access these fields from within the report generator, you need a code fragment similar to the following:

```
auto ps;
ps = project[project_name()].state;
auto cs;
cs = ps.branch.change[change_number()];
```

All of the fields mentioned in the man page can now be accessed as members of the `cs` variable. For example, `cs.state` contains the state the change is in.

If this change state refers to a branch, when you access a member of the *branch.change* field, you are given access to the change state data of that change on the branch.

When you index the *src* field by a filename string, you may obtain access the the relevant file state (see *aefstate*(5) for more information).

SEE ALSO

aenc(1) create a new change

aegis(5) aegis file format syntax

aecattr(5)
: change attributes file format

aefstate(5)
: file state file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aedir − aegis directory structures

DESCRIPTION

The project directory structure is dictated by *aegis* at the top level, but is completely under the project's control from various points below the top level.

The project directory has the following contents

project`/`
- `baseline/`
  - `aegis.conf`
  - *...project specific...*
  - `test/`
    - *[0−9][0−9]*`/`
      - `t`*[0−9][0−9][0−9][0−9]*`a.sh`
      - `t`*[0−9][0−9][0−9][0−9]*`m.sh`
- `history/`
  - *...echo of baseline...*
- `delta.`*[0−9][0−9][0−9]*`/`
  - *...echo of baseline...*
- `info/`
  - `state`
  - `change/`
    - *[0−9]*`/`
      - *[0−9][0−9][0−9]*

The directory is structured in this way so that it is possible to pick an entire project up off the disk, and be confident that you got it all.

The location of the root of this tree is configurable, and may even be changed during the life of a project.

The contents of the *baseline* subdirectory, other than those given, are defined by the project, and not dictated by aegis.

The contents of the *delta.NNN* directory, when it exists, are an image of the *baseline* directory. It is frequently linked with the baseline, rather than a copy of it; see the *link_integration_directory* field description in *aepconf*(5) for more information.

The contents of the *history* contains the edit histories of the *baseline* directory, and is in all other ways an image of it. Note that *baseline* always contains the latest source; the *history* directory is just history. The actual files in the history directory tree will not always have names the same as those in the baseline; compare the methods used by SCCS and RCS.

The contents of the *baseline/test* directory are the tests which are created by changes. Test histories are also stored in the *history* subdirectory. Tests are treated as project source.

The edit histories are separated out to simplify the task of taking a "snapshot" of the source of a project, without airing all the dirty laundry.

The *baseline* directory always contains the latest source, and so the *history* directory need not be readily accessible, because the build mechanism (something like *make*(1), but preferably better) does not need to know anything about it. Similarly for tests.

The *baseline/aegis.conf* file is used to tell aegis everything else it needs to know about a project. See *aepconf*(5) for more information. This file is a source file of the project, and is treated in the same way as all source files. The name of this file is not mandatory.

SEE ALSO

aenc(1) create a new change

aenpr(1)
create a new project

aegis(5) aegis file format syntax

aepconf(5)
project configuration file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aefattr − aegis file attribute file format

SYNOPSIS

aefa −edit *filename*

DESCRIPTION

This file format is used to set or edit file attributes.

CONTENTS

attribute = [{ ... }];

This is a list of *(name,value)* pairs, defining user specified attributes.

name = string;

The name of the attribute. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not.

value = string;

The value of the attribute.

SEE ALSO

aegis(5) aegis file format syntax

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aefstate − aegis file state file

SYNOPSIS

project`/info/change/`*[0−9]/[0−9][0−9][0−9]*.fs

DESCRIPTION

A file state file is used to store information about the files in a transaction. These files are created and maintained by aegis. These files should not be edited by humans. These files is owned by the project owner and group.

CONTENTS

src = [{ ... }, ...];

This field is a list of all the files in the change. The records have the form

file_name = string;

This file names the file. The name is relative to the root of the baseline directory tree.

uuid = string;

This field uniquely identifies the file for its entire lifetime. This field remains constant across file renames. The value of this field shall be formatted as a valid UUID, all in lower case.

action = (create, modify, remove, insulate, transparent);

This field describes what is being done with the file.

create The file is being created. Once integrated, the edit fields record the file version created and stored in the history.

modify The file is being created. Once integrated, the edit fields record the file version stored in the history.

remove The file is being created. The edit field is only informational, and describes the file version at the time it was removed from the repository.

insulate The file is insulating a development directory from changes to the baseline, it shall be uncopied before development may end. This action shall only be present in changes. It shall never be present in branch change state files.

transparent

The file wasonce present in the branch, however it is desired that the ancestor version "show through". This is the equivalent of "uncopy" for branches. When the branch is integrated, this file will be omitted.

edit = { ... };

For a project or an active branch, this field records the head revision of the file. For a completed change or branch, this field records the revision number after integrate pass.

revision = string;

This is the edit number, as reported by the *history_get_command* in the project *config* file at integrate pass time.

encoding = (none, quoted_printable, base64);

This field records the encoding used when the file was added to the history at integrate pass time, as configured by one of the *history_put_command* or *history_get_command* and *history_content_limitation* fields of the project *config* file.

none No encoding was applied to the file. Either it had no binary characters, or the history tool is able to cope with binary files.

quoted_printable

The MIME Quoted Printable encoding (see RFC 1521) has been used to escape the binary characters of the file content.

base64 The MIME Base 64 encoding (see RFC 1521) has been used to encode the file content.

The *history_content_limitation* field of the project *config* file is used to determine which files need encoding. The size of the encoded file is compared to determine which of quoted printable and base 64 encodings is used; the smaller is chosen.

uuid = string;
This is the UUID of the change responsible for the edit.

edit_number = string;
This field is obsolescent. It is only present for backwards compatibility. It has been replaced by the *edit* field.

edit_origin = { ... };
This field records the edit number of the file when it was added to the change or branch. In changes, this field is not present for new files. (A change file is out of date if it's edit number_origin field does not equal the edit_number field in the project.)

It has the same fields, with the same meaning, as the *edit* field, above.

edit_number_origin = string;
This field is obsolescent. It is only present for backwards compatibility. It has been replaced by the *edit_origin* field.

edit_origin_new = { ... };
This field records the edit number of the file to replace the edit_number_origin field in the branch at integrate pass time. This is used to perform cross branch merging. This field cleared at integrate pass time.

It has the same fields, with the same meaning, as the *edit* field, above.

edit_number_origin_new = string;
This field is obsolescent. It is only present for backwards compatibility. It has been replaced by the *edit_origin_new* field.

usage = (source, config, build, test, manual_test);
This field describes what function the file serves.

file_fp = fingerprint;
This field records the last time modified of the source file. It is only present between the *being_developed* and *being_integrated* states, inclusive (for both changes and branches). It is not present for files which are being deleted. This field is used to determine if a difference has been done, or a test has been done if the source file is a test, and if the file has been tampered with before state transitions.

The fingerprint consists of the following fields:

youngest = time;
The youngest time see for this file with this fingerprint.

oldest = time;
The oldest time see for this file with this fingerprint.

crypto = string;
This field records a cryptographically strong fingerprint for the file. There is no known method of constructing a file to match a given fingerprint, and there is less than 1 in 2**200 chance that two files will have the same fingerprint. Thus if the fingerprint is the same, the file can reliably assumed to be the same.

diff_file_fp = fingerprint;
: This field records the last time modified of the difference file when the last *aegis −DIFFerence* command was run. It is only present between the *being_developed* and *being_integrated* states, inclusive (for both changes and branches). This field is used to determine if a difference has been done, and if the difference file has been tampered with before state transitions.

idiff_file_fp = fingerprint;
: This field records the last time modified of the integration difference file when the last *aegis −DIFFerence* command was run. It is only present in the *being_integrated* state. This field is used to determine if a difference has been done.

architecture_times = [{ ... }];
: This field records the time of various operations for each variant named in the *architecture* field. It is only present in the *being_developed* and *being_integrated* states. This field is used to determine if a test has been done, and thus optimize test runs.

  variant = string;
  : This field is one of the patterns named in the *architecture* field.

  test_time = time;
  : This field records the last time the last successful *aegis −Test* command was run for this specific pattern instance.

  test_baseline_time = time;
  : This field records the last time the last successful *aegis −Test −BaseLine* command was run for this specific pattern instance.

move = string;
: To change the name of a file, a combination of deleting the old name and creating the new name is used. With deleted files, this field is used to say where it went. With new files, this field is used to say where it came from.

locked_by = integer;
: The change which locked this file.
Caveat: this field is redundant, you can figure it out by scanning all of he change files. Having it here is very convenient, even though it means multiple updates.

about_to_be_created_by = integer;
: The change which is about to create this file for the first time. Same caveat as above.

about_to_be_copied_by = integer;
: For each change file that is acting on a project file from a deeper baseline than the immediate parent project's baseline, the file needs to be added to the immediate parent project. Note that this field says that this file record is a place marker, so that it can be deleted again should the change not be integrated for some reason.

deleted_by = integer;
: The change which last deleted this file. We never throw them away, because (a) it may be created again, and more important (b) we need it to recreate earlier deltas.

test = [string];
: This field is used to remember test correlations for source files. This is used by *aet*(1) to suggest suitable tests.

metrics = [{ ... }];
: This field is used to describe various file metrics. It is committed during *aeipass*(1), when the file is added to the history. The name must be given, and exactly one value.

name = string;
This is the name of the metric. This field must be set.

value = real;
This is the value of the metric. This field must be set. (If you have an integer-valued metric, just use integers, Aegis will cope. If you have a string-valued metric, assign integers to the enumerands.)

executable = boolean;
This field is used to remember whether the source file had any executable permission bits set at *develop end* time. This mode will be restored (taking the project umask into account) when the file is copied.

This field is only meaningful for changes in the *completed* state, because this field is only set by *aeip*(1). Until then, the mode if the file itself is the authority.

attribute = [{ ... }];
This is a list of *(name,value)* pairs, defining user specified attributes.

name = string;
The name of the attribute. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not. Attribute names are case-insensitive.

value = string;
The value of the attribute.

Arguably, most file properties which may be altered by the user (and some that can't) should be of this form. Due to an accident of history, this is not the case.

WRITING REPORT SCRIPTS

When attempting to access these fields from within the report generator, you need a code fragment similar to the following:

```
auto ps, pfs;
ps = project[project_name()].state;
fps = ps.src["somefile"];

auto cs, cfs;
cs = ps.branch.change[change_number()];
cfs = cs.src["somefile"];
```

Notice that the top-level fields of the file state are not available, but instead are mapped onto the relevant project file and change file *src* arrays.

All of the src member fields mentioned in the man page can now be accessed as members of the `pfs` or `cfs` variables.

SEE ALSO

aegis(5) aegis file format syntax

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis − meta-data file format

DESCRIPTION

The files used by the *aegis* program all have the same format. Some of the files used by aegis are created and maintained by humans, and some are created an maintained by aegis itself. The various manual entries say which is which.

LEXICAL CONSIDERATIONS

Names are any C identifier. Comments are C-style comments (or C++ or shell). Numbers are decimal, octal or hexadecimal, as for C constants. Whitespace (spaces, tabs and newlines) are ignored except in strings or as they serve to separate tokens.

Strings are C-style strings, and similar to C, sequential string constants are silently catenated together.

In addition, there is a style of `@string@` which use at-signs (@) for quoting. Unlike the C style of string, newlines are allowed within these strings. To get an at-sign in such a string, double the at-sign. There is no other escape mechanism available.

GRAMMAR

The format of all aegis files is described by a *yacc (1)* grammar.

```
%%
file
    : field_list
    ;
```

A file contains a field list.

```
field_list
    : /* empty */
    | field_list field
    ;
```

A field list is zero or more fields.

```
field
    : NAME '=' value ';'
    ;
```

A field is set by giving a name and a value.

```
value
    : NAME
    | INTEGER
    | STRING
    | structure
    | list
    ;
```

A value may be a member of an enumeration (NAME), or an integer constant, or a literal string. More complex values may be constructed from these simple values.

```
structure
    : '{' field_list '}'
    ;
```

A structure is a grouped list of fields.

```
list
    : '[' list_body ']'
    ;
list_body
    : /* empty */
    | value_list
    | value_list ','
    ;
value_list
```

```
        : value
        | value_list ',' value
```

A list is a sequential list of values separated by commas. It may be empty, or it may have a trailing comma.

SEMANTICS

The types of the values must match those in the definition of the file. See the relevant man pages for more information.

Files which are rewritten by **aegis** will lose any comments placed in them. When time fields are emitted by **aegis** they are usually followed by a human readable date in a comment.

SEE ALSO

aegis(1) a project change supervisor

aecattr(5)
change attribute file format

aecstate(5)
change state file format

aedir(5) directory structures

aegstate(5)
aegis state file

aepattr(5)
project attribute file format

aepconf(5)
project configuration file format

aepstate(5)
project state file format

aesub(5)
available command substitutions

aeuconf(5)
user configuration file format

aeustate(5)
user state file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegstate − aegis global state file

SYNOPSIS

`/usr/local/com/state`

DESCRIPTION

The aegis state file is used to store the pointers to project directories.

CONTENTS

where = [{ ... }];

This field is a table relating project name to project directory. The structure is as follows:

project_name = string;

The name of a project.

directory = string;

Absolute path of the project's directory. (Only set of *alias_for* is not set.)

alias_for = string;

This is the name of another project, possibly including branch numbers. It allows you to have shorter or more meaningful project names. (Only set if *directory* is not set.)

WRITING REPORT SCRIPTS

When attempting to access these fields from within the report generator, you need a code fragment similar to the following:

```
auto p;
p = project[project_name()];
```

That is, the `where` field is represented by the `project` array variable, however, it does not mention the aliases, only the actual projects, similar to the "*ael projects*" command. (You can, however, index the `projects` array by an alias, or even by a project name with branches on the end.)

In addition to the *project_name* and *directory* fields specified above, the report generator insets a *state* field, which gives you access to the project state fields (see *aepstate*(5) for more information).

SEE ALSO

aegis(5) aegis file format syntax

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aegis locks − how locking works, and which commands use them

DESCRIPTION

Aegis maintains a database of information about the projects in its care, and the various changes, both completed and in progress. In order to ensure the integrity of this database, and also your project repository, it uses locks.

From time to time, these locks are visible to the users, because they will be told that a command is waiting for a particular lock. For some transactions, this can be a long wait.

Dining Philosophers

While UNIX supplies locks in various flavors, if you need an entire set of locks simultaneously, there is no elegant “all or nothing ” interface available. This is unsurprising, as this is one of the classic computer science problems, known as the Dining Philosophers problem.

The master lock is used to solve the Dining Philosophers problem, and is meant to be very transient. It is only held while the other locks which are required (frequently two or more, hence the problem) are requested − non-blocking. Once they are all obtained (or not, and any partials given back) the master lock is released. It is usually held for *much* less than a second. If you notice the master lock being held, it is almost always a symptom of the NFS lock daemon misbehaving.

If the lock(s) could not be obtained, the blocking lock is waited on (without the master). This is when the "waiting for" message is issued. When obtained, it is *released* and the whole cycle starts again. This is why you occasionally see a series of "waiting for" messages. (This could maybe be optimized some, but it is still possible to block on yet another lock, and they you have to release all and wait again. As yet, I’m not convinced the extra code complexity is required.)

Listing Locks

There is a command available to list the current Aegis locks.

```
aegis −list locks
```

Note that the project names are change numbers are *guesses* as the locks are hashed over a 16-bit range, and range overlaps are possible. Collisions are also possible, but fortunately rarer.

Known Problems

There is a known problem with the HP/UX NFS clients. If you see persistent “no locks available” error messages when */usr/local/lib* is NFS mounted, try making the */usr/local/lib/lockfile* file world writable. chmod 666 /usr/local/lib/lockfile There is the possibility of a denial of service attack (which is why the default is 0600) but since you are presently denied service anyway, it’s academic.

COMMANDS

The following table shows the locks taken by the various commands. Note that theoretically some of the commands take too *few* locks, but this has yet to prove to be a problem in practice. Also, "project state file" and "change state file" are the same thing for branches, it just depends which way you are looking at them at the time.

Command	Global State File	Project State File	Project Baseline	Ancestor Baselines	Change State File	User State File
aeb (dev)	.	.	shared	shared	exclusive	.
aeb (int)	.	.	.	shared	exclusive	.
aeca	.	.	.	.	exclusive	.
aechown	.	.	.	.	exclusive	exclusive
aeclean	.	.	.	.	exclusive	.
aeclone	.	exclusive	.	.	exclusive	exclusive
aecp	.	,	,	,	exclusive	.
aecpu	.	.	.	.	exclusive	.
aed	.	,	,	,	exclusive	.
aedb	.	.	.	.	exclusive	exclusive
aedbu	.	.	.	.	exclusive	exclusive

aede	.	,	.	.	exclusive	exclusive
aedeu	.	.	.	.	exclusive	exclusive
aedn	.	exclusive	.	.	.	.
aeib	.	exclusive	.	.	exclusive	exclusive
aeibu	.	exclusive	.	.	exclusive	exclusive
aeifail	.	exclusive	.	.	exclusive	exclusive
aeipass	.	exclusive	exclusive	.	exclusive	exclusive
aemv	.	,	,	,	exclusive	.
aena	.	exclusive	.	.	.	.
aenbr	.	exclusive	.	.	.	.
aenbru	exclusive	exclusive	.	.	.	.
aenc	.	exclusive	.	.	.	.
aencu	.	exclusive	.	.	.	.
aend	.	exclusive	.	.	.	.
aenf	.	.	.	.	exclusive	.
aenfu	.	.	.	.	exclusive	.
aeni	.	exclusive	.	.	.	.
aenpa	exclusive	.	.	.	.	.
aenpr	exclusive	.	.	.	.	.
aenrv	.	exclusive	.	.	.	.
aent	.	exclusive	.	.	exclusive	.
aentu	.	.	.	.	exclusive	.
aepa	.	exclusive	.	.	.	.
aera	.	exclusive	.	.	.	.
aerd	.	exclusive	.	.	.	.
aerfail	.	.	.	.	exclusive	exclusive
aeri	.	exclusive	.	.	.	.
aerm	.	,	.	.	exclusive	.
aermu	.	.	.	.	exclusive	.
aermpr	exclusive	.	.	.	.	.
aerpa	exclusive	.	.	.	.	.
aerpass	.	.	.	.	exclusive	.
aerpu	.	.	.	.	exclusive	.
aerrv	.	exclusive	.	.	.	.
aet	.	.	,	,	exclusive	.

SEE ALSO

You may wish to see the manual pages of all of the above commands. Many have descriptions of the locking interactions.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aemetrics − metrics values file format

SYNOPSIS

filename,S

DESCRIPTION

Metrics files are created at integration build time, and recorded into the file attributes at integration pass time. This allows trend analysis and other statistics to be calculated.

CONTENTS

metrics = [{ ... }];

This field is used to describe various file metrics. It is committed during *aeipass*(1), when the file is added to the history. The name must be given, and exactly one value.

name = string;

This is the name of the metric. This field must be set.

value = real;

This is the value of the metric. This field must be set. (If you have an integer-valued metric, just use integers, Aegis will cope. If you have a string-valued metric, assign integers to the enumerands.)

SEE ALSO

aegis(5) aegis file format syntax

aemeasure(1)

simple file metrics

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aepattr − aegis project attribute file

DESCRIPTION

The project attribute file is used to store modifiable information about a project.

CONTENTS

description = string;

This field contains a description of the project. Large amounts of prose are not required; a single line is sufficient.

developer_may_review = boolean;

If this field is true, then a developer may review her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

Note that the *develop_end_action* field may not contradict the *developer_may_review* field. If developers may not review their own work, then their changes may not goto directly to the *being integrated* state (as this means much the same thing).

developer_may_integrate = boolean;

If this field is true, then a developer may integrate her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

reviewer_may_integrate = boolean;

If this field is true, then a reviewer may integrate a change she reviewed. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

developers_may_create_changes = boolean;

This field is true if developers may created changes, in addition to administrators. This tends to be a very useful thing, since developers find most of the bugs.

forced_develop_begin_notify_command = string;

This command is used to notify a developer that a change requires developing; it is issued when a project administrator uses an *aedb −User* command to force development of a change by a specific user. All of the substitutions described in *aesub*(5) are available. This field is optional.

Executed as: the new developer. Current directory: the development directory of the change for the new developer. Exit status: ignored.

develop_end_notify_command = string;

This command is used to notify that a change is ready for review. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

develop_end_undo_notify_command = string;

This command is used to notify that a change had been withdrawn from review for further development. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

review_begin_notify_command = string;

This command is used to notify that a review has begun. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_begin_undo_notify_command = string;

This command is used to notify that a review is no longer in progress, the reviewer has withdrawn. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_notify_command = string;

This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_undo_notify_command = string;

This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. Defaults to the same action as the *develop_end_notify_command* field. All of the substitutions described by *aesub*(5) are available.

Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_fail_notify_command = string;

This command is used to notify that a review has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

integrate_pass_notify_command = string;

This command is used to notify that an integration has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Some compilers bury absolute path names into object files and executables. The renaming of the integration directory to become the new baseline breaks these paths. This command is passed an environment variable called AEGIS_INTEGRATION_DIRECTORY so that the appropriate symlink may be placed, if desired.

Executed as: the project owner. Current directory: the new project baseline. Exit status: ignored.

integrate_fail_notify_command = string;
: This command is used to notify that an integration has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the integrator. Current directory: the development directory of the change. Exit status: ignored.

default_development_directory = string;
: The pathname of where to place new development directories. The pathname must be absolute. This field is only consulted if the field of the same name in the user configuration file is not set.

umask = integer;
: File permission mode mask. See *umask*(2) for more information. This value will always be OR'ed with 022, because *aegis* is paranoid.

default_test_exemption = boolean;
: This field contains what to do when a change is created with no test exemption specified.

default_test_regression_exemption = boolean;
: This field contains what to do when a change is created with no regression test exemption specified.

minimum_change_number = integer;
: The minimum change number for *aenc(1)*, if no change number is specified. This allows the low-numbered change numbers to be used for branches later in the project.

reuse_change_numbers = boolean;
: This controls whether the automatically selected *aenc*(1) change numbers "fill in" any gaps. Defaults to true if not set.

minimum_branch_number = integer;
: The minimum branch number for *aenbr(1)*, if no branch number is specified. Defaults to 1 if not set.

skip_unlucky = boolean;
: This field may be set to true if you want to skip various unlucky numbers for changes, branches and tests. Various traditions are avoided, both Eastern and Western. Defaults to false if not set.

compress_database = boolean;
: This field may be set to true if you want to compress the database on writing. (It is always uncompressed on reading if necessary.) Defaults to false if not set.

  Unless you have an exceptionally large project, coupled with fast CPUs and high network latency, there is probably very little benefit in using this feature. (The database is usually less than 5% of the size of the repository.) On slow networks, however, this can improve the performance of file-related commands.

develop_end_action = (...);
: This field controls the state the change enters after a successful *aede*(1) action.

  goto_being_reviewed
  : This means that the change goes from the *being_developed* state to the *being_reviewed* state. The *aerb*(1) command only sends informative email.

  goto_awaiting_review
  : This means that the change goes from the *being_developed* state to the *awaiting_review* state. The *aerb*(1) command is now mandatory.

  goto_awaiting_integration
  : This means that the change goes from the *being_developed* state into the *awaiting_integration* state. Code review is skipped entirely. If the *developer_may_review* is

false, it is not possible to use this setting.

Note that the *develop_end_action* field may not contradict the *developer_may_review* field. If developers may not review their own work, then their changes may not goto directly to the *being integrated* state (as this means much the same thing). A contradictory setting will be replaced with *goto_being_reviewed*.

protect_development_directory = boolean;

This field may be used to protect the development directory after the *being developed* state. It does this by making it read-only at develop end time. Should the change ever be returned to the *being developed* state, it will be made writable again.

The default is false, meaning to leave the development directory writable while is being reviewed and integrated. Aegis' normal tampering detection will notice if files are changed, but there is no reminder to the developer that the change should be left alone.

This field defaults to false, because it can sometimes be slow.

SEE ALSO

aepa(1) modify the attributes of a project

aegis(5) aegis file format syntax

aecattr(5)
change attributes file format

aecstate(5)
change state file format, particularly as branches are used to remember most project state

aepstate(5)
project state file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aepconf − aegis project configuration file

SYNOPSIS

project`/baseline/aegis.conf` *(default)*
project`/baseline/config` *(obsolete)*

DESCRIPTION

A project configuration file is used to store information about a project. This file is under source control, and is one of the project's source files. Developers may thus modify this file as part of a change.

As of aegis.4.17, it is possible to assign any arbitrary name to the project configuration file or files. See *aenf*(1) for more information.

This file contains a number of commands to be executed by Aegis. There are times when the substitutions in these commands may contain shell special characters, which would change the meaning of the commands in unintended ways. There are two main sources of these problems: file names and architecture names. In order to have shell special characters in filenames, you must set the *shell_safe_filenames* field (see below) to *false*. If you do this, you will need to use the `quote` substitution (see *aesub*(5)) to quote them, so that the shell does not abuse them. Other things which may need quoting include architecture names if you get creative, and edit numbers if unusual ones are generated by your history tool.

Getting Started

Because the project *aegis.conf* file is under source control like any other file, you must create the project *aegis.conf* file in the very first change of your project. Use the

```
$ aenf aegis.conf
$
```

command and then editing the file to fill in the fields. Subsequent Aegis commands in that change will use that file. Once the change is completed (see *aeipass*(1) for more information) the file will be present in the baseline, and be used by all users and all changes.

If you ever need to change one of the fields of the project *aegis.conf* file, you do this the same way as for any other source file, by copying it into a change using the

```
$ aecp aegis.conf
$
```

command and then edit the file to make the desired changes. While it's *being developed* your change will use it's copy of the project *aegis.conf* file, but once the change is completed (see *aeipass*(1) for more information), it becomes the new version used by all users and changes.

If you would prefer a different name for the project configuration file, use the *aenf −config* option. For example, the

```
$ aenf -config project.configuration
$
```

command would create a file called *project.configuration* and Aegis would then proceed to use it to obtain project configuration information for the duration of the project. This attribute will even be preserved across file renames (see the *aemv*(1) command).

CONTENTS

This file contains the following fields:

configuration_directory = string;

> This field names a directory which will be searched for additional configuration files. (This directive is only legal or meaningful in the master project *aegis.conf* file.)
>
> All source files (change source files and project source files) present in this directory will be read in as if they were added to the end of the project "aegis.conf" file.
>
> The usual priority of files (development directory, branch baseline, *etc*, project trunk baseline) is observed when these files are read.
>
> Please note that the physical directories are never searched, only the Aegis concept of the change and project files is consulted (*i.e.* files created and modified in the usual way with *aenf*(1) and

aecp(1) commands). Placing additional files in the physical directories will have no effect.

It is recommended that if you use this field at all, that your top level project *aegis.conf* file should only contain this one field. This is to avoid overly-large re-reading of this file when it is joined to all the others.

build_command = string;

This field describes how to build the project (actually, how to do an integration build). This field is mandatory. Used by the *aeb*(1) command. All of the substitutions described by *aesub*(5) are available.

Executed as: the integrator (for integration builds) or the developer (for development builds). Current directory: the integration directory of the change (for integration builds) the development directory of the change (for development builds). Exit status: zero is considered success, non-zero is a failure and a subsequent successful (exit zero) build will be required.

If this field is set to `"exit 0"` then no integration build will be required, and will not be checked for by the *aeipass*(1) command.

development_build_command = string;

This field describes how to do a development build. If this field is absent, it defaults to the above. Used by the *aeb*(1) command. All of the substitutions described by *aesub*(5) are available.

Executed as: the developer. Current directory: the development directory of the change. Exit status: zero is considered success, non-zero is a failure and a subsequent successful (exit zero) build will be required.

If this field is set to `"exit 0"` then no development build will be required, and will not be checked for by the *aede*(1) command.

development_directory_style = { ... };

This field encapsulates a set of parameters controlling the appearance of the development directory. It has significant implications for the way the DMT is used, and the directory appearance presented to the DMT.

source_file_link = boolean;

This field is true if hard links are to be used for project source files (which are not part of the change) so that the work area has a complete set of source files.

Defaults to false if not set.

If the host system does not have hard links, this field will be ignored.

Maintaining the hard links can be time consuming for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid links.

source_file_symlink = boolean;

This field is true if symbolic links are to be used for project source files (which are not part of the change) so that the work area has a complete set of source files.

Defaults to false if not set. [If the obsolete *create_symlinks_before_build* field is set, defaults to the value of that field, with a warning.]

If (*source_file_link* == true **and** hard links are available) this field will be ignored. If the host system does not have symbolic links, this field will be ignored.

Maintaining the symbolic links can be time consuming for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid symbolic links.

source_file_copy = boolean;

This field is true if copies are to be used for project source files (which are not part of the change) so that the work area has a complete set of source files. File modification time attributes will be preserved.

Defaults to false if not set.

If ((*source_file_link* == true **and** hard links are available) OR (*source_file_symlink* == true **and** symbolic links are available)) this field will be ignored.

Maintaining the copies can be time consuming (and space consuming) for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid file copies.

source_file_whiteout = boolean;

The *source_file_whiteout* field mat be used to specify the presence (true) or absence (false) of white-out files, used to "cover up" files being removed by a change set. These files contain 1kB of random data, intended to cause a syntax error should be build reference them.

It is rarely necessary to explicitly set this field. It defaults to false if you set any of the *source_file_link*, *source_file_symlink* or *source_file_copy* to true; it defaults to true only if none of them are true.

Not meaningful (always false) for integration builds.

derived_file_link = boolean;

This field is true if hard links are to be used for non-source files which are present in the project baseline(s) but which are not present in the work area, so that the work area has a complete set of derived files. This allows work areas to take advantage of "precompiled" object files (*etc*) in the baseline(s).

Defaults to false if not set.

If the host system does not have hard links, this field will be ignored.

Maintaining the links can be time consuming for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid hard links. Alternatively, set *derived_at_start_only = true;* and your work area will get a "head start" but the derived files will not be checked for every build, but this will occasionally result in long build times after integrations.

See also the *integrate_begin_exceptions* and *symlink_exceptions* fields (they apply to hard links as well as symbolic links).

derived_file_symlink = boolean;

This field is true if symbolic links are to be used for non-source files which are present in the project baseline(s) but which are not present in the work area, so that the work area has a complete set of derived files. This allows work areas to take advantage of "precompiled" object files (etc) in the baseline(s).

Defaults to false if not set. [If the obsolete *create_symlinks_before_build* field is set, defaults to the value of that field, with a warning.]

If (*derived_file_link* == true **and** hard links are available) this field will be ignored. If the host system does not have symbolic links, this field will be ignored.

Maintaining the symbolic links can be time consuming for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid symbolic links. Alternatively, set *derived_at_start_only = true;* and your work area will get a "head start" but the derived files will not be checked for every build, occasionally resulting in long build times after integrations.

See also the *integrate_begin_exceptions* and *symlink_exceptions* fields.

derived_file_copy = boolean;

This field is true if copies are to be used for non-source files which are present in the project baseline(s) but which are not present in the work area, so that the work area has a complete set of derived files. This allows work areas to take advantage of "precompiled" object files (*etc*) in the baseline(s).

Defaults to false if not set.

If ((*derived_file_link* == true **and** hard links are available) **or** (*derived_file_symlink* == true **and** symbolic links are available)) this field will be ignored.

Maintaining the copies can be time consuming (and space consuming) for large projects, and add quite a noticeable delay before builds start doing anything. If possible, change your build system to use the *$search_path* substitution instead and avoid symbolic links. Alternatively, set *derived_at_start_only = true;* and your work area will get a "head start" but the derived files will not be checked for every build, occasionally resulting in long build times after integrations.

See also the integrate_begin_exceptions and symlink_exceptions fields (they apply to copies as well as symbolic links).

during_build_only = boolean;

This field is set to true if you want the symbolic links, hard links and/or copies removed again after each build. This allows the user to maintain the illusion of using a search path, without actually doing so. This option is not especially efficient.

Defaults to false if not set. [If the obsolete *remove_symlinks_after_build* field is set, defaults to the value of that field, with a warning.]

If this field is false, the development directory will be populated by the develop begin (*aedb*) command, and the integration directory will be populated by the integrate begin (*aeib*) command.

derived_at_start_only = boolean;

This field controls whether the above fields controlling the appearance of derived files are acted upon before every build (false) or only when the work area is created (true).

Defaults to false if not set.

This field is ignored if the *during_build_only* field is true.

This field can be complex. Here are a few examples; but much, much more is possible. The first example will get you a development directory very similar to one presented by CVS:

```
development_directory_style =
{
  source_file_copy = true;
};
```

Note that this is hugely space inefficient, and can be quite slow. The second example will get you a development directory very similar to one presented by Tom Lord’s *arch*:

```
development_directory_style =
{
  source_file_link = true;
  source_file_symlink = true;
  source_file_copy = true;
};
```

Ideally, however, you should use the *$search_path* substitution of the *build_command* field. This is because the view path scales better than any other method. On the other hand, you need a DMT with an excellent view path implementation (and GNU *make* doesn’t).

The development directory style is applied *after* the `develop_begin_command` hook is run.

integration_directory_style = { ... };

This field encapsulates a set of parameters controlling the appearance of the integration directory. It has significant implications for the way the DMT is used, and the directory appearance presented to the DMT.

Defaults to the value of the *development_directory_style* field if not set. Note that the obsolete *create_symlinks_before_integration_build* and *remove_symlinks_after_integration_build* fields affect this default (with a warning) but only if they are *explicitly* set.

Note that the *link_integration_directory* field is still relevant. That field controls how the baseline is cloned to form the integration directory. This field operates after that operation.

build_time_adjust_notify_command = string;

This command is run when Aegis adjusts the last-time-modified time-stamp on files in the integration directory. If the build tool uses additional information to supplement file modification times, this command gives you the opportunity to re-sync the associated database.

Executed as: the project owner.

Current directory: the integration directory. This is what is about to be come the new baseline.

Exit status: NOT ignored. Note that a failure here puts the change in a partial state from which recovery may be difficult. Best to define this command with a *set*+e so that errors are ignored at the command level.

build_covers_all_architectures = boolean;

This field is set to true if the build command, when executed on any architecture, results in all architectures being built. This may be accomplished, for example, by using cross-compilation techniques, or Cook's ability to nominate hosts on which to execute each build rule.

test_covers_all_architectures = boolean;

This field is set to true if the test command, when executed on any architecture, results in all architectures being tested. This may be accomplished, for example, by using Cook's ability to nominate hosts on which to execute each test rule.

symlink_exceptions = [string];

This field is used to list filename patterns for which symbolic links must not be made between the development directory and the baseline. These are usually state files for various tools. The patterns are matched against the whole filename; naming only the last filename path element will *not* work (unless the pattern starts with "*").

change_file_command = string;

This field contains a command to be executed whenever a ´aegis −CoPy_file´, ´aegis −New_File´ ´aegis −New_Test´ ´aegis −MoVe_file´ or ´aegis −ReMove_file´ command is successful. See also command-specific overrides. If this field is absent, nothing is done. Used by the *aecp*(1), *aenv*(1), *aenf*(1), *aerm*(1), and *aemv*(1) commands. All of the substitutions described by *aesub*(5) are available; in addition,

${File_List}

Space separated list of files named.

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

change_file_undo_command = string;

This field contains a command to be executed whenever a ´aegis −CoPy_file_Undo', ´aegis −MoVe_file_Undo' ´aegis −New_File_Undo', ´aegis −New_Test_Undo', or ´aegis −ReMove_file_Undo' command is successful. Default to *change_file_command* if absent. See also command-specific overrides. If both fields are absent, nothing is done. Used by the *aecpu*(1), *aemvu*(1), *aenfu*(1), *aentu*(1) or *aermu*(1), commands. All of the substitutions described by *aesub*(5) are available; in addition,

${File_List}
Space separated list of files named.

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

new_file_command = string;
Executed whenever the aegis −new_file command is run successfully. Defaults to ‘change_file_command’ if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_List}
Space separated list of files named (at times, can be empty).

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

new_test_command = string;
Executed whenever the aegis −new_test command is run successfully. Defaults to ‘change_file_command’ if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_List}
Space separated list of files named (at times, can be empty).

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

copy_file_command = string;
Executed whenever the aegis −copy_file command is run successfully. Defaults to ‘change_file_command’ if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_List}
Space separated list of files named (at times, can be empty).

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

remove_file_command = string;
Executed whenever the aegis −remove_file command is run successfully. Defaults to ‘change_file_command’ if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_List}
Space separated list of files named (at times, can be empty).

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

new_file_undo_command = string;
Executed whenever the aegis −new_file_undo command is run successfully. Defaults to change_file_undo_command if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_List}
Space separated list of files named (at times, can be empty).

Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

new_test_undo_command = string;
: Executed whenever the aegis −new_test_undo command is run successfully. Defaults to change_file_undo_command if not set.

  All of the substitutions described in *aesub*(5) are available. In addition:

  ${File_List}
  : Space separated list of files named (at times, can be empty).

  Executed as: the developer Current directory: the development directory of the change Exit status: ignored

copy_file_undo_command = string;
: Executed whenever the aegis −copy_file_undo command is run successfully. Defaults to change_file_undo_command if not set.

  All of the substitutions described in *aesub*(5) are available. In addition:

  ${File_List}
  : Space separated list of files named (at times, can be empty).

  Executed as: the developer Current directory: the development directory of the change Exit status: ignored

remove_file_undo_command = string;
: Executed whenever the aegis −remove_file_undo command is run successfully. Defaults to change_file_undo_command if not set.

  All of the substitutions described in *aesub*(5) are available. In addition:

  ${File_List}
  : Space separated list of files named (at times, can be empty).

  Executed as: the developer Current directory: the development directory of the change Exit status: ignored

make_transparent_command = string;
: The make_transparent_command is executed whenever the aegis −make_transparent command is run successfully. Defaults to change_file_command if not set.

  All of the substitutions described in *aesub*(5) are available. In addition:

  ${File_List}
  : Space separated list of files named (at times, can be empty).

  Executed as: the developer Current directory: the development directory of the change Exit status: ignored

make_transparent_undo_command = string;
: The make_transparent_undo_command is executed whenever the aegis −make_transparent_undo command is run successfully. Defaults to change_file_undo_command if not set.

  All of the substitutions described in *aesub*(5) are available. In addition:

  ${File_List}
  : Space separated list of files named (at times, can be empty).

  Executed as: the developer Current directory: the development directory of the change Exit status: ignored

project_file_command = string;
: This field contains a command to be executed during a development build before the *development build command* above, when (a) it is the first build after a develop begin, or (b) some other change has been integrated into the baseline since the last build. If this field is absent, nothing is done. Used by the *aeb*(1) command. All of the substitutions described by *aesub*(5) are available.

develop_begin_early_command = string;
: This field contains a command to be executed at the beginning of a ’aegis −Develop_Begin’ command, immediately after the development directory has been created. If this field is absent, nothing is done. Used by the *aedb*(1) command. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

  Note: This command is run from inside the lock, so running *any* Aegis command that modifies the change state will cause a deadlock.

develop_begin_command = string;
: This field contains a command to be executed whenever a ’aegis −Develop_Begin’ command is successful. If this field is absent, nothing is done. Used by the *aedb*(1) command. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

develop_begin_undo_command = string;
: This field contains a command to be executed whenever a ’aegis −Develop_Begin_Undo’ command is successful. If this field is absent, nothing is done. Used by the *aedbu*(1) command. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: wherever the command was executed from. Exit status: ignored.

integrate_begin_command = string;
: This field contains a command to be executed whenever a ’aegis −Integrate_Begin’ command is successful. If this field is absent, nothing is done. Used by the *aeib*(1) command. All of the substitutions described by *aesub*(5) are available.

  Executed as: the project owner. Current directory: the integration directory. Exit status: ignored.

link_integration_directory = boolean;
: This flag is true if Aegis should link the files from the baseline into the integration directory, rather than copy them (the default). This has risks, as the build script (e.g. *Howto.cook* or *Makefile*, etc) must unlink targets before rebuilding them; if this is not done the baseline will be corrupted. Used by the *aeib*(1) command.

integrate_begin_exceptions = [string];
: This field may be used to specify a list of file names (and file name patterns) which are to be omitted from the copy (link) of the baseline when creating the integration directory. Used by the *aeib*(1) command. This field only applies to derived files, it does *not* apply to source files. The patterns are matched against the whole filename; naming only the last filename path element will *not* work (unless the pattern starts with “*”).

history_create_command = string;
: This field is used to create a new history. The command is always executed as the project owner. Used by the *aeipass*(1) command.

  It is strongly recommended that the *history_create_command* and *history_put_command* fields are identical. If not set, the *history_create_command* field defaults to the same value as the *history_put_command* field.

  All of the substitutions described by *aesub*(5) are available; in addition,

  ${Input}
  : Absolute path of the source file.

${History}
Absolute path of the history file. This may need to be reworked with the *Dirname* and *Basename* substitutions to yield a string suitable for the history tool in question.

${File_Name}
The base relative file name of the file for this check-in. Note that the file name can vary over the lifetime of the file as it is renamed, but the history file name (above) never varies. *Do not use* this as the name of the history file. **(Optional)**

${UUID}
The universally unique identifier of the source file. This is invariant for the lifetime of the file. *Do not use* use this as the name of the history file. **(Optional)**

See also the *history_put_trashes_file* field, below.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (the integrate pass will fail).

Note: For projects created Aegis 4.26 or later, it is possible to change the history tool used by the project simply changing the various *history_*_command* documented in this man page. It is also possible to change the history tool for projects created with *aeimport* 4.26 or later, however the original tool must be available to access older file's revisions.

history_get_command = string;
This field is used to get a file from history. The command may be executed by developers. Used by the *aeipass*(1) and *aecp*(1) commands. All of the substitutions described by *aesub*(5) are available; in addition,

${History}
The absolute path of the history file. This may need to be reworked with the *Dirname* and *Basename* substitutions to yield a string suitable for the history tool in question.

${Edit}
The edit number to be extracted. It may be an arbitrary string, varying on the particular history tool.

${Output}
The absolute path of the destination file.

Executed as: the developer (or the executing user, in the case of the −independent option). Current directory: the base of the history tree Exit status: zero indicates success, all non-zero exits indicate failure (the *aecp* will fail).

history_put_command = string;
This field is used to add a new change to the history. The command is always executed as the project owner. Used by the *aeipass*(1) command.

It is strongly recommended that the *history_put_command* and *history_create__command* fields are identical. If not set, the *history_put_command* field defaults to the same value as the *history_create_command* field.

All of the substitutions described by *aesub*(5) are available; in addition,

${Input}
The absolute path of the source file.

${History}
The absolute path of the history file. This may need to be reworked with the *Dirname* and *Basename* substitutions to yield a string suitable for the history tool in question.

${File_Name}
The base relative file name of the file for this check-in. Note that the file name can vary over the lifetime of the file as it is renamed, but the history file name (above) never varies. *Do not use* this as the name of the history file. **(Optional)**

${UUID}
The universally unique identifier of the source file. This is invariant for the lifetime of the file. *Do not use* use this as the name of the history file. **(Optional)**

See also the *history_put_trashes_file* field, below.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (the integrate pass will fail).

history_transaction_begin_command = string;
The history_transaction_begin_command field is used to specify a command to be run by *aeipass*(1) before any history create or history put commands are run. The default is to do nothing.

All of the substitutions described in *aesub*(5) are available. If you need a transaction ID, use the *$version* substitution.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (the integrate pass will fail).

history_transaction_end_command = string;
The history_transaction_end_command field is used to specify a command to be run by *aeipass*(1) after any history create or history put commands are run, but before any history query commands are run. The default is to do nothing.

All of the substitutions described in *aesub*(5) are available. If you need a transaction ID, use the *$version* substitution.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (the integrate pass will fail).

history_transaction_abort_command = string;
The history_transaction_abort_command field is used to specify a command to be run by *aeipass*(1) to indicate that a transaction has been abandoned. The default is to do nothing.

All of the substitutions described in *aesub*(5) are available. If you need a transaction ID, use the *$version* substitution.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: ignored (the integrate pass has already failed).

history_query_command = string;
This field is used to query the topmost edit of a history file. Result to be printed on the standard output. This command may be executed by developers. Used by the *aeipass*(1) and *aecp*(1) commands. All of the substitutions described by *aesub*(5) are available; in addition,

${History}
The absolute path of the history file. This may need to be reworked with the *Dirname* and *Basename* substitutions to yield a string suitable for the history tool in question.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (the integrate pass will fail).

history_label_command = string;
This field contains a command to be executed whenever a *aeipass*(1) or *aedn*(1) command is successful. This command is invoked for **every** file in the project. So using it incurs a performance penalty. If this field is absent, nothing is done. All of the substitutions described by *aesub*(5) are available; in addition,

${History}
The absolute path of the history file.

${Edit}
The edit number to be labeled. It may be an arbitrary string, varying on the particular history tool.

${Label}
The label to be attached to the history. When executed from *aeipass*(1) this value is the same as *${Version}*, which may need to be reworked with the *${Subst}* substitutions to yield a string suitable for the history tool in question. When executed from *aedn*(1) it is set to the value passed in from the command line.

Executed as: the project owner. Current directory: the base of the history tree. Exit status: zero indicates success, all non-zero exits indicate failure (a warning will be issued).

Labeling does not scale, so the use of this command is not encouraged. If you have a project with 10,000 files, and a change modified exactly one of them, only one *history_put_command* execution is required, which operates on one history file. If you have labeling turned on, it will also be necessary to execute 10,000 *history_label_command*s, to add information Aegis will never use.

history_put_trashes_file = (fatal, warn, ignore);
Many history tools (e.g. RCS) can modify the contents of the file when it is committed. While there are usually options to turn this off, they are seldom used. The problem is: if the commit changes the file, the source in the repository now no longer matches the object file in the repository − i.e. the history tool has compromised the referential integrity of the repository.

fatal
Emit a fatal error if one or more source files are modified by a *history_put_command* or *history_create_command*. This is the default.

warn
Emit a warning if a source file is modified.

ignore
Ignore a source file changing. You sure better hope it was only in a comment!

history_content_limitation = (ascii_text, international_text, binary_capable);
This field describes the content style which the history tool is capable of working with.

ascii_text
The history tool can only cope with files which contain printable ASCII characters, plus space, tab and newline. The file must end with a newline. This is the default.

international_text
The history tool can only cope with files which do not contain the NUL character. The file must end with a newline.

binary_capable
The history tool can cope with all files without any limitation on the form of the contents.

When a file is added to the history (by either the *history_create_command* or the *history_put_command* field) it is examined for conformance to this limitation. If there is a problem, the file is encoded in either quoted printable for MIME64, whichever is smaller, before being given to the history tool. This encoding is transparent, the file in the baseline is unchanged.

On extract (the *history_get_command* field) the encoding is reversed, using information attached to the change file information. This is because each put could use a different encoding (although in practice, file contents rarely change that dramatically, and the same encoding is likely to be deduced every time).

Please note that this field **does not** apply to the *diff_command* or *merge_command* fields.

diff_command = string;
This field is used to difference of 2 files. The command is always executed by developers. Used by the *aed*(1) command. All of the substitutions described by *aesub*(5) are available; in addition,

${ORiginal}
The absolute path of the original file copied into the change. Usually in the baseline, but not always.

${Input}
The absolute path of the file in the development directory.

${Output}
The absolute path of the file in which to write the difference listing.

Executed as: the project owner (for integration diffs), or the developer (for development diffs). Current directory: the integration directory (for integration diffs), or the development directory (for development diffs). Exit status: zero indicates success, all non-zero exits indicate failure (the aed will fail).

Note: It is possible to configure a project to omit the diff step as unnecessary, by the following setting:

```
diff_command = "exit 0";
```

This disables all generation, checking and validation of difference file for each change source file. The merge functions of the *aediff*(1) command are unaffected by this setting.

merge_command = string;
This field is used to merge two competing edits to a file. The command is always executed by developers. The current directory will be the development directory. This field is used by the *aed*(1) command. All of the substitutions described by *aesub*(5) are available; in addition,

${ORiginal}
The absolute path of the original file copied into the change. Usually not in the baseline, often a temporary file.

${Most_Recent}
The absolute path of the competing edit, usually in the baseline.

${Input}
The absolute path of the file in the development directory. This is the “preferred” edit, if the tool has this concept when highlighting conflicting edits.

${Output}
The absolute path of the file in which to write the merged result. This will usually be the name if a change source file in the development directory.

It is important that this command does not move files around. (See the obsolete *diff3_command* field, below, for some history.)

Executed as: the project owner (for integration diffs), or the developer (for development diffs). Current directory: the integration directory (for integration diffs), or the development directory (for development diffs). Exit status: zero indicates success, all non-zero exits indicate failure (the aed will fail).

patch_diff_command = string;
The difference of 2 files, to send around as a patch. (This isn’t the same as diff_command, because it’s aimed at GNU Patch, not at humans.) The command is always executed by developers. Used by the *aepatch*(1) command.

Defaults to `"set +e; diff -c -L $index -L $index $original $input > $output; test $? -le 1"` if not set.

All of the substitutions described by *aesub*(5) are available; in addition,

${ORiginal}
The absolute path of the original file copied into the change. Usually in the baseline, but not always.

${Input}
The absolute path of the file in the development directory.

${Output}
The absolute path of the file in which to write the difference listing.

${INDex}
The project-relative name of the file, for use when the file name is embedded in the output. (Optional.)

Executed as: the project owner (for integration diffs), or the developer (for development diffs). Current directory: the integration directory (for integration diffs), or the development directory (for development diffs). Exit status: zero indicates success, all non-zero exits indicate failure (the aed will fail).

annotate_diff_command = string;
The difference of 2 files, for the use of the *aeannotate*(1) command. (This isn't the same as the *diff_command* field, because it's aimed at *aeannotate*(1), not at humans.) The command is always executed by developers. Used by the *aeannotate*(1) command.

Extreme care should be taken if you are considering setting this field, otherwise the result reported by *aeannotate*(1) may bear little relation to reality. The most useful option is GNU diff's **−−ignore-all-space** option, which will have the effect of ignoring the majority of indenting and code formatting changes. The **−−ignore-case** option could also be useful for case insensitive languages such as FORTRAN or PL/1. Avoid options which would alter the number of lines, such as **− −ignore-blank-lines** or **−−context** as these will produce misleading results.

Defaults to `"set +e; diff $option $original $input > $output; test $? -le 1"` if not set.

All of the substitutions described by *aesub*(5) are available; in addition,

${ORiginal}
The absolute path of the original file copied into the change. Usually in the baseline, but not always.

${Input}
The absolute path of the file in the development directory.

${Output}
The absolute path of the file in which to write the difference listing.

${INDex}
The project-relative name of the file, for use when the file name is embedded in the output. (Optional.)

${OPTion}
Extra options to be passed to the diff command, as set by the *aeannotate*(1) **−diff-option** command line option. Use with extreme care.

Executed as: the project owner (for integration diffs), or the developer (for development diffs). Current directory: the integration directory (for integration diffs), or the development directory (for development diffs). Exit status: zero indicates success, all non-zero exits indicate failure (the aed will fail).

review_policy_command = string;
This field is used to set the command to be executed by the *aerpass*(1) command. This command is useful in cases where the enterprise has determined that more than one review is necessary or that the reviewer must be senior to the developer, *etc*. Defaults to "`exit 0`" if not set.

The exit status is examined. An zero exit status (success) means that the change will proceed to the *awaiting integration* state; a non-zero exit status (failure) means that the change requires further review state, and the *develop_end_action* is consulted to determine the appropriate state

(*awaiting_review* or *being_reviewed*) for the change to move to.

All of the substitutions described by *aesub*(5) are available. Of particular interest are `${Change_Developer_List}` and `${Change_Reviewer_List}` for passing the specific staff involved with the change.

Executed as: the current reviewer. Current directory: the development directory. Exit status: zero indicates success, non-zero indicates failure.

For example, to have a script which is a project source file to be used to gate the code review process, a setting such as the following may be used:

```
review_policy_command =
  "$sh ${source script/reviewpolicy.sh} "
  "−p $project −c $change "
  "−d ${developer_list} "
  "−r ${reviewer_list}"
  ;
```

This is only one of many ways to implement a project specific review policy.

develop_end_policy_command = string;

This field is used to set the command to be executed by the *aede*(1) command. This command is useful in cases where the enterprise has determined that additional pre-conditions must be met (in addition to those already imposed by the *aede*(1) command) before a change may leave the *being developed* state. Defaults to "`exit 0`" if not set.

The exit status is examined. An zero exit status (success) means that the change may leave to the *being developed* state; a non-zero exit status (failure) means that the change requires further development.

All of the substitutions described by *aesub*(5) are available.

Executed as: the developer. Current directory: the development directory. Exit status: zero indicates success, non-zero indicates failure.

There are some common validations available in the *aede-policy*(1) command; you may choose all or only some of them, or you may choose to write a policy command specific to your project.

unchanged_file_develop_end_policy = (...);

This field may be used to control what happens when development of a change is ended, and the change contains files which have not had their contents or their attributes changed.

ignore Does not look for or warn about unchanged files. This the default.

warning If the change sets contains unchanged files, a warning will be issued for each one.

error If the change set contains unchanged files, an error will be issued for each one, and develop end will not complete (the change will remain in the *being developed* state).

unchanged_file_integrate_pass_policy = (...);

This field may be used to control what happens when a change is completed, and the change contains files which have not had their contents or their attributes changed.

ignore Does not look for or warn about unchanged files. The file version will be added to the history. This the default.

warning If the change sets contains unchanged files, a warning will be issued for each one. The file version will be added to the history.

remove If the change set contains an unchanged file, it will be silently removed from the change set. The file version will not be added to the history. The project file is unaffected.

test_command = string;

This field is used to set the command to be executed by the *aet*(1) command. Defaults to "$shell $file_name" if not set.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_Name}
The absolute path of the test to be executed.

${Search_Path}
Colon separated list of directories to search for tests and test support files. (This is a normal *aesub*(5) substitution.)

${Search_Path_Executable}
Colon separated list of directories to search for executable files and executable support files. Usually it is the same as the above, *except* during an “aet −bl” command.

${VARiables}
The text of name=value variable settings from the command line, suitably quoted to protect special character from the shell. Will be appended to the end of the command if not used explicitly.

Note that tests are source files, and thus never have the execute bit set.

Executed as: the project owner (for integration tests) or the developer (for development tests), or the executing user (for −independent tests). Current directory: the integration directory (for integration tests), the development directory (for development tests), the project baseline (for −bl tests), or the current directory (for −independent tests). Exit status: zero indicates success, one indicates failure, anything else indicates "no result".

development_test_command = string;
This field is used to set the command to be executed by the *aet*(1) command when a change is in the *being developed* state. Defaults to be the same as the *test_command* field if not set.

Note: It is a significantly bad idea to make tests behave differently in *being development* and *being integrated* states; avoid this at all costs.

All of the substitutions described in *aesub*(5) are available. In addition:

${File_Name}
The absolute path of the test to be executed.

${File_Name}
The absolute path of the test to be executed.

${Search_Path}
Colon separated list of directories to search for tests and test support files. (This is a normal *aesub*(5) substitution.)

${Search_Path_Executable}
Colon separated list of directories to search for executable files and executable support files. Usually it is the same as the above, *except* during an “aet −bl” command.

${VARiables}
The text of name=value variable settings from the command line, suitably quoted to protect special character from the shell. Will be appended to the end of the command if not used explicitly.

Note that tests are source files, and thus never have the execute bit set.

Executed as: the developer. Current directory: the development directory (for development tests), the project baseline (for −bl tests). Exit status: zero indicates success, one indicates failure, anything else indicates "no result".

batch_test_command = string;
This field is used to set the command to be executed by the *aet*(1) command, in preference to the *test_command* or *development_test_command*, if set. It is capable of running more than one test at once.

All of the substitutions described in *aesub*(5) are available. In addition:

${Output}
This is the name of the file to be generated to hold the test results. See *aetest*(5) for the format of this file.
A space separated list of absolute paths of the tests to be executed.

${File_Names}
The absolute path of the tests to be executed.

${File_Name}
The absolute path of the test to be executed.

${Search_Path}
Colon separated list of directories to search for tests and test support files. (This is a normal *aesub*(5) substitution.)

${Search_Path_Executable}
Colon separated list of directories to search for executable files and executable support files. Usually it is the same as the above, *except* during an “aet −bl” command.

${Current}
Number of first test in the batch.

${Total}
Total number of tests. If this is 0 then no progress messages should be issued.

${VARiables}
The text of name=value variable settings from the command line, suitably quoted to protect special character from the shell. Will be appended to the end of the command if not used explicitly.

Note that tests are source files, and thus never have the execute bit set.

It is strongly recommended that you design your test scripts so that they may be executed by either batch or non-batch methods. This permits simple migration when your environment changes.

Executed as: the project owner (for integration tests) or the developer (for development tests), or the executing user (for −independent tests). Current directory: the integration directory (for integration tests), the development directory (for development tests), the project baseline (for −bl tests), or the current directory (for −independent tests). Exit status: zero indicates success, one indicates failure, anything else indicates "no result".

architecture_discriminator_command = string;
If this field is present it is used as a command to be executed in order to further identify the platform architecture (see below). All of the substitutions described by *aesub*(5) are available;
Executed as: the developer. Current directory: the development directory of the change. Exit status: zero indicates success, all non-zero exits indicate failure.

architecture = [{ ... }];
This field is a list of system and machine architectures on which each change must successfully build and test. May be assigned more than once. The structures listed have fields as follows:

name = string;
The name of the architecture. This name is available in the *${ARCHitecture}* substitution (see *aesub*(5) for more information), as well as being used internally by Aegis. You may use almost any name for your architecture, but it is best to avoid shell special characters and white space, because it may be substituted into commands to be executed by Aegis.

pattern = string;

The system and machine architecture are determined by using the *uname*(2) system call. The *uname*(2) return value is assembled into a string of the form "*sysname-release-version-machine*", or "*sysname-release-version-machine-disc*" if *architecture_discriminator_command* is used.

The pattern field must match this uname result string. The first match found is used. The pattern is a shell file name pattern, see *sh*(1) for more information.

For example, the pattern *SunOS-4.1*-*-sun4** matches a machine the author commonly uses, which returns *SunOS-4.1.3-8-sun4m* from the *uname*(2) system call.

mode = (required, optional, forbidden);

The *mode* field is used to control how the architecture information is used.

required Architectures of thus mode will be copied into changes as their required architectures when the change is created. This is the default.

optional Architectures of thus mode will *not* be copied into changes as their required architectures when the change is created. However, if you add them subsequently, they become required *for that change*.

forbidden

Aegis will refuse to build or test on architectures of this mode.

When a change is created, the *required* architecture names are copied into the change's architecture list. Once names are in this list, they are required for the change, and the project attributes are less relevant.

If the architecture field is not set, it defaults to

```
architecture =
[
        {
                name = "unspecified";
                pattern = "*";
                mode = required;
        }
];
```

file_template = [{ ... }];

The file template is consulted whenever a new file is created, by one of the *aenf*(1) or *aent*(1) commands. May be assigned more than once. Each list item has the form:

pattern = [string];

The name of the file, relative to the development directory. Each string is a shell file name pattern; see *sh*(1) for more information. The patterns are matched against the whole filename; naming only the last filename path element will *not* work (unless the pattern starts with "*").

body_command = string;

Command to run to initialize the body of the file.
Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

body = string;

What to initialize the body of the file to.

All of the substitutions described in *aesub*(5) are available for the *body* and *body_command* strings. (Only specify one of them.) In addition:

${File_Name}

will be replaced by the name of the new file.

whiteout_template = [{ ... }];
: The file template is consulted whenever a file is removed, by one of the *aerm*(1) or *aemv*(1) commands. It is used to place a "whiteout" entry in the development directory, in order to induce compile errors of the removed file is referenced during the build. Each list item has the form:

  pattern = [string];
  : The name of the file, relative to the development directory. Each string is a shell file name pattern; see *sh*(1) for more information. The patterns are matched against the whole filename; naming only the last filename path element will *not* work (unless the pattern starts with "*").

  body = string;
  : What to initialize the body of the file to. If not present, no whiteout file will be created; if the empty string, a zero-length whiteout file will be created.

  All of the substitutions described in *aesub*(5) are available for the body string. In addition:

  ${File_Name}
  : will be replaced by the name of the removed file.

  If the name of the file being removed does not match any of the filename patterns, a file consisting of 1KB of very ugly garbage will be generated. The idea is that it will produce a syntax error for most languages if you try to run it, compile it, or include it.

maximum_filename_length = integer;
: This field is used to limit the length of file names. All new files may not have path components longer than this. Existing files are not affected. The last component must also allow for the ",D" suffix of difference files. Where this value is larger than the file system allows, the file system limit will be imposed. Defaults to 255 if not set. Legal values range from 9 to 255.

  The file name lengths of project files will be checked at develop end if the project *aegis.conf* file is in the change. See *aede (1)* for more information.

posix_filename_charset = boolean;
: This field may be used to limit the characters allowed in file names to only those explicitly allowed by POSIX. Defaults to false if not set.

  For a filename to be portable across conforming implementations of IEEE Std 1003.1-1988, it shall consist only of alphanumeric characters, dot, hyphen or underscore. Hyphen shall not be used as the first character of a portable filename.

  If this field is false, all characters are allowed except non-printing characters, space characters and leading hyphens.

dos_filename_required = boolean;
: This field may be used to limit file names so that they conform to the DOS 8+3 filename limits and to the DOS filename character set. Also denies file names which look like devices (AUX, *etc*). Defaults to false if not set. This field is used in combination with the other filename fields, it does not replace them.

windows_filename_required = boolean;
: This field may be used to limit file names so that they conform to the Windows98 and WindowsNT filename limits and character set. Also denies file names which look like devices (AUX, *etc*). Defaults to false if not set. This field is used in combination with the other filename fields, it does not replace them.

shell_safe_filenames = boolean;
: This field may be used to limit file names so that they may not contain shell special characters. If you do not set this to true, you will need to use the ${quote} substitution around file names in commands, or risk unexpected errors.

  This field defaults to true if not set.

The white space characters (space, tab, newline, *etc*) are considered shell special characters.

allow_white_space_in_filenames = boolean;
: This field may be used to allow white space characters in file names. This will allow the following characters to appear in filenames: backspace (BS, \b, 0x08), horizontal tab (HT, \t, 0x09), new line (NL, \n, 0x0A), vertical tab (VT, \v, 0x0B), form feed (FF, \f, 0x0C), and carriage return (CR, \r, 0x0D).

  Defaults to false if not set.

  Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as well.

  The user must take great care to use the ${quote} substitution around all file names in commands in the project configuration. And even then, substitutions which expect a space separated list of file names will have undefined results.

allow_non_ascii_filenames = boolean;
: This field may be used to allow file names with non-ascii-printable characters in them. Usually this would mean a UTF8 or international charset of some kind.

  Defaults to false if not set.

  Note that this field does not override other file name filters. It will be necessary to explicitly set *shell_safe_filenames = false* as well. It will be necessary to set *dos_filename_required = false* (the default) as well. It will be necessary to set *posix_filename_charset = false* (the default) as well.

filename_pattern_accept = [string];
: This field is used to specify a list of patterns of acceptable file names. The patterns are matched against each filename path element. The patterns are constructed from the usual shell filename wild-cards. Defaults to "*" if not set.

filename_pattern_reject = [string];
: This field is used to specify a list of patterns of unacceptable file names. The patterns are matched against each filename path element. The patterns are constructed from the usual shell filename wild-cards. Defaults to "*,D" if not set. The pattern "*,D" is always appended. Where the *filename_pattern_accept* and filename_pattern_reject fields conflict, the reject takes precedence.

new_test_filename = string;
: This field is used to form the filename of new tests, where the filename is not specified on the aent command line. Defaults to "test/${zpad $hundred 2}/t${zpad $number 4}${left $type 1}.sh" if not set.

  All of the substitutions defined in *aesub*(5) are available. The following three substitutions are also available:

  $Hundred
  : The test number divided by 100, optional

  $Number
  : The test number, mandatory

  $Type The test type: "automatic" or "manual", optional

development_directory_template = string;
: This field is used to determine the name of the development directory at develop begin. All of the substitutions defined in *aesub*(5) are available. The following substitutions is also available:

Default_Development_Directory
The directory within which the development directory is to be created.

Magic A single letter, starting from “C”, which can be inserted. This must be used, as it allows Aegis to try different names should there be a conflict.

If not set, defaults to "`$ddd/${left $p ${expr ${namemax $ddd} - ${length .$magic$c}}}.$magic$c`".

For DOS compatibility (8+3 file names), a useful setting is "`$ddd/${downcase ${left ${id $p} 8}.$magic${right 0$c 2}}`". This ensures that the filename is always a valid 8.3 filename, that it is always lowercase, and it translates any punctuation in the project name into underscores.

metrics_filename_pattern = string;
This field is used to form the name of the metrics file, given a source file. All of the substitutions defined in *aesub*(5) are available. The following substitutions is also available:

File_Name
The absolute path name of the source file.

Defaults to "$filename,S" if not set.

trojan_horse_suspect = [string];
This list of filename patterns is consulted by aedist −receive when it is checking for files which could be used to host Trojan horse attacks. This will be different for different projects, so you will need to update this yourself. The patterns are matched against the whole filename; naming only the last filename path element will *not* work (unless the pattern starts with “`*`”).

project_specific = [{ ... }];
This is a list of name and value pairs for use within the ${project-specific} substitution (see *aesub*(5) for more information). May be assigned more than once. The sub-fields are

name = string;
The name of the value. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not. Attribute names are case-insensitive.

value = string;
The value to be substituted.

There are almost no limitations on the strings which may appear in either of these fields.

There are several attribute names which are known to and used by Aegis, these include:

aede-policy
This attribute is used when no policy names are listed on the *aede-policy*(1) command line.

ae-repo-ci:commit-message
See *ae-repo-ci*(1) for more information.

aede-policy
See *aede-policy*(1) for more information.

aede-policy:version-info:library
See *aede-policy*(1) for more information.

aeget:inventory:hide
See *aeget*(1) for more information.

aemakegen:debian:build-depends
See *aemakegen*(1) for more information.

aemakegen:debian:copyright
See *aemakegen*(1) for more information.

aemakegen:debian:depends
: See *aemakegen*(1) for more information.

aemakegen:debian:description:*name*
: See *aemakegen*(1) for more information.

aemakegen:debian:extended-description:*name*
: See *aemakegen*(1) for more information.

aemakegen:debian:homepage
: See *aemakegen*(1) for more information.

aemakegen:debian:maintainer
: See *aemakegen*(1) for more information.

aemakegen:debian:priority
: See *aemakegen*(1) for more information.

aemakegen:debian:section
: See *aemakegen*(1) for more information.

aemakegen:pkg-config:cflags
: See *aemakegen*(1) for more information.

aemakegen:pkg-config:conflicts
: See *aemakegen*(1) for more information.

aemakegen:pkg-config:libs
: See *aemakegen*(1) for more information.

aemakegen:pkg-config:libs.private
: See *aemakegen*(1) for more information.

aemakegen:pkg-config:requires
: See *aemakegen*(1) for more information.

aemakegen:version-info
: See *aemakegen*(1) for more information.

aetar:exclude
: This attribute is used by he *aetar*(1) receive command to exclude files in tarballs from consideration. This is a space separated list of file names.

copyright-owner
: This string is available via the *${copyright-owner}* substitution, and is the one checked by the *aede-policy*(1) command. Only set this attribute if your project is a work-for-hire under copyright law. It defaults to the value of *${user name}* if not set, this is almost always correct for Open Source projects.

html:body-begin
: This attribute is used by the *aeget*(1) command to customize generated web pages. See *aeget*(1) for more information.

html:meta
: This attribute is used by the *aeget*(1) command to customize generated web pages. See *aeget*(1) for more information.

html:body-end
: This attribute is used by the *aeget*(1) command to customize generated web pages. See *aeget*(1) for more information.

svn:password
: See *ae-repo-ci* for more information.

svn:username
See *ae-repo-ci* for more information.

When commands are executed by Aegis, it ensures that the AEGIS_PROJECT, AEGIS_CHANGE, AEGIS_ARCH, LINES and COLS environment variables are set appropriately. The project configuration file's *project_specific* field is also consulted, looking for value's whose name starts with "setenv:" and sets the corresponding environment variable. All of the substitutions described by *aesub*(5) are available. For example: specifying a PATH and a SEARCH_PATH to be used for all commands may be set as follows:

```
project_specific =
[
  {
    name = "setenv:PATH";
    value = "/usr/bin:/bin";
  },
  {
    name = "setenv:SEARCH_PATH";
    value = "${search_path}";
  },
];
```

As many environment variables as desired may be specified in this way.

build_time_adjust = (...);
This field controls the adjustment of file modification times at the end of integrate-pass. File times are adjusted so that development directories are, in the main, out of date with respect to the baseline. The idea is that, at the very least, programs need to be re-linked so that *aet −reg* does not give false negatives.

Combining this with the *project_file_command* (above) can alleviate the vast majority of file modification time inconsistencies experienced as a result of a project integration and the subsequent changes in the baseline's file modification times.

Unless you are a masochist, do not set this field. Leave it as the default.

adjust_and_sleep
Causes the file times to be adjusted, and if the file times would extend into the future, aeipass will sleep until that time has passed. This is the default.

adjust_only
Causes the file times to be adjusted. If the file time extend into the future, a warning is issued.

dont_adjust
File modification times are not adjusted. This is a really bad idea. Really. Make sure that, at the very minimum, *project_file_command* touches all of the change's files, otherwise the build problems which ensue are going to take you weeks to track down and lose you much productivity. You have been warned.

See also the *build_time_adjust_notify_command* field.

signed_off_by = boolean;
If this field is set each *aedb*(1), *aechown*(1), *aede*(1) and *aerpass*(1) will append a `Signed-off-by` line to the change description. This field should only be set to true for open source projects.

For a description of `Signed-off-by` see http://www.ussg.iu.edu/hypermail/linux/kernel/0405.2/1301.html and http://www.osdl.org/newsroom/press_releases/2004/2004_05_24_dco.html

cache_project_file_list_for_each_delta = boolean;

It is possible to have Aegis cache the list of project files that were present at integrate pass for each delta (integrated change set). This is used to optimize all project-history-based operations, such as *aecp −delta* or *aepatch*(1).

This cache will optimize many operations which would otherwise require time to reconstruct the project state using the roll-forward data available in each change set. However, it comes at the cost of disk space, and not everyone can afford more and more disk.

This field defaults to true if not set.

clean_exceptions = [string];

It is possible to have Aegis exclude from the clean process any file that match one of the pattern listed in the clean_exceptions list.

This field default to an empty list if not set.

cache_project_file_list_for_each_delta = boolean;

It is possible to have Aegis cache the list of project files that were present at integrate pass for each delta (integrated change set). This is used to optimize all project-history-based operations, such as *aecp −delta* or *aepatch*(1).

This cache will optimize many operations which would otherwise require time to reconstruct the project state using the roll-forward data available in each change set. However, it comes at the cost of disk space, and not everyone can afford more and more disk.

This field defaults to true if not set.

RSS FEEDS

Aegis has the ability to feed RSS channels when change sets transition states. See the User Guide for full details. Following is a brief description of the project-specific attributes used to control this process.

Create / Add to a channel

An RSS channel is specified with the rss:feedfilename project_specific attribute:

```
project_specific =
        [
          {
            name = "rss:feedfilename−<filename>";
            value = "<space-separated list of states>";
          }
        ]
```

Specify the Description of an RSS channel

The description of an RSS channel is specified with the rss:feeddescription project_specific attribute:

```
project_specific =
        [
          {
            name = "rss:feeddescription−<filename>";
            value = "<description>";
          }
        ]
```

Specify the Title of an RSS channel

The title of an RSS channel is specified with the rss:feedtitle project_specific attribute:

```
project_specific =
        [
          {
            name = "rss:feedtitle−<filename>";
            value = "<title>";
```

```
    }
  ]
```

Specify the Language of an RSS channel

The language of an RSS channel is specified with the rss:feedlanguage project_specific attribute:

```
project_specific =
  [
    {
      name = "rss:feedlanguage−<filename>";
      value = "<language";
    }
  ]
```

OBSOLETE FIELDS

There are some obsolete fields in the file. They are provided for backwards compatibility only, and should not be used.

diff3_command = string;

This field is used to difference 3 files. The command is always executed by developers. Used by the *aed*(1) command. All of the substitutions described by *aesub*(5) are available; in addition,

${ORiginal}

The absolute path of the original file copied into the change. Usually not in the baseline.

${Most_Recent}

The absolute path of the competing edit, usually in the baseline.

${Input}

The absolute path of the file in the development directory.

${Output}

The absolute path of the file in which to write the difference listing.

Executed as: the project owner (for integration diffs), or the developer (for development diffs). Current directory: the integration directory (for integration diffs), or the development directory (for development diffs). Exit status: zero indicates success, all non-zero exits indicate failure (the aed will fail).

The problem with this field was that the default usage placed the merged source in a strange place. And subsequent *aed*(1) commands would over-write it. This meant that merges would be lost, causing a number of nasty problems. Some sites overcame this by adding “mv” commands to put the output back where the input came from, but this meant that Aegis’ commentary was misleading. Use the “merge_command” field instead. It is almost identical, but Aegis will move the files around for you − so you get the good behavior by default (no lost merges) and the error message is consistent.

create_symlinks_before_build = boolean;

This flag is true if Aegis should create symlinks from the development directory to the baseline for all files in the baseline not in the development directory immediately before a development_build_command is issued. Usually used to trick dumb DMTs into believing the development directory contains an entire copy of the project, though sometimes the DMT is smart enough, the tools it must work with are not. Symlinks in the development directory which point to nonexistent files will be removed.

Defaults to false if not set.

create_symlinks_before_integration_build = boolean;

This flag is true if Aegis should create symlinks from the integration directory to the ancestral baseline for all files in the ancestral not in the integration directory immediately before a build_command is issued. Usually used to trick dumb DMTs into believing the integration

directory contains an entire copy of the project, though sometimes the DMT is smart enough, the tools it must work with are not. Symlinks in the integration directory which point to nonexistent files will be removed.

Defaults to the same value as *create_symlinks_before_build* if not set.

remove_symlinks_after_build = boolean;

This flag is true if Aegis should remove symlinks which point from the development directory to the baseline directory immediately after a development_build_command is issued. Only consulted if the *create_symlinks_before_build* field is true, for the purpose of reversing the actions of the *create_symlinks_before_build* field.

Defaults to false if not set.

remove_symlinks_after_integration_build = boolean;

This flag is true if Aegis should remove symlinks which point from the integration directory to the ancestral baseline directory immediately after a build_command is issued. Only consulted if the *create_symlinks_before_integration_build* field is true, for the purpose of reversing the actions of the *create_symlinks_before_integration_build* field.

Defaults to **true** if not set. This default is intentional. It is important that there are no symlinks in the (new) baseline, because they could go stale between integrations. If you set this field to false, *caveat emptor*.

CHANGING HISTORY TOOL

Since version Aegis 4.26 it is possible to change the history tool, if needed. Note, however, that the old tool is still required to retrieve file's revisions saved before the switch to the new tool. As an example it is not advisable to uninstall the *fhist* package if a project switched to *aesvt*, since *fhist* will be required to retrieve file's revisions pre−dating the switch.

The *aeipass*(1) command stores the *history_get_command* as the *aegis:history_get_command* user defined change's attribute, that command will be used later to retrieve files altered by the change.

In any project created using Aegis v. 4.26 (or later) it is possible to use a different history tool simply editing the appropriate *aepconf*(5) fields.

It is possible to change the history tool of projects created with a older Aegis version, but it's required to properly set the *aegis:history_get_command* user defined attribute for each change integrated before the switch.

SEE ALSO

aeb(1) build a change

aecp(1) copy a file into a change

aecpu(1)
reverse action of aecp

aed(1) find differences between a change and the baseline

aede(1) end development of a change *aede-policy*(1) check things about a change

aerpass(1)
pass a review of a change

aeib(1) begin integration of a change

aeipass(1)
pass integration of a change

aemv(1) rename a file as part of a change

aenf (1) add new files to be created by a change

aenfu(1) remove new files from a change

aent(1) add a new test to be created by a change

aentu(1) remove new tests from a change

aet(1) run tests

aegis(5) aegis file format syntax

aesub(5)
available command substitutions

aetest(5)
batch test results file

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aepstate − aegis project state file

SYNOPSIS

project`/info/state`

DESCRIPTION

The *project*`/info/state` file is used to store state information about a project.

This file is maintained by **aegis** and thus should not be edited by humans.

CONTENTS

next_test_number = integer;

Each test is numbered uniquely across all branches of the project. The name is of the form *t[0−9][0−9][0−9][0−9][am].sh* (’a’ for automatic and ’m’ for manual.)

Almost Obsolete Fields

The following fields are obsolete. They will persist until the next *aenrls*(1), and the new project so generated will use them to define its default branching.

version_major = integer;

The major version number of this release of the project. Always one or more.

version_minor = integer;

The minor version number of this release of the project. Always zero or more.

Obsolete Fields

The following fields are obsolete. They are only present in projects which have yet to be converted to the new branch format. When *Aegis* sees them, they will be moved into the "trunk" transaction.

description = string;

This field contains a description of the project. Large amounts of prose are not required; a single line is sufficient.

owner_name = string;

This field is ignored.

group_name = string;

This field is ignored.

developer_may_review = boolean;

If this field is true, then a developer may review her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

developer_may_integrate = boolean;

If this field is true, then a developer may integrate her own change. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

reviewer_may_integrate = boolean;

If this field is true, then a reviewer may integrate a change she reviewed. This is probably only a good idea for projects of less than 3 people. The idea is for as many people as possible to critically examine a change.

developers_may_create_changes = boolean;

This field is true if developers may created changes, in addition to administrators. This tends to be a very useful thing, since developers find most of the bugs.

forced_develop_begin_notify_command = string;

This command is used to notify a developer that a change requires developing; it is issued when a project administrator uses an *aedb −User* command to force development of a change by a specific user. All of the substitutions described in *aesub*(5) are available. This field is optional.

Executed as: the new developer. Current directory: the development directory of the change for

the new developer. Exit status: ignored.

develop_end_notify_command = string;
: This command is used to notify that a change is ready for review. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

develop_end_undo_notify_command = string;
: This command is used to notify that a change had been withdrawn from review for further development. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the developer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_notify_command = string;
: This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

review_pass_undo_notify_command = string;
: This command is used to notify that a review has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. Defaults to the same action as the *develop_end_notify_command* field. All of the substitutions described by *aesub*(5) are available.

review_fail_notify_command = string;
: This command is used to notify that a review has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Executed as: the reviewer. Current directory: the development directory of the change. Exit status: ignored.

integrate_pass_notify_command = string;
: This command is used to notify that an integration has passed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

  Some compilers bury absolute path names into object files and executables. The renaming of the integration directory to become the new baseline breaks these paths. This command is passed an environment variable called AEGIS_INTEGRATION_DIRECTORY so that the appropriate symlink may be placed, if desired.

  Executed as: the project owner. Current directory: the new project baseline. Exit status: ignored.

integrate_fail_notify_command = string;
: This command is used to notify that an integration has failed. It will probably use mail, or it could be an in-house bulletin board. This field is optional, if not present no notification will be

given. This command could also be used to notify other management systems, such as progress and defect tracking. All of the substitutions described by *aesub*(5) are available.

Executed as: the integrator. Current directory: the development directory of the change. Exit status: ignored.

default_development_directory = string;
The pathname of where to place new development directories. The pathname must be absolute. This field is only consulted if the field of the same name in the user configuration file is not set.

umask = integer;
File permission mode mask. See *umask*(2) for more information. This value will always be OR'ed with 022, because *aegis* is paranoid.

default_test_exemption = boolean;
This field contains what to do when a change is created with no test exemption specified.

copyright_years = [integer];
This field contains a list of copyright years, for use in copyright notices, etc. It is updated each integrate_begin, if necessary, to include the current year. Available as the ${Copyright_Years} substitution, and included in the version listing.

next_change_number = integer;
Changes are numbered sequentially from one. This field records the next unused change number.

next_delta_number = integer;
Deltas are numbered sequentially from one. This field records the next unused delta number.

src = [{ ... }];
If you are writing a report, see *aefstate*(5) for the current documentation for this field. This field is a list of files in the project. Each list item has the form:

file_name = string;
The name of the file, relative to the baseline.

usage = (source, config, build, test, manual_test);
What the file is for.

edit_number = string;
The edit number of the file.

locked_by = integer;
The change which locked this file.
Caveat: this field is redundant, you can figure it out by scanning all of he change files. Having it here is very convenient, even though it means multiple updates.

about_to_be_created_by = integer;
The change which is about to create this file for the first time. Same caveat as above.

deleted_by = integer;
The change which last deleted this file. We never throw them away, because (a) it may be created again, and more important (b) we need it to recreate earlier deltas.

history = [{ ... }];
This field contains a history of integrations for the project. Updated by each successful 'aegis −Integrate_Pass' command.

delta_number = integer;
The delta number of the integration.

change_number = integer;
The number of the change which was integrated.

name = [string];
The names by which this delta is known.

change = [integer];
The list of changes which have been created to date.

administrator = [string];
The list of administrators of the project.

developer = [string];
The list of developers of the project.

reviewer = [string];
The list of reviewers of the project.

integrator = [string];
The list of integrators of the project.

currently_integrating_change = integer;
The change currently being integrated. Only one change (within a project) may be integrated at a time. Only set when an integration is in progress.

version_major = integer;
The major version number of this release of the project. Always one or more.

version_minor = integer;
The minor version number of this release of the project. Always zero or more.

version_previous = string;
The version number this project was derived from. This is of most use when producing "patch" files.

WRITING REPORT SCRIPTS

When attempting to access these fields from within the report generator, you need a code fragment similar to the following:

```
auto ps;
ps = project[project_name()].state;
```

All of the fields mentioned in the man page can now be accessed as members of the `ps` variable.

When you access the *branch* field, you obtain access to the change state of the branch. Even the trunk has one of these, it just doesn't have a number, and it is perpetually being developed.

When you index the *branch.change* field by a change number, you obtain access to the change state of that change.

When you index the *branch.src* field by a filename string, you may obtain access the the relevant project file state (see *aefstate*(5) for more information).

In addition to the above fields, the report generator inserts a *name* field containing the project name, and a *directory* field containing the project directory path.

SEE ALSO

aenpr(1)
create a new project

aegis(5) aegis file format syntax

aepattr(5)
project attributes file format

aecstate(5)
change state file

aefstate(5)
file state file

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aer − aegis report script language definition

DESCRIPTION

This manual entry describes the report generator script language used by the *aer*(1) command. The language resembles C, with a touch of *awk* and *perl* for flavour. It also closely resembles the appearance of aegis' database files.

This language grew out of the need to have a general purpose programming language to describe reports, and yet be as familiar as possible to the people who will be using it.

WORDS AND SYMBOLS

This section describes the various words and symbols understood by the language.

Names

A name is a contiguous set of alphanumeric characters, including underscore (_). It must not start with a digit. Names may be of any length. Names are case sensitive, so uppercase and lowercase letters are unique.

Here are some examples of names

print	sqrt	if
how_long	UpperCase	dig57

Some words are *reserved* as keywords. These are the words which appear in **bold** in the statement descriptions, below.

Integer Constants

An integer constant may be decimal, any sequence of digits. Constants may be octal, any sequence of octal digits starting with a zero. Constant may be hexadecimal, any sequence of hexadecimal digits, starting with a `0x` prefix. These are represented by the internal `long` type, so significance is limited.

Here are some examples of integer constants:

43	015	0xbeEf
2147483647	017777777777	0x7FFFFFFF

Floating Point Constants

A floating point constant has an integer part, a fraction part and an exponent part.

Here are some examples of floating point constants:

1.2e3	4.2e+1	1.628e-94
0.567	5e6	.67

String Constants

A string constant is represented as characters within double quotes ("). All characters in the script file are required to be printable, so special characters are represented by *escape sequences.* These escape sequences are:

`\"`	the " character
`\\`	the \ character
`\n`	Newline
`\f`	Form Feed
`\r`	Carriage Return
`\b`	Backspace
`\t`	Horizontal Tab
`\`*nnn*	octal character value

Here are some examples of string constants:

"Hello, World!"	"Go away"	""
"The End0	"slosh is \\"	"Say \"Please\""

Symbols

The non-alphanumeric characters are used to represent symbols, usually expression operators or statement terminators. The symbols used include:

```
 !    !=   !~   ##   ##=
 %    %=   &    &&   &=
 (    )    *    **   **=
 *=   +    ++   +=   ,
 -    --   -=   .    /
 /=   :    ;    <    <<
 <<=  <=   =    ==   >
 >=   >>   >>=  ?    [
 ]    ^    ^=   {    |
 |=   ||   }    ~    ~~
```

White Space

White space serves to separate words and symbols, and has no other significance. The language is free-form. White space includes the SPACE, TAB, FF, and NEWLINE characters.

Comments

Comments are delimited by `/*` and `*/` pairs, and are treated as a single white space character.

STATEMENTS

Statement serve to control the flow of execution of the program, or the existence of variables.

The Expression Statement

The commonest statement consists of an expression terminated by a semicolon. The expression is evaluated, and any result is discarded.

Examples of this statement include

```
x = 42;
print("Hello, World!0);
```

The If Statement

The *if* statement is used to conditionally execute portions of code. Examples if the *if* statement include:

```
if (x == 42)
    x = 1;
if (x * x < 1)
    print("no");
else
    print("yes");
```

The For Statement

The *for* statement has two forms. The first form is described as

```
for (expr1; expr2; expr3)
    stmt
```

The *expr1* is done before the loop begins. The *expr2* controls, the loop; if it does not evaluate to `true` the loop terminates. The loop body is the *stmt*. The loop increment is done by the *expr3*, and the the test is performed again.

Each of the expressions is optional; any or all may be omitted.

Here is an example of a *for* loop:

```
for (j = 0; j < 10; ++j)
    print(j);
```

The second form of the *for* statement looks like this:

```
for (name in keys(passwd))
    print(name, passwd[name].pw_comment);
```

The Break Statement

The *break* statement is used to break out of a loop.

Here is an example of a *break* statement:

```
for (j = 0; ; j = 2 * j + 4)
{
    print(j);
    if (j >= 0x800)
        break;
}
```

The *break* statement works within all loop statements.

The Continue Statement

The *continue* statement is used to terminate the loop body and start another repetition.

Here is an example of a *continue* statement:

```
for (j = 0; j < 1000; j = 2 * j + 4)
{
    if (j < 42)
        continue;
    print(j);
}
```

The *continue* statement works within all loop statements.

The While Statement

The *while* statement is another loop construct. The condition is evaluated before the loop body.

```
line = 0;
while (line < 7)
{
    print("");
    ++line;
}
```

The Do Statement

The *do* statement is another loop construct. The condition is evaluate after the loop body.

```
do
    print("yuck");
while
    (line++ < 7);
```

The Compound Statement

The *compound* statement is a way of grouping other statements together. It is enclosed in curly braces.

```
if ( lines < 7)
{
    print("This\n");;
    print("could\n");;
    print("have\n");;
    print("been\n");;
    print("seven\n");;
    print("blank\n");;
    print("lines.\n");;
}
```

The Local Statement

The *auto* statement is used to declare variables and initialize them to be nul.

```
auto x, y, z;
x = 42;
```

All user-defined variables must be declared before they are used.

The Null Statement

The *null* statement does nothing. It consists of a single semicolon. It is most often seen as a loop body.

```
for (n = 0, bit = 1; n < bit_num; ++n, bit <<= 1)
    ;
```

The Try Catch Statement

The *try catch* statement is used to catch errors which would usually cause the report to fail.

```
try
    statement1
catch (variable)
    statement2
```

The first statement is executed. If no error occurs, nothing else is done. If an error occurs in the execution of the first statement the firsdt statement execution is terminated and then the given variable is set to a description of the error and the second statement is executed.

EXPRESSIONS

Expressions are much the same as in C, using the same operators. The following table describes operator precedence and associativity:

`[ ]`	subscripting	*value* `[` *expr* `]`
`( )`	function call	*expr* `(` *expr_list* `)`
`( )`	grouping	`(` *expr* `)`
`++`	post increment	*lvalue* `++`
`++`	pre increment	`++`*lvalue*
`--`	post decrement	*lvalue* `--`
`--`	pre decrement	`--`*lvalue*
`~`	compliment	`~` *expr*
`!`	not	`!` *expr*
`-`	unary minus	`-` *expr*
`+`	unary plus	`+` *expr*
`**`	exponentiation	*expr* `**` *expr*
`*`	multiply	*expr* `*` *expr*
`/`	divide	*expr* `/` *expr*
`%`	modulo (remainder)	*expr* `%` *expr*
`~~`	matches	*expr* `~~` *expr*
`!~`	does not match	*expr* `!~` *expr*
`in`	list member	*expr* `in` *expr*
`+`	addition (plus)	*expr* `+` *expr*
`-`	subtraction (minus)	*expr* `-` *expr*
`##`	list and string join	*expr* `##` *expr*
`<<`	shift left	*expr* `<<` *expr*
`>>`	shift right	*expr* `>>` *expr*
`<`	less than	*expr* `<` *expr*
`<=`	less than or equal	*expr* `<=` *expr*
`>`	greater than	*expr* `>` *expr*
`>=`	greater than or equal	*expr* `>=` *expr*

`==`	equal	*expr* `==` *expr*
`!=`	not equal	*expr* `!=` *expr*
`&`	bitwise AND	*expr* `&` *expr*
`^`	bitwise exclusive OR	*expr* `^` *expr*
`|`	bitwise inclusive OR	*expr* `|` *expr*
`? :`	arithmetic if	*expr* `?` *expr* `:` *expr*
`=`	simple assignment	*expr* `=` *expr*
`*=`	multiply and assign	*expr* `*=` *expr*
`/=`	divide and assign	*expr* `/=` *expr*
`%=`	modulo and assign	*expr* `%=` *expr*
`+=`	add and assign	*expr* `+=` *expr*
`-=`	subtract and assign	*expr* `-=` *expr*
`<<=`	shift left and assign	*expr* `<<=` *expr*
`>>=`	shift right and assign	*expr* `>>=` *expr*
`&=`	AND and assign	*expr* `&=` *expr*
`^=`	exclusive OR and assign	*expr* `^=` *expr*
`|=`	inclusive OR and assign	*expr* `|=` *expr*
`,`	comma (sequencing)	*expr* `,` *expr*

Most of these operators behave as they do in C, but some of these operators will require some explanation.

Exponentiation

The `**` operator raises the left argument to the right'th power. It is right associative.

Match

The `~~` operator compares two strings. It returns a number between 0.0 and 1.0. Zero means completely different, one means identical. Case is significant.

Not Match

The `!~` is used to compare two strings, and returns the opposite of the `~~` operator; one if completely different, and zero if identical.

String Join

The ## operator is used to join two strings together.

TYPES

There are several types used within the report language.

array Values of this type contain other values, indexed by a string. If you attempt to index by an arithmetic type, it will be silently converted to a string. Use the *keys* function to determine all of the keys; use the *count* function to determine how many entries an array has. The type of an array element is not restricted, only the index must be a string.

boolean This type has two values: `true` and `false`. These value arise from the boolean operators described earlier.

integer This type is represented by the *long* C type. It has a limited range of values (usually −2e9 to 2e9 approximately). If used in a string context, it will be silently converted to a string. For exact control of the format, used the *sprintf* function.

list Values of this type contain a list of other values. The type of these values is not restricted. The array index operator (e[e]) may be used to access list elements; indexes start at zero (0).

string Values of this type are an arbitrary string of C characters, except the NUL character ( ). Strings may be of any length.

struct Values of this type contain additional values. These values are accessed using the "dot" operator. These values may also be treated as if they were arrays.

real This type is represented the the *double* C type. If used in a string context, it will be silently converted to a string. For exact control of the format, used the *sprintf* function.

FUNCTIONS

There are a number of built-in functions.

basename
This function is used to extract the last element from a file path.

capitalize
This function converts it argument to a capitalized string in Title Case.

ceil This function is used to round a number to an integer, towards positive infinity.

change_number
This function is used to determine the change number. It may be set by the **−Change** command line option, or it may default. The return value is an integer.

change_number_set
This function maybe used to determine if the change number was set by the **−Change** command line option. The return value is a boolean.

columns This function is used to define the report columns. Each argument is a structure containing some or all of the following fields:

left	the left margin, counting characters from 0 on the left
right	the right margin, plus one
width	the width in characters, defaults to 7 if right not specified
padding	white space between columns, defaults to 1 if not set
title	the title for this column, separate multiple lines with \n

The columns must be defined before the *print* function is used.

count This function is used to count the number of elements in a list or array.

dirname This function is used to extract all but the last element from a file path.

downcase
This functions converts its argument to lower case.

eject This function is used to start a new page of output.

floor This function is used to round a number to an integer, towards negative infinity.

getenv This function is used to get the value of an environment variable. Will return the empty string if not set.

gettime This function is used to parse a string to produce a time. It understands a variety of different date formats.

getuid This function takes no arguments, and returns the user ID of the process which invoked the report generator. The return value is an integer.

keys This function may be given an array or a list as argument. It returns a list of keys which may be used to index the argument. Most often seen in for loops.

length This function is used to find the length of a string.

mktime This a synonym for the *gettime* function.

mtime This function may be used to obtain the modification time of a file.

need This function is used to insert a page break into the report if the required number of lines is not available before the end of page. If sufficient lines are available, only a single blank line will be inserted. The return value is void.

now This function takes no arguments, and returns the current time.

page_length
This function may be used to determine the length of the output page in lines. The return value is an integer.

page_width
This function may be used to determine the width of the output page in columns. The return value is an integer.

print This function is used to print into the defined columns. Columns will wrap around.

project_name
This function is used to determine the project name. It may be set by the **−Project** command line option, or it may default. The return value is a string.

project_name_set
This function maybe used to determine if the project name was set by the **−Project** command line option. The return value is a boolean.

quote_html
This function quotes its argument string to insulate HTML special characters; these include “less than” (<), “ampersand” (&) and non-printing characters. This is most often used to generate suitable text for web pages.

quote_tcl
This function quotes its argument string to insulate TCL special characters; these include “[]” and non-printing characters. This is most often used to generate suitable text for TCL interface scripts.

quote_url
This function quotes its argument string to insulate URL special characters; these include “?+#:&=” and non-printing characters. This is most often used to generate suitable text for web pages.

round This function is used to round a number to an integer, towards the closest integer.

sort This function must be given a list as argument. The values are sorted into ascending order. A new list is returned.

split This function is used to split a string into a list of strings. The first argument is the string to split, the second argument is the character to split around.

sprintf This function is used to build strings. It is similar to the *sprintf* (3) function.

strftime This function is used to format times as strings. The first argument is the format string, the second argument is a time. See the *strftime*(3) man page for more the format specifiers.

subst This function is used to substitute strings by regular expression. The first argument is the pattern to match, the second argument is the substitution pattern, the third argument is the input string to be substituted. The option fourth argument is the number of substitutions to perform; the default is as many as possible.

substr This function is used to extract substrings from strings. The first argument is a string, the second argument is the starting position, starting from 0, and the third argument is the length.

terse This function may be used to determine of the **−TERse** command line option was used. The return type is a boolean.

title This function is used to set the title of the report. It takes at most two arguments, one for each available title line.

trunc This function is used to round a number to an integer, towards zero.

typeof This function is used to determine the type of a value. The return type is a string containing the name of the type, as described in the

unquote_url
This function will remove URL quoting from the argument string. URL quoting takes the form of a percent sign (%) followed by two hex digits. This is replaced by a single character with the value represented by the hex digits.

upcase This functions converts its argument to upper case.

working_days
This function is used to determine the number of working days between two times.

wrap This function is used to wrap a string into a list of strings. The first argument is the wring to wrap, the second argument is the maxmium width of the output strings.

wrap_html
This function is used to wrap a string into a list of strings. The first argument is the wring to wrap, the second argument is the maxmium width of the output strings. This is very similar to the *wrap* functions, except thatit inserts HTML paragraph breaks <p> or line breaks
 to reflect the newlines within the string (2 or 1, respectively). *TYPES* section.

VARIABLES

There are a number of built-in variables.

arg This variable is a list containing the arguments passed on the *aer*(1) command line.

change
There is a special type of variable created by using an expression similar to *project[project_name()].state.change[n]* which contains all of the fields described in aecstate*(5), plus some extras:*

change Branches have a change array, just like *project* below.

change_number
The number of the change.

config This gives access to all of the fields described in *aepconf(5).*

project_name
The name of the project containing the change.

src This gives access to the change files, and when indexed by file name, yields a value conataining fields as described in *aefstate*(5), for the *src* field.

group This variable is an array containing all of the entries in the */etc/group* file. Each entry is a structure with fields as documented in the *group*(5) manual entry. The *gr_mem* element is a list of strings. This array may be indexed by either a string, treated as a group name, or by an integer, treated as a GID.

passwd This variable is an array containing all of the entries in the */etc/passwd* file. Each entry is a structure with fields as documented in the *passwd*(5) manual entry. This array may be indexed by either a string, treated as a user name, or by an integer, treated as a uid.

project This variable is an array containing one entry for each aegis project, indexed by name. Each array element is a structure, containing

name	the project name
directory	the root of the project directory tree
state	the project state

The project state contains the fields documented in the *aepstate*(5) manual entry. Except: the *change* field is not a list of change numbers, it is an array indexed by change number of change states, as documented in the *aecstate*(5) manual entry. (See *change*, above.)

user This variable is an array containing the *.aegisrc* file of each user. Each entry is a structure with fields as documented in the *aeuconf*(5) manual entry. This array may be indexed by either a string, treated as a user name, or by an integer, treated as a uid. Files which are unreadable or absent will generate an error, so you need to wrap accesses in a try/catch statement. (Note: `count()` and `keys()` functions think the array is empty; if you want a list of users, consult the `passwd` array.)

FILES

The reports are kept in the */usr/local/share/report* directory. The reports are associated with a name by the */usr/local/share/report.index* file. Their names use the command line argument abbreviation scheme, so that report names may be abbreviated.

SEE ALSO

aer(1) report generator

aecstate(5)
change state description

aepstate(5)
project state description

aerptidx(5)
report index file format

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aerptidx − aegis report index file format

SYNOPSIS

```
/usr/local/share/report.index
/usr/local/share/report.local
```

DESCRIPTION

The report index file is used to store pointers to report scripts, and descriptions of the reports.

When searching for a report, the *aer*(1) command searches down the *AEGIS_PATH* looking for *report.index* files, and searching them for the report named.

CONTENTS

where = [{ ... }];

This field is a table relating report name to file name. The structure is as follows:

name = string;

The name of a report. The command line argument naming scheme is used, to provide abbreviatable names.

description = string;

A brief description of the report. It should not be very long, one or two lines at most.

filename = string;

The name of the file containing the report script. If a relative path is given, it will be interpreted to be relative to the directory containing the *report.index* file.

SEE ALSO

aer(1) report generator

aegis(5) aegis file format syntax

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aesub − aegis command substitutions

DESCRIPTION

When other programs are invoked by the *aegis* program, it is usually via a command string in a configuration file. This section describes the format of these command strings.

GENERAL FORM

The command strings are very similar to shell variables. An example will illustrate this:

```
build_command =
   "cook project=${project} change=${change}";
```

In this command definition, the "${project}" part is a substitution: the name of the project will be substituted in the command at this point.

Substitutions may take several forms:

$name
: This is the same as saying "${name}". The name must start with an alphabetic, and be followed by zero or more alphanumerics.

${name}
: The name in this form may contain any non-blank characters, and it may be subject to substitution.

${name *arg*...}
: The name and the arguments in this form may contain any non-blank characters, and it may be subject to further substitution. Within the braces (**{** and **}**) pairs of single quote characters (**'***blah blah***'**) may be used to insulate spaces and other special characters, or you may use the back quote (\) to escape a single character.

$$
: This is replaced by a single $ character. It avoid RCS expansions, you can also use ${$}.

%%
: This is replaced by a single % character. Percent (**%**) followed by anything else is illegal.

$#...\n
: This is a comment, usually found in template files read in using the ${read_file} substitution. It consumes all characters up to and including the next newline. (See also ${comment}, below.)

As another example, the *dirname* substitution is replaced by the directory name of the argument, rather like the *dirname*(1) command. In the command

```
history_query_command =
   "get -t -g ${Dirname $History}/s.${Basename $History}";
```

the *Dirname* and *Basename* substitutions are used to construct a suitable path to the SCCS file in the history directory.

ABBREVIATIONS

The names of the various substitutions may be abbreviated. In the above examples, and in the lists which follow, the minimum abbreviation is the uppercase letters. All substitution name are case insensitive.

The above example could be abbreviated to

```
history_query_command =
   "get -t -g ${d $h}/s.${b $h}";
```

Ambiguous abbreviations will result in a fatal error message.

SUBSTITUTIONS

There are many substitutions which are always understood, and some which are specific to the command being substituted. Specific entries will be defined in the relevant manual section.

The following lists contains those substitutions which are always understood:

Active_Directory
: The absolute path of the change's development directory, if the change is between the *being developed* and *awaiting integration* states. The absolute path of the change's integration directory, if the change is in the *being integrated* state. Not available when the change is in the *awaiting development* or *completed* states. This rather like the default behaviour of the *aecd*(1) command.

Add_Path_Suffix
: This substitution may be used to add a suffix to each element of a colon-separated path list. The first argument is the suffix to use, the second and subsequent arguments are the colon-separated paths to work on. The result is a single colon separated path. Often used in combination with the ${search_path} substitution, below.

Administrator_List
: Space separated list of the project's administrators. Takes an optional argument in the same form as the *user* substitution.

ARCHitecture
: This substitution is replaced by the architecture name appropriate for the current execution environment. Requires no arguments. See the *architecture* field of *aepconf*(5) for more information. When used in commands, you may need to surround this substitution with the *quote* substitution (see below), if any of your architecture names contain shell special characters.

BaseLine
: Absolute path of the the project's baseline.

Basename
: This substitution takes one argument, a pathname. The value of the substitution will be the last element of the pathname. This is similar to the *basename*(1) command.

BAse_RElative
: This substitution takes at least one pathname. The value of the substitution is the base-relative filenames, with any change-specific or project baseline specific leading path removed. The file does not have to be a project source file. (This is almost the inverse of the $source substitution, below.)

BINary_DIRectory
: The absolute path of Aegis' architecture-specific binary (executables) directory. This corresponds to the "*./configure −−bindir*" option when Aegis was built. This is where most of the Aegis executable programs are installed.

CAPitalize
: This substitution takes at least one argument. The value of the substitution will be the arguments with the first letter of each word forced to upper case and the rest forced to lower case.

Change
: This substitution provides various information about the change, based on the argument it is given.

  attribute
  : This substitution takes an additional argument, the name of a change attribute (see *aeca*(1) and *aecattr*(5) for more information). This returns the value listed in the change attributes, or the empty string if the change does not have the named attribute.

  cause
  : This returns the cause of the change.

  date *format*
  : This returns the completion date of the change. See *DATE* section for additional arguments.

debian-version
: This returns a Debian-esque version stamp for the change.

rpm-version
: This returns an RPM-esque version stamp for the change.

delta
: This returns the delta number of the change. Only valid for completed changes.

delta_uuid
: This returns the delta UUID of the change, assigned on integrate pass, a globally unique identifier for the state of the project when this change was integrated (different for all repositories). Only valid for *being_integrated* and *completed changes.*

description
: This returns the brief description of the change.

developer
: This returns the name of the developer of the change.

development_directory
: This returns the development directory of the change.

integrator
: This returns the name of the integrator of the change.

integration_directory
: This returns the integration directory of the change.

number
: This returns the number of the change. (This is the default if no argument is given.)

reviewer
: This returns the name of the reviewer of the change.

state
: This returns the state of the change.

uuid
: This returns the UUID of the change.

version
: This returns the version of the change.

Change_Attribute
: This substitution takes exactly one argument. This argument is a name of a change attribute (see *aeca*(1) and *aecattr*(5) for more information). This returns the value listed in the change attributes, or the empty string if the change does not have the named attribute.

Change_Files
: This is replaced by a space separated list of change file names. There are several qualifying arguments you can give to this substitution:

  action
  : You may give one or more file actions (e.g. *modify*). Only files with one of the actions will be returned. By default, all file actions are returned.

  type
  : You may give one or more file types (e.g. *source*). Only files with one of the types will be returned. By default, all file types are returned.

  not
  : Inverts the sense of operations. For example *${change_files not remove}* will return the names of all change files not being removed.

  quote
  : This does not affect which file are selected, but it causes the file names to be quoted if they contain shell meta-characters.

  If you specify both actions and types, only files both qualifiers will be returned. For example *${change_files modify test}* will return only the names of automatic test files which are being modified.

Change_Developer_List
: Space separated list of all the change's developers. Note that this is different than the `Developer_List` substitution.

Change_Reviewer_List
: Space separated list of the change's reviewers since the last *develop end*. Note that this is different than the `Reviewer_List` substitution. Bt using the *review_policy_command* field of the project configuration file this value can have more than one reviewer, because this allows a project to require a change to need to be reviewed more than once before it can be integrated.

COMment
: Inserts exactly nothing; any and all arguments are ignored. Another form of comment is "$#" which extends to the end of the current line.

Copyright_Years
: Inserts a comma separated list of copyright years from the project attributes. This list of years is maintained by *aegis* at integrate begin, and so is only guaranteed to be up-to-date in the'*being integrated*' state. Do not use this substitution in new file templates, it is not guaranteed to be up-to-date in the '*being developed*' state; use the ${date %Y} substitution in new file templates.

  This list contains spaces, so if you use it to build commands, you will probably need to quote, it as well.

DATa_DIRectory
: The absolute path of Aegis' architecture-neutral library directory. This corresponds to the "*./configure −−datadir*" option when Aegis was built. This is where most of the scripts included with Aegis are installed.

DAte
: With no arguments, the output is the current date. If there are arguments, they form a format string. This is similar to the *date*(1) command on many UNIX systems. For a description of the date formats, see the *DATE* section, below.

DELta
: The delta number of the change. This is only available when the change is in the *being integrated* state or the *completed* state.

DEVeloper
: The name of the developer. Takes an optional argument in the same form as the *user* substitution.

DEVeloper_List
: Space separated list of the project's developers. Takes an optional argument in the same form as the *user* substitution. Note that this is different than the `Change_DeveloperList` substitution.

Development_Directory
: The absolute path of the change's development directory. Only available when the change is between the *being developed* state and the *being integrated* state.

DIFF
: The absolute path of the diff command, as discovered when Aegis was built. It tries to locate GNU Diff at build time to provide maximum functionality.

Dirname
: This substitution takes at least one argument, a pathname. The value of the substitution will be everything but the last element of the pathname. This is similar to the *dirname*(1) command.

Dirname_RELative
: This substitution takes at least one argument, a pathname. The value of the substitution will be everything but the last element of the pathname. This is similar to the *dirname* substitution, except that if there are no directory components, it returns dot (".").

DownCase
: This substitution takes at least one argument. The value of the substitution will be the argument with any upper case letters mapped to lower case.

EMail_Address
: This substitution takes one or more user names as arguments. It replaces them with email addresses. (It is an error if any user name is unknown.)

  This substitution takes options. You may specify one or more of them immediately after the substitution name.

  −Comma
  : This option may be used to specify that the email addresses are to be separated by commas.

  −Quote This option may be used to specify that the email addresses are to be quoted to insulate shell special characters.

  See *aeuconf*(5) for where this is set.

ENVironment
: This substitution takes at least one argument. The value of the substitution is the value of the corresponding environment variable, or empty of undefined.

ERrno
: This substitution takes no arguments. The value of the substitution will be the value if the *errno* variable provided by the system, as mapped through the *strerror* function. Thus you may give the users informative system error messages.

EXpression
: This substitution evaluates simple arithmetic expressions. Addition, subtraction, division, multiplication, modulo and negation are understood. The 6 basic comparison operators are available. The usual C syntax and precedence are used. The arguments must constitute a valid expression, white space and word boundaries are ignored.

History_Directory
: This substitution takes zero arguments. It is replaced by the absolute path of the history directory of the project.

History_Path
: This substitution takes one argument, the name of a source file. It is replaced by the absolute path of the history file for that source file. Note that you may beed to massage the file name a little for you proticular history tool, just as the history commands in the *aegis.conf* file do.
  This substitution takes zero arguments. It is replaced by the absolute path of the history directory of the project.

IDentifier
: This substitution takes at least one argument. The value of the substitution will be the argument with all characters but alpha numerics mapped into an underscore (_), so as to form a legal C identifier.

INTegration_Directory
: The absolute path of the change's integration directory. Only available when the change is in the *being integrated* state.

INTegrator
: The name of the change's integrator. Only available when the change is in the *being integrated* state or the *completed* state. Takes an optional argument in the same form as the *user* substitution.

INTegrator_List
: Space separated list of the project's integrators. Takes an optional argument in the same form as the *user* substitution.

LEFt This substitution extracts the left hand side of strings. It takes two arguments: the first is the string, the second is the number of characters.

LENgth This substitution determines the length of strings, the result is a number. It takes one argument: the string to be measured.

LIBrary
The absolute path of Aegis' library directory. This corresponds to the "*./configure −−datadir*" option when Aegis was built. This substitution is deprecated − please use ${datadir} instead.

LIBrary_DIRectory
The absolute path of Aegis' architecture-specific library directory. This corresponds to the "*./configure −−libdir*" option when Aegis was built.

Name_Maximum
This substitution is used to get the maximum file name length within a file system. It takes at least one argument: the name of a directory within the file system. Frequently used with ${left} to crop filenames to the file system maximum.

PAth_Reduce
This function requires at least one argument. It is used to remove duplicates from a command search path, such as may be found in the PATH environment variable. If more than one argument is given, all are included in the results, as if they were separated by colons.

PERL This function requires zero arguments. It is replaced by the absolute path of a Perl interpreter.

PLural
This function requires 2 or 3 arguments. The first argument is evaluated as a number, if it is plural (not equal to 1) the second argument is the result, otherwise the third argument is the result (or empty if not given). This is mostly used to pluralize sentences in Germainic error messages.

PLural_Forms
The *plural_forms* substitution is similar to the *${plural}* substitution, except that it reads and understands the `Plural-Forms:` header in the message catalogue. This means that it understands a greater range of pluralization mechanisms than the simple *${plural}* substitution. (For a description of the `Plural-Forms:` header, see the GNU Gettext manual.)

The first argument is the number. Second is the singular form (corresponding to the `Plural-Forms:` expression evaluating to zero), the third and subsequent arguments are the various plural forms (corresponding to the `Plural-Forms:` expression evaluating to 1, 2, 3, *etc*.

The `Plural-Forms:` expression is required evaluate to less than `nplurals`. If it does not, the second argument (the singular form) is used. If there are too few arguments to this substitution, the second argument (the singular form) is again used.

Note that in the default case (used for English and other Germanic languages), the arguments are the *reverse* of those expected by the *${plural}* substitution.

Project
This substitution provides various information about the project, based on the argument it is given.

name This returns the name of the project. (This is the default if no argument is given.)

description
This returns the description of the project (the one which appears in the project listing).

trunk_name
This returns the name of the trunk of the project (i.e. no branch numbers included).

trunk_description
This returns the description of the trunk of the project.

version This returns the version of the project.

version_long
This returns the version of the project, including the delta number.

Project_Specific
This substitution takes exactly one argument. This argument is a name to be found in the project configuration file's *project_specific* field (see *aepconf*(5) for more information). This returns the value listed in the project configuration file. Unknown attributes will be replaced with the empty string.

QUote
This substitution may be used to quote shell special characters. If no quoting is required, not quotes will be inserted. This is used to insulate shell special characters in filenames when forming commands.

Read_File
Read a file and substitute the contents of the file. Requires exactly one argument, the pathname of the file to be read. If the pathname is a project source file, you will need to use the *source* substitution to resolve the path. It is a fatal error if the file does not exist, or is not readable. It is a fatal error if the pathname is not absolute (because the current directory is undefined).

Read_File_Simple
Read a file and without substituting the contents of the file. Requires exactly one argument, the pathname of the file to be read. If the pathname is a project source file, you will need to use the *source* substitution to resolve the path. It is a fatal error if the file does not exist, or is not readable. It is a fatal error if the pathname is not absolute (because the current directory is undefined).

Reviewer
The name of the change's reviewer. Only available when the change is between the *awaiting integration* state and the *completed* state. Takes an optional argument in the same form as the *user* substitution.

Reviewer_List
Space separated list of the project's reviewers. Takes an optional argument in the same form as the *user* substitution. Note that this is different than the `Change_Reviewer_List` substitution.

RIght This substitution extracts the right hand side of strings. It takes two arguments: the first is the string, the second is the number of characters.

Search_Path
The Search_Path substitution is replaced by a colon separated list of absolute paths to search when building a change, it will point from a change to its branch and so on up to the project trunk.

Search_Path_Executable
The Search_Path_Executable substitution is usually the same as the Search_Path substitution. However, during an "aegis −Test −BaseLine" command, it contains the baseline as the first element, rather then the development directory or the integration directory. This is of most use when looking for executables and executable support files while running tests.

SHell
The absolute path of a Bourne shell which understands functions. Requires exactly zero arguments.

Source
Resolve the argument filename into a pathname. It is an error if the file is not a source file. An optional second argument may be "Absolute" or "Relative", and may be abbreviated. Relative will attempt to provide a development-directory-relative pathname whenever possible, absolute will always result in an absolute path. The default is "Relative". (For the inverse mapping, see *${BAse_RElative}*, above.)

SPLit This substitution may be used to split strings are specified separators. The first argument is the separator character to be used, subsequent arguments are strings to be split. The result is the collection is split strings of the second a follwoing arguments, separated by spaces.

STate
The state the current change is in. It is an error if the substitution does not refer to a change.

SUBSTitute
Regular expression substitution. The first argument is the pattern to match, the second argument is the replacement string. The third and subsequent arguments are modified as specified by the first two arguments. The search is not anchored, and the replacement will happen as many times as possible. Use “^” to match the beginning, and “$” to match the end.

SUBSTRing
This substitution extracts a substring from the middle of strings. It takes three arguments: the first is the string, the second is the star character (counting from zero), the third is the number of characters.

SWitch
Select amongst a set of values. The first argument is expected to be a number. If the number is zero, the second argument is used; if the number is one, the third argument is used; etc. If the number is negative, or exceeds the available arguments, the last argument is used.

Trim_DIRectory
This substitution takes one or two arguments. If given one argument, one directory component (if present) is removed from the argument, which is assumed to be a file name. If two arguments are present, the first is a directory count; at most this many directory components (if present) will be removed. The base file name is always left.

Trim_EXTension
This substitution takes one argument. Any file name extension (a dot characters and the characters following) will be removed from the final filename section of the argument.

UNSplit This substitution may be used to reverse the effects of the *split* substitution. The first arguments is a seaparator character, the second and following arguments are strings to be joined together using the separator character. The result is a single string.

UpCase
This substitution takes at least one argument. The value of the substitution will be the argument with any lower case letters mapped to upper case.

USer
This substitution provides various information about the user who executed the command, based on the argument it is given.

login The login name of the user. (This is the default if no argument is given.)

name The full name of the user.

email The email address of the user.

quoted_email
The email address of the user, quoted to avoid shell special characters.

home The home directory of the user.

Version
The version of the change. If the change is in the *being integrated* state or the *completed* state, the version will be of the form "*a.b.***D***ddd*", where "a" is the project’s major version number, "b" is the project’s minor version number, and "ddd" is the change’s delta number. If the change is in any other state, the version will be of the form "*a.b.***C***ccc*", where "ccc" is the change number.

delta_uuid
: This variant gives the change's delta-UUID assigned at integrate pass. Only valid for *being_integrated* and *completed* changes.

Zero_Pad
: This substitution is used to zero pad a string on the left. It takes two arguments: the first is the string to be padded, the second is the minimum string width.

DATE

This section describes the format specifiers accepted by the date substitution. These are the same specifiers as defined by the ANSI C standard for the strftime function.

%%	The percent character (%)
%a	the abbreviated weekday name
%A	the full weekday name
%b	the abbreviated month name
%B	the full month name
%c	the date and time
%d	the day of the month, zero padded
%H	the hour of the 24-hour day
%I	the hour of the 12-hour day
%j	the day number of year, zero padded, one based
%m	the month of the year, zero padded, one based
%M	the minute of the hour, zero padded
%p	meridian indicator, AM or PM as appropriate
%S	the second of the minute
%U	the Sunday week of the year
%w	the day of the week, Sunday is 0
%W	the Monday week of the year
%x	the date, as *mmm dd yyyy*
%X	the time, as *hh:mm:ss*
%y	the year of the century
%Y	the year including the century
%Z	time zone abbreviation

Using an undefined format specifier will produce random results, depending on the version of UNIX you are on.

SEE ALSO

aesub(1)
: Substitute and print strings.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aetest − aegis test results file format

DESCRIPTION

The default configuration of the *test_command* and *development_test_command* commands of the project *config* file (see *aepconf*(5) for more information) is for the test commands to test a single test file and a single architecture at a time. On some systems, this is not efficient.

When configured to run multiple simultaneous tests, or multiple simultaneous architectures, a file of the format described here is used to communicate the test results back to Aegis.

CONTENTS

Use a separate row for each unique filename and architecture combination.

test_result = [{ ... }];

All the fields are mandatory.

file_name = string;

This is the name of the file being tested. Use the same filename as was given to the test command.

exit_status = integer;

This is the exit status returned by the test.

architecture = string;

This is the architecture which the test was run on. Defaults to the architecture of the current system if not set. (Try to avoid setting this field unless you have a very clever multi-architecture test system.)

SEE ALSO

aet(1) run tests

aegis(5) aegis file format syntax

aepconf(5)

Project configuration file format.

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeuconf − user configuration file

SYNOPSIS

$AEGIS_FLAGS
$HOME`/.aegisrc`
/usr/local/share/aegisrc
/usr/local/lib/aegisrc

DESCRIPTION

A user configuration file is used to hold user defaults. This file is created and edited by the user. This file is only ever read by aegis, it is never written.

The sources of user preferences are scanned in the order given above. Earlier sources have higher priority.

AEGIS_FLAGS

This environment variable has the same format. It is read first, and over-rides the *.aegisrc* file contents. This is intended to be used within the tests distributed with aegis, but can also be of use within some shell scripts. It contains session specific preferences.

$HOME`/.aegisrc`
This file contains user specific preferences.

/usr/local/share/aegisrc
This file contains architecture-neutral preferences.

/usr/local/lib/aegisrc
This file contains architecture-specific preferences.

CONTENTS

The file contains the following fields:

default_development_directory = string;
The pathname of where to place new development directories. The pathname may be relative, in which case it is relative to *$HOME*. The default is the field of the same name in the project attributes, or *$HOME* neither is set.

default_project_directory = string;
The pathname of where to place new project directories. The pathname may be relative. If this path is relative, it is relative to *$HOME*. The default is *$HOME*.

delete_file_preference = (no_keep, interactive, keep);
All of the commands which delete files will consult this field to determine if the file should be deleted. Defaults to *no_keep* if not set.

default_project_name = string;
The name of a project.

default_change_number = integer;
The number of a change.

Please note that the *default_project_name* field and the *default_change_number* field are unrelated. Specifying both does not mean that single change within that single project, they have nothing to do with each other.

diff_preference = (automatic_merge, no_merge, only_merge);
The *aed(1)* command will consult this field to determine what to do:

no_merge
means only diff the files, even if some have out of date versions.

only_merge
means merge those files with out of date versions, and do not do anything else, even if they need to be diffed.

automatic_merge
means to do *only_merge* if any source files require merging, otherwise do *no_merge*. It never combines merges and differences in the same pass.

The corresponding command line options to the *aed*(1) command take precedence, this field is only consulted if you do not give a corresponding command line argument. Defaults to *automatic_merge* if not set.

pager_preference = (foreground, never);
This field is consulted for listings and help. The standard output is only piped to a pager if the command is run in the foreground and the standard output is directed at a terminal.

foreground
The standard output will be piped through the command given in the PAGER environment variable (or *more* if not set).

never The standard output will not be redirected.

This field defaults to *foreground* if not set.

persevere_preference = (all, stop);
This field is consulted by the *aet*(1) command, to determine if it should run all tests, or stop after the first failure. This field defaults to *all* if not set.

log_file_preference = (snuggle, append, replace, never);
This field controls the behavior of the log file. It usually defaults to *snuggle* if not set, although some commands may default it to *append*. When the log file is in use, the output continues to be sent to the screen if the process is in the foreground and the standard output is a terminal.

never Do not redirect the output to a log file.

replace Replace any log file that is present, create a new one if none already exists.

append Append the log to the end of any existing log file, create a new one if none already exists.

snuggle Append the log to the end of any existing log file if that log file was last modified less than 30 seconds ago, otherwise replace any existing log file; create a new one if none already exists. This option allows runs of aegis commands to produce a meaningful log file.

lock_wait_preference = (always, background, never);
This field is consulted by all commands which wait for locks.

always The “always” setting says that all commands should always wait for locks. This is the default.

background
The “background” setting says that background commands should always wait for locks, and foreground commands will not.

never The “never” setting says that no command should ever wait for locks. If the command would wait, it will exit with status 1.

This user preference can be over-ridden by the **−wait** and **−nowait** command line options.

symbolic_link_preference = (verify, assume);
This field is consulted by *aeb*(1) when the project configuration file specifies *create_symbolic_links_before_build* as true. The verification of the links can be quite time consuming; if you are confident that they are already correct (say, from a recent build run) you may wish to assume they are correct and not verify them repeatedly.

verify This setting says to always verify the symbolic links to the baseline. This is the default.

assume This setting says to always assume the links are correct, unless there has been a recent integration.

This user preference can be over-ridden by the **−Verify_Symbolic_Links** and **−Assume_Symbolic_Links** command line options.

relative_filename_preference = (current, base);
: This field is consulted by most commands which accept filenames on the command line. It controls whether relative filenames are relative to the current directory (this is the default), or relative to the base of the project source tree.

  current This setting says to interpret relative filenames against the current directory.

  base This setting says to interpret relative filenames against the base of the source tree.

  This user preference can be over-ridden by the **−BAse_RElative** and **−CUrrent_RElative** command line options.

email_address = string;
: This field may be used to set the preferred email address. If not set, defaults to `‘whoami‘@‘cat /etc/mailname‘` if not set, and if */etc/mailname* exists. Otherwise, defaults to `‘whoami‘@‘hostname‘` if not set, which is usually not what is required, particularly if you are behind a firewall.

whiteout_preference = (always, never);
: All of the commands which cause a change to remove files will consult this field to determine if the file should be have a dummy “whiteout” file put in the development directory. Defaults to “always” if not set.

editor_command = string;
: This command is used to edit a file, if the editing is being done in the background. Defaults to the EDITOR environment variable if not set, or “ed” if not set.

visual_command = string;
: This command is used to edit a file, if the editing is being done in the foreground. Defaults to the VISUAL environment variable if not set, or to the EDITOR environment variable if not set, or “vi” if not set.

pager_command = string;
: This is the command used to paginate report and listing output. Defaults to the PAGER environment variable if not set, or to “more” if not set.

attribute = [{ ... }];
: This is a list of *(name,value)* pairs, defining user specified attributes.

  name = string;
  : The name of the attribute. By convention, names which start with an upper-case letter will appear in listings, and lower-case will not. Attribute names are case-insensitive.

    Arguably, most user attributes which may be altered by the user (and some that can’t) should be of this form. Due to an accident of history, this is not the case.

    The attributes known to Aegis are:

    progress-preference
    : boolean; true if *aet*(1) should emit progress messages, false if not. Can be overridden with the **−progress** and **−no-progress** command line options.

    FIXME: there needs to be a *aesub*(5) way to get at these values.

  value = string;
  : The value of the attribute.

SEE ALSO

aegis(5) aegis file format syntax

aed(1) difference and merge files

COPYRIGHT

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/

NAME

aeustate − aegis user state file

SYNOPSIS

`/usr/local/com/user/`*user-name*

DESCRIPTION

A user state file is used to store information about a user. These file are created and maintained by aegis. These file should not be edited by humans.

CONTENTS

own = [{ ... }];

This field is a list of change the user is currently working on, within project. The changes are in either the *being_developed* or *being_integrated* state. The structure of this field is as follows:

project = string;

The name of a project.

change = [integer];

The changes this user is working on in the project.

SEE ALSO

aegis(5) aegis file format syntax

COPYRIGHT

aegis version 4.25.D510

AUTHOR

Peter Miller E-Mail: pmiller@opensource.org.au
/\/* WWW: http://miller.emu.id.au/pmiller/