

Silicon Graphics Inc

**IRIS WORKSTATION
COURSE
68020 Based**

STUDENT'S WORKBOOK

Publication Number: 0002 R5 A

August 5, 1987

REVISION NOTICE

This is the Fifth revision of the Student Workbook.

READER COMMENT FORM

A User's Comment Form or Class Survey is provided at the end of this publication. Comments and suggestions may be sent to: Silicon Graphics, Inc., Training Dept, 2011 Stierlin Rd, Mountain View, CA, 94043.

RESTRICTION ON USE

The information contained in this manual is the property of Silicon Graphics, Inc. The contents of this manual may not be reproduced, distributed, or sold without the written consent of the Silicon Graphics Training Department.

This manual has not been published or otherwise placed in the public domain.

NOTICES

1986 Silicon Graphics, Inc.
All right reserved

IRIS, Geometry Link, Geometry Partners, Geometry Accelerator, Geometry Engine, are registered trademarks of Silicon Graphics, Inc.

The contents of this publication are subject to change without notice.

COURSE OUTLINE

HARDWARE MAINTENANCE COURSE IRIS 68020 BASED WORKSTATIONS

I. COURSE DESCRIPTION.

This course aims at producing a person who is proficient in the skills necessary to perform first line maintenance on a Silicon Graphics 2400 Turbo, 3010, 3020, 3030 and 3130 workstations.

These skills include: An understanding of basic data flow from a level of system block diagrams; how to locate, load, and run any available diagnostic; how to perform mechanical and electronic adjustments; how to edit system files; how to perform software backups and various system administration functions; and finally, how to locate specific information from the supporting documentation.

II. EQUIPMENT.

The IRIS 2400T, 3010, 3020, 3030 and 3130 workstations combine the computing power of the MC 68020 with *real-time* 3-D color graphics in a workstation ideal for applications requiring fast, high-resolution graphics. These workstations bring the user fast response, large amounts of available memory, extensive computational power, multi-windowing, multi-tasking and powerful *real-time* high-resolution graphics.

These workstations come in a package accommodating up to 16 MB of memory, up to 32 bit-planes of display memory, and several sizes of Winchester disks. They are capable of displaying 4096 colors simultaneously, double buffered, or from a palette of 16.7 million colors using full color mode. Gouraud shading, Z-buffering, and depth cuing, aid in the display of graphical objects.

These workstations operate as a personal workstation or as a node in a networked workstation environment.

III. SCOPE.

A. This course is 12 days in length and is divided into the topics listed below:

1. Course Introduction.
2. Workstation Introduction.
3. UNIX: Survival Skills and VI Editor.
4. System Administration.
5. Component Location.
6. Data Flow: Theory of Operation.
7. Troubleshooting Lab.
8. Adjustments.

B. The students are required to complete several lab projects, quizzes, and tests. These are given as reviews and reinforcers of the students knowledge and learning. They are used to inform the instructor and the student of the progress toward obtaining the lesson objectives.

C. Lab bugs, exams, and performance problems will be used to determine if the course objectives have been met.

D. The theory-of-operation is taught using block diagrams; with emphasis placed on the data flow.

E. Appendix A, of this outline, lists the day-to-day, hours-per-lesson schedule for the course.

IV. COURSE OBJECTIVES.

At the completion of this course the student shall be able to:

- A. Working unassisted, isolate an induced failure to a single Field Replaceable Unit (FRU) within one hour of the failure.
- B. Correctly name and explain the use and contents of each IRIS Workstation manual that is available as supporting documentation.

- C. Demonstrate that he/she has developed basic UNIX operating skills that will allow the student to gather required status information, monitor system activity, create system backup tapes, rebuild/restore system disk drives, edit system files, and run high level system diagnostics and demo programs.

Evidence that this objective is being met will come from instructor observation of the student and successful completion of several tests and lab projects that each student must perform.

V. LESSON OBJECTIVES.

Each lesson will focus on specific areas of the IRIS workstation. The student, using any available documentation and /or tools, will be able to meet or perform the stated objectives for each of the following lessons.

- A. **Lesson 1: Course Introduction.** Upon completion of this lesson, the student will be able to describe to his/her own satisfaction, the following:
1. What will be taught during this course.
 2. How the course will be presented.
 3. How he/she is expected to perform.
- B. **Lesson 2: Workstation Introduction.** Upon completion of this lesson, the student will be able to describe to his/her own satisfaction, the following:
1. An overview of the IRIS workstation operation and hardware configuration using block diagrams.
 2. Name and explain the use and contents of each IRIS workstation manual.
- C. **Lesson 3: UNIX: Survival Skills and VI Editor.** Upon completion of this lesson, the student, using available documentation and an IRIS workstation will be able to:
1. Given a list of UNIX commands, correctly match the commands to explanations and use of each command, then demonstrate (on the workstation) that each command can be correctly executed.
 2. Demonstrate that specific system files can be located and modified using the VI editor.

3. Demonstrate that specific IRIS demonstration programs can be located and correctly run.
 4. Correctly draw the UNIX directory/sub-directories tree.
- D. **Lesson 4: System Administration.** Upon completion of this lesson, the student, using available documentation and an IRIS workstation will be able to perform the following:
1. Create a Bootable tape.
 2. Create backup tapes of /ROOT and /USR files systems.
 3. Using the Bootable tape, rebuild a trashed disk drive and restore the operating system.
 4. Install software options onto the existing system.
 5. Add a new account, creating and editing all required files.
 6. Demonstrate how to add the following equipment to the existing system: ASCII Terminal, modem, printer, enabling Ethernet.
- E. **Lesson 5: Component Location.** Upon completion of this lesson, the student, using available documentation will be able to:
1. Locate, remove, and re-install the following FRU's: Power supply, Winchester disk, Quarter inch tape drive, CPU board, Tape/Disk controller board, Frame buffer controller board, Update controller board, Display controller board, Bit plane boards, and memory boards.
 2. Locate specific hardware configuration switches and demonstrate that the switches can be set to select or enable a stated function.
- F. **Lesson 6: Data Flow: Theory of Operation.** Upon completion of this lesson, the student, using available documentation will be able to:
1. Name each one of the boards in the IRIS workstation; stating the purpose of each board and correctly indicate where that board fits into the overall block diagram of the system.
 2. Be able to name the major subsystems of the IRIS workstation.

G. **Lesson 7: Troubleshooting Lab.** The objective of this lesson is to allow each student to apply the knowledge learned and reinforce that learning by diagnosing induced failures on the training workstations.

1. Each student must demonstrate that he/she can locate a majority of the lab bugs using any available documentation and/or tools.
2. Each student must be able to locate, unaided by other students and within **one hour**, specific failures known as performance problems. He/she may ask general questions to collect necessary information (concerning the problem) from the lab instructor.
3. Each student must demonstrate that specific diagnostics can be located, loaded, and correctly run.

H. **Lesson 8: Adjustments.** Upon completion of this lesson, the student, using available documentation and test equipment, must correctly perform the required adjustments to the IRIS color monitor and workstation power supply.

The adjustments must be performed per current adjustment procedures and the equipment must be left in normal operational condition once the adjustments have been made.

VI. PREREQUISITES.

A. The students must meet the following minimum requirements:

1. Graduate of an accredited electronics school or military electronics school.
2. 2-4 years of electronics experience.
3. 1-2 years of field engineering experience in the computer industry.

B. It is recommended that each student obtain and read the following books:

1. SAMS.
Computer Graphics
User's Guide
By Andrew S. Glassner
2. SAMS
UNIX Primer Plus
By Mitchell Waite, Donald Martin, and Stephen Prata.

or

The UNIX Operating System
By Kaare Christian
John Wiley and Sons Publication

VII. REFERENCES.

- A. Student Workbook.
- B. IRIS workstation manuals.
- C. Course Instructor.

VIII. PERFORMANCE.

The prime objective of this course is to ensure that each student receives the minimal basic skills required to locate a majority of hardware failures within a reasonable time period.

In order to evaluate each student's progress at satisfying the course and lesson objectives, tests, lab projects, lab bugs, and performance problems are given to each student.

Final grading (of the student) will be a combination of all the above mentioned evaluation methods, with the greatest emphasis placed on performance problems, lab performance, and instructor evaluation.

A final composite grade of 80% or better is required by each student in order to receive a completion certificate. The instructor will also submit a written evaluation of each student. A copy of the evaluation form is included with this outline as Appendix B.

**APPENDIX A
STUDENT EVALUATION**

A. Student name:

B. Immediate supervisor:

C. Composite score =

The composite score is calculated by adding together the average scores of all quizzes, tests, and exams (which are objective scores), to the lab score and the instructor evaluation score; the later two scores being subjective scores. The total score is then divided by a factor of 15. The factor is equal to the combined weights of all five of the evaluation methods. (See example scores of two students in chart below)

The sample scores shown in each column of the chart would represent the average score of all the quizzes, tests, exams, and lab projects completed by the student.

Weight	Quizzes 1	Tests 2	Exams 3	Lab 4	Instructor 5 = 15
Student 1	72	68	75	85	90
Student 2	90	87	91	65	65

Student 1
 $1 \times 72 = 72$
 $2 \times 68 = 136$
 $3 \times 75 = 225$
 $4 \times 85 = 340$
 $5 \times 90 = 450$

 Composite score = 1223 divide by 15 = 81.5

Student 2 1 x 90 = 90
 2 x 87 = 174
 3 x 91 = 273
 4 x 65 = 260
 5 x 65 = 325

 Composite score = 1122 divide by 15 = 74.8

The composite score ranks each student in relation to the other students attending the course.

D. Student critique:

NOTE: If "no" is selected for any of the following questions, the instructor must make a written comment in section E of this evaluation as to why.

- | | | |
|---|-----|----|
| 1. Did the student meet all the course objectives? | YES | NO |
| 2. Is the student capable of maintaining the workstation? | YES | NO |
| 3. Did the student actively participate in class, display a positive attitude, and strive to meet the objectives? | YES | NO |
| 4. Did the student complete all assigned projects? | YES | NO |
| 5. Did the student spend additional time (off hours) working on the system. | YES | NO |
| 6. Did the student complete and turn in the course critique? | YES | NO |
| 7. Did the student return all 1/4" tape cartridges and classroom IRIS reference manuals? | YES | NO |

Silicon Graphics, Inc. Education Center

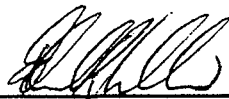
Certificate of Completion

PARKS FIELDS

has successfully completed

3000 HARDWARE MAINTENANCE CLASS

on this day of 27 MAY, 1988



Ed McCracken, President



Instructor

TEST 1

NAME:

1. What directory in UNIX holds the majority of the configuration files?

A. etc

B. dev

C. config

D. usr

2. What UNIX command would you use to *rename* a file? P 27

A. cp

B. mv

C. ln

D. chgname

3. You need to know **where** in the UNIX tree a file called xyz is located so that you can edit it. Show the complete command that you would use to get the file's (xyz) pathname.

find / -Name xyz -PRINT

4. A file called abc has the following permissions set for it: 666. You want to change the permissions to allow the following:

Owner = Read and Write permission.

Group = Read only.

Other = Read only.

Show the complete command that you would use to set these permissions.

chmod 644 abc

5. What *C-shell* environmental is used to set the search-rule that the command interpreter uses to look for commands that the user enters?

PATH

6. What section of the Programmer's Manual would you use to obtain information concerning the Berkeley Shell environmental, **PATH**?

A. Vol. 1A

B. CSH(1)

C. SH(1)

D. Vol.1B

7. Show the command and its arguments that you would use to copy a file from a host named *julie* to your workstation. The file pathname is *~mark/folder/records/xyz*. (Show where any spaces in the command syntax would go using the up arrow character)

XCP julie: ~mark/folder/records/xyz ^ xyz

8. What command was recommended for use when you want to look at the status of all jobs running on your workstation?

PS -AUX

9. You want to kill a job. What command would you use to insure that a specific job is killed?

Kill ^ -9 ^ PID

10. What keystroke do you use to open a new line below the cursor and enter insert mode for the Vi editor?

O

11. You want to position the cursor at the line that contains the character string ABC. Show the Vi command that you would use.

 /ABC

12. You want to write the contents of your Vi buffer to another file, show the command that you would use.

 !W File Name

13. What command would you use to delete from the point in a line where you positioned the cursor, to the end of the line?

 d) or D

14. Show the PROM monitor command (using proper syntax) that you would use to boot your workstation over the Ethernet network. The host name is *elvin* and the boot file is *vmunix*, which is located in the directory */usr/people*.

~~XXS.~~ ELVIN: /usr/people/vmunix

P52

 n . elvin

15. You are using the C-shell (csh). When a user logs into an account, what two files are accessed by UNIX to obtain instructions for setting variables and parameters for the shell just opened?

 .login .cshrc

*Profile is same as .login
 .cshrc*

Contents

LESSON 1: COURSE INTRODUCTION 1

LESSON 1: COURSE INTRODUCTION

Upon completion of this lesson, the student will be able to describe to his/her own satisfaction, the following:

- What will be taught during this course.
- How the course will be presented.
- How he/she is expected to perform.

Contents

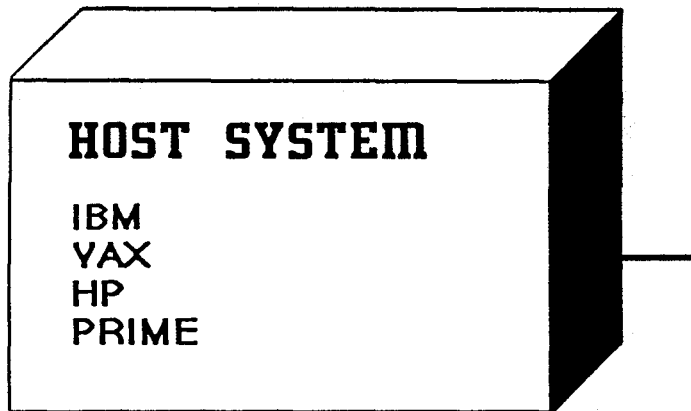
LESSON 2: WORKSTATION INTRODUCTION	1
SYSTEM OVERVIEW	2
IRIS WORKSTATION SUBSYSTEMS	3
IRIS WORKSTATION OVERVIEWS	4
2400 TURBO SPECIFICATIONS	5
3010 SPECIFICATIONS	6
3020 SPECIFICATIONS	7
3030 SPECIFICATIONS	8
WORKSTATION SOFTWARE	9
BASIC SYSTEM BLOCK DIAGRAM	10
SUPPORT DOCUMENTATION - page 1	11
SUPPORT DOCUMENTATION - page 2	12

LESSON 2: WORKSTATION INTRODUCTION

Upon completion of this lesson, the student will be able to describe to his/her own satisfaction, the following:

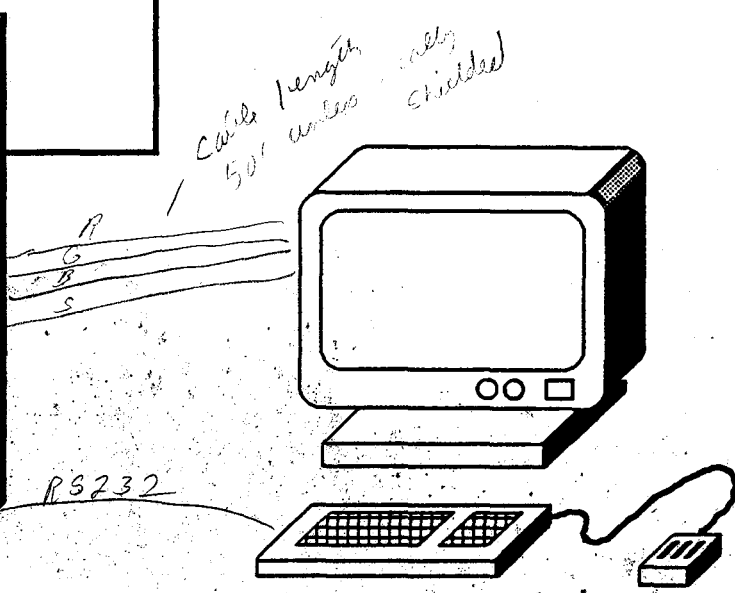
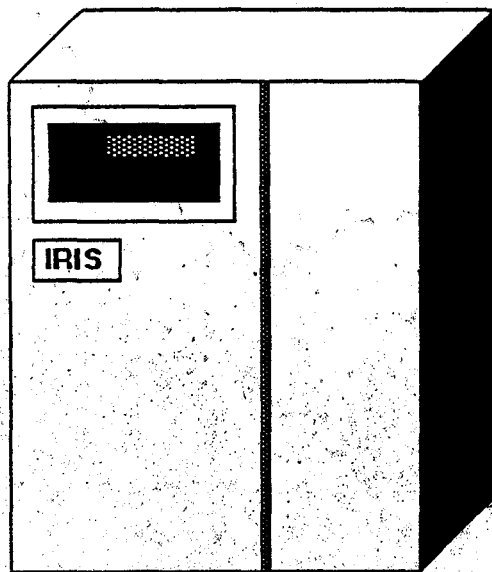
- An overview of the IRIS workstation operation and hardware configuration.
- Name and explain the use and contents of each IRIS workstation manual.

SYSTEM OVERVIEW



COMMUNICATIONS:

- RS-232C *Serial*
- ETHERNET *TCP/IP*
- IEEE-488 *XNS*
- HYPER-CHANNEL *-Cray*
- IBM 327X



IRIS WORKSTATION SUBSYSTEMS

The data flow portion of an IRIS workstation is divided into four subsystems. Each subsystem is comprised of one or more printed circuit boards (PCB).

1. CENTRAL PROCESSING SUBSYSTEM

- 68020 Processor
- Memory (up to 16 MB)
- Floating Point Accelerator (optional)

2. GEOMETRY PIPELINE SUBSYSTEM

- 2 Geometry Accelerators *> VLSI chips*
- 12 Geometry Engines

3. RASTER SUBSYSTEM

- Framebuffer Controller
- Update Controller
- Display Controller
- Display Memory (up to 8 boards)

4. I/O SUBSYSTEM

- Tape/Disk Controller
- Ethernet Controller
- Hyper-Channel (optional)
- IEEE-488 (optional)
- IBM 327X Interface (optional)
- Color Printer Controller (optional)

IRIS WORKSTATION OVERVIEWS

The following gives general information about each of the IRIS products that use the 68020 CPU. The next four pages in your student workbook give the total specifications for all four workstations.

- **IRIS 3010 TERMINAL**

The IRIS 3010 Terminal is a high-performance, high-resolution, UNIX-based terminal for real-time 3-D color graphics and computing applications. The IRIS Terminal utilizes the 32-bit Motorola 68020 CPU and can function as either a **host-dependent terminal** or an **execute-only workstation**. When functioning as a host-dependent terminal, the applications program resides in the host computer system. The IRIS 3010 can be linked to an IRIS Workstation or a variety of other host systems via an RS-232C link or optionally via Ethernet. As an execute-only workstation, programs can be developed, debugged, and tested on an IRIS 3020 Workstation or an IRIS 3030 Workstation and then loaded and executed, without modification, on a stand-alone 3010 Terminal.

- **IRIS 2400 TURBO and 3020 WORKSTATIONS**

These workstations are high-performance, UNIX-based engineering workstations with a 72 MB disk drive and controller designed for real-time 3-D color graphics and computing applications. They utilize the 32-bit Motorola 68020 CPU and can operate as a stand-alone personal workstation, a node in a networked workstation environment, or integrated with other computer systems via Ethernet.

The 2400 Turbo is an upgrade of the IRIS 2400 workstation. It is the same system as the IRIS 3020 except for the following areas: Cabinet, Power supply, Chassis, and Monitor.

- **IRIS 3030 WORKSTATION**

The IRIS 3030 workstation is the same as the 3020, but it has a 170 MB disk drive.

2400 TURBO SPECIFICATIONS**NOTE:**

The spec sheet shown below is for a 3020. The differences are in the cabinet, power supply (turbo has a 600 watt supply), and monitor (turbo standard monitor is 15").

System Specifications**Processors:**

- 16 MHz MC68020 central processor 32-bit internal registers, 32-bit address space
- 10 125-nanosecond Geometry Engines
- 16-bit bit-slice frame buffer controller
- Independent microcoded display processor with 32 KB memory for fonts, textures, and cursors

CPU Memory:

- 2 MB dynamic RAM with parity error detection, expandable to 16 MB
- 256 KB EPROM for hardware initialization, self-configuration, and diagnostics, expandable to four 256/512 KB EPROMS

Image Memory:

- 8 1024 x 1024 bit-planes standard, expandable to 32 with 16-bit Z-buffer

Video Interface:

- RGB levels 0.7 V p-p into 75 ohms
- Separate composite 2 V p-p sync into 75 ohms
- 60 Hz non-interlaced 1024 x 768 resolution frame
- Other frame resolutions and rates available
 - 33 Hz interlaced 1024 x 768
 - 30 Hz interlaced 636 x 485
 - 25 Hz interlaced 768 x 575
- Genlock available with 485 and 575 visible line frames

Color Range:

- Color map mode (8- or 12-bit, single or double buffered)
- 4096 simultaneous colors, double buffered, displayable from palette of 16.7 million
- RGB mode (24-bit), 16.7 million colors displayable

Standard Peripherals:

- 72 MB unformatted 5.25" Winchester disk drive using an ST-506 interface
- 83-key up-down encoded keyboard with user definable keys
- 19" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor
- Optical mouse X-Y encoder with three buttons

Communications:

- Ethernet local area network with XNS software
- Four RS-232C ports for keyboard and serial communications (up to 19.2K baud)

Standard Software:

- UNIX System V operating system with Berkeley 4.2 and local enhancements
- C compiler and development environment
- IRIS Graphics Library II
- IRIS Window Manager

Chassis

- 20-slot Multibus™ card cage
- 720 watt power supply

Options**Hardware:**

- Z clipping
- 2 or 4 MB CPU memory cards
- 4 bit-plane image memory cards
- Floating point accelerator
- 60" rack mounted chassis

Peripheral:

- Floppy disk drive
- Second 72 MB or 170 MB Winchester disk drive
- 60 MB ¼" cartridge tape drive
- ½" tape drive and controller
- Color printer and controller
- 11" x 11" digitizer tablet
- Dial and button box
- Programming Terminal
- 19" diagonal 30 Hz interlaced RGB monitor
- 15" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor

Software:

- FORTRAN and Pascal compilers --
- Terminal software
- EMACS text editor
- GKS library, level 2b

Communication:

- TCP/IP Ethernet software
- IBM link for 3278-9 emulation and file transfer

Physical and Environmental Specifications**Power Requirements:**

- AC voltage 93-132 or 186-264 VAC (factory set)
- AC frequency 47-63 Hz
- Chassis: 1250 VA, 1000 W, 3410 BTU/hr
- 19" monitor: 225 VA, 150 W, 512 BTU/hr

Size and Weight:

- 19" monitor: 18.5" H x 20" W x 21.5" D (51 x 48 x 54 cm), 84 lb. (38 Kg)
- Chassis: 29" H x 18" W x 27" D (74 x 46 x 69 cm), 190 lb. (86 Kg)

Environment:

- Operating: 50-86°F (10-30°C), 20-80% relative humidity, no condensation.
- Shipping/storage: 32-122°F (0-50°C), 10-90% relative humidity, no condensation.

Specifications are subject to change without notice.

UNIX is a trademark of AT&T

Ethernet is a trademark of Xerox.

Multibus is a registered trademark of Intel Corporation.

Silicon Graphics, IRIS, Geometry Pipeline, IRIS Window Manager, Geometry Engine, Geometry Accelerator, IRIS Graphics Library, and Geometry Partners are trademarks of Silicon Graphics, Inc.

IRF-3030-01 Printed in U.S.A. 2/86

3010 SPECIFICATIONS

System Specifications

Processors:

- 16 MHz MC68020 central processor
- 32-bit internal registers, 32-bit address space
- 10 125-nanosecond Geometry Engines
- 16-bit bit-slice frame buffer controller
- Independent microcoded display processor with 32 KB memory for fonts, textures, and cursors

CPU Memory:

- 2 MB dynamic RAM with parity error detection, expandable to 12 MB
- 256 KB EPROM for hardware initialization, self-configuration, and diagnostics, expandable to four 256/512 KB EPROMS

Image Memory:

- 8 1024 x 1024 bit-planes standard, expandable to 32 with 16-bit Z-buffer

Video Interface:

- RGB levels 0.7 V p-p into 75 ohms
- Separate composite 2 V p-p sync into 75 ohms
- 60 Hz non-interlaced 1024 x 768 resolution frame
- Other frame resolutions and rates available
 - 33 Hz interlaced 1024 x 768
 - 30 Hz interlaced 636 x 485
 - 25 Hz interlaced 768 x 575
- Genlock available with 485 and 575 visible line frames

Color Range:

- Color map mode (8- or 12-bit, single or double buffered)
- 4096 simultaneous colors, double buffered, displayable from palette of 16.7M
- RGB mode (24-bit), 16.7 million colors displayable

Standard Peripherals:

- 3½" Winchester disk, 22 MB unformatted, using an ST-506 interface
- 1 MB 5.25" floppy disk drive
- 83-key up-down encoded keyboard with user definable keys
- 19" diagonal non-interlaced RGB tilt and swivel monitor
- Optical mouse X-Y encoder with three buttons

Communications:

- Four RS-232C ports for keyboard and serial communication (up to 19.2K baud)

Standard Software:

- IRIS Graphics Library II
- UNIX kernel with demand paging and IRIS Terminal software
- IRIS Window Manager

Chassis:

- 20-slot Multibus™ card cage
- 720 watt power supply

Options

Hardware:

- Z-clipping
- 2 or 4 MB CPU memory cards
- 4 bit-plane image memory cards
- Floating point accelerator
- 60" rack mounted chassis

Peripherals:

- 11" x 11" digitizer tablet
- Dial and button box
- Color printer and controller
- 15" diagonal non-interlaced RGB tilt and swivel monitor
- 19" diagonal 30 Hz interlaced RGB monitor

Communication:

- Ethernet with XNS or TCP/IP software
- IBM Link for 3278-9 emulation and file transfer

Physical and Environmental Specifications

Power Requirements:

- AC voltage 93-132 or 186-264 VAC (factory set)
- AC frequency 47-63 Hz
- Chassis: 1250 VA, 1000 W, 3410 BTU/hr
- 19" monitor: 225 VA, 150 W, 512 BTU/hr

Size and Weight:

- 19" monitor: 18.5"H x 20"W x 21.5"D (51 x 48 x 54 cm), 84 lb. (38 Kg)
- Chassis: 29"H x 18"W x 27"D (74 x 46 x 69 cm) 190 lb. (86 Kg)

Environment:

- Operating: 50-86°F (10-30°C), 20-80% relative humidity, no condensation
- Shipping/storage: 32-122°F (0-50°C), 10-90% relative humidity, no condensation

Specifications are subject to change without notice.

UNIX is a trademark of AT&T. Ethernet is a trademark of Xerox. Multibus is a trademark of Intel Corporation. Silicon Graphics, IRIS, IRIS Window Manager, Geometry Pipeline, Geometry Engine, Geometry Accelerator, and IRIS Graphics Library are trademarks of Silicon Graphics, Inc.

LT-3010-01 Printed in U.S.A. 2/86

3020 SPECIFICATIONS**System Specifications****Processors:**

- 16 MHz MC68020 central processor 32-bit internal registers, 32-bit address space
- 10 125-nanosecond Geometry Engines
- 16-bit bit-slice frame buffer controller
- Independent microcoded display processor with 32 KB memory for fonts, textures, and cursors

CPU Memory:

- 2 MB dynamic RAM with parity error detection, expandable to 16 MB
- 256 KB EPROM for hardware initialization, self-configuration, and diagnostics, expandable to four 256/512 KB EPROMS

Image Memory:

- 8 1024 x 1024 bit-planes standard, expandable to 32 with 16-bit Z-buffer

Video Interface:

- RGB levels 0.7 V p-p into 75 ohms
- Separate composite 2 V p-p sync into 75 ohms
- 60 Hz non-interlaced 1024 x 768 resolution frame
- Other frame resolutions and rates available
 - 33 Hz interlaced 1024 x 768
 - 30 Hz interlaced 636 x 485
 - 25 Hz interlaced 768 x 575
- Genlock available with 485 and 575 visible line frames

Color Range:

- Color map mode (8- or 12-bit, single or double buffered)
- 4096 simultaneous colors, double buffered, displayable from palette of 16.7 million
- RGB mode (24-bit), 16.7 million colors displayable

Standard Peripherals:

- 72 MB unformatted 5.25" Winchester disk drive using an ST-506 interface
- 83-key up-down encoded keyboard with user definable keys
- 19" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor
- Optical mouse X-Y encoder with three buttons

Communications:

- Ethernet local area network with XNS software
- Four RS-232C ports for keyboard and serial communications (up to 19.2K baud)

Standard Software:

- UNIX System V operating system with Berkeley 4.2 and local enhancements
- C compiler and development environment
- IRIS Graphics Library II
- IRIS Window Manager

Chassis

- 20-slot Multibus™ card cage
- 720 watt power supply

Options**Hardware:**

- Z clipping
- 2 or 4 MB CPU memory cards
- 4 bit-plane image memory cards
- Floating point accelerator
- 60" rack mounted chassis

Peripheral:

- Floppy disk drive
- Second 72 MB or 170 MB Winchester disk drive
- 60 MB ¼" cartridge tape drive
- ½" tape drive and controller
- Color printer and controller
- 11" x 11" digitizer tablet
- Dial and button box
- Programming Terminal
- 19" diagonal 30 Hz interlaced RGB monitor
- 15" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor

Software:

- FORTRAN and Pascal compilers
- Terminal software
- EMACS text editor
- GKS library, level 2b

Communication:

- TCP/IP Ethernet software
- IBM link for 3278-9 emulation and file transfer

Physical and Environmental Specifications**Power Requirements:**

- AC voltage 93-132 or 186-264 VAC (factory set)
- AC frequency 47-63 Hz
- Chassis: 1250 VA, 1000 W, 3410 BTU/hr
- 19" monitor: 225 VA, 150 W, 512 BTU/hr

Size and Weight:

- 19" monitor: 18.5" H x 20" W x 21.5" D (51 x 48 x 54 cm), 84 lb. (38 Kg)
- Chassis: 29" H x 18" W x 27" D (74 x 46 x 69 cm), 190 lb. (86 Kg)

Environment:

- Operating: 50-86°F (10-30°C), 20-80% relative humidity, no condensation.
- Shipping/storage: 32-122°F (0-50°C), 10-90% relative humidity, no condensation.

Specifications are subject to change without notice.

UNIX is a trademark of AT&T

Ethernet is a trademark of Xerox.

Multibus is a registered trademark of Intel Corporation.

Silicon Graphics, IRIS, Geometry Pipeline, IRIS Window Manager, Geometry Engine, Geometry Accelerator, IRIS Graphics Library, and Geometry Partners are trademarks of Silicon Graphics, Inc.

US-3030-01 Printed in U.S.A. 2/86

3030 SPECIFICATIONS

System Specifications

Processors:

- 16 MHz MC68020 central processor, 32-bit internal registers, 32-bit address space
- 10 125-nanosecond Geometry Engines
- 16-bit bit-slice frame buffer controller
- Independent microcoded display processor with 32 KB memory for fonts, textures, and cursors

CPU Memory:

- 2 MB dynamic RAM with parity error detection, expandable to 16 MB
- 256 KB EPROM for hardware initialization, self-configuration, and diagnostics, expandable to four 256/512 KB EPROMS

Image Memory:

- 8 1024 x 1024 bit-planes standard, expandable to 32 with 16-bit Z-buffer

Video Interface:

- RGB levels 0.7 V p-p into 75 ohms
- Separate composite 2 V p-p sync into 75 ohms
- 60 Hz non-interlaced 1024 x 768 resolution frame
- Other frame resolutions and rates available
 - 33 Hz interlaced 1024 x 768
 - 30 Hz interlaced 636 x 485
 - 25 Hz interlaced 768 x 575
- Genlock available with 485 and 575 visible line frames

Color Range:

- Color map mode (8- or 12-bit, single or double buffered)
- 4096 simultaneous colors, double buffered, displayable from palette of 16.7 million
- RGB mode (24-bit), 16.7 million colors displayable

Standard Peripherals:

- 170 MB unformatted 5.25" Winchester disk drive, using an ESDI interface
- 83-key up-down encoded keyboard with user definable keys
- 19" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor
- Optical mouse X-Y encoder with three buttons

Communications:

- Ethernet local area network with XNS software
- Four RS-232C ports for keyboard and serial communications (up to 19.2K baud)

Standard Software:

- UNIX System V operating system with Berkeley 4.2 and local enhancements
- C compiler and development environment
- IRIS Graphics Library II
- IRIS Window Manager

Chassis

- 20-slot Multibus™ card cage
- 720 watt power supply

Options

Hardware:

- Z clipping
- 2 or 4 MB CPU memory cards
- 4 bit-plane image memory cards
- Floating point accelerator
- 60" rack mounted chassis

Peripheral:

- Floppy disk drive
- Second 170 MB Winchester disk drive
- 60 MB ¼" cartridge tape drive
- ½" tape drive and controller
- Color printer and controller
- 11" x 11" digitizer tablet
- Dial and button box
- Programming Terminal
- 19" diagonal 30 Hz interlaced RGB monitor
- 15" diagonal 60 Hz non-interlaced RGB tilt and swivel monitor

Software:

- FORTRAN and Pascal compilers
- Terminal software
- EMACS text editor
- GKS library, level 2b

Communication:

- TCP/IP Ethernet software
- IBM link for 3278-9 emulation and file transfer

Physical and Environmental Specifications

Power Requirements:

- AC voltage 93-132 or 186-264 VAC (factory set)
- AC frequency 47-63 Hz
- Chassis: 1250 VA, 1000 W, 3410 BTU/hr
- 19" monitor: 225 VA, 150 W, 512 BTU/hr

Size and Weight:

- 19" monitor: 18.5" H x 20" W x 21.5" D (51 x 48 x 54 cm), 84 lb. (38 Kg)
- Chassis: 29" H x 18" W x 27" D (74 x 46 x 69 cm), 190 lb. (86 Kg)

Environment:

- Operating: 50-86°F (10-30°C), 20-80% relative humidity, no condensation
- Shipping/storage: 32-122°F (0-50°C), 10-90% relative humidity, no condensation

Specifications are subject to change without notice.

UNIX is a trademark of AT&T.

Ethernet is a trademark of Xerox.

Multibus is a registered trademark of Intel Corporation.

Silicon Graphics, IRIS, Geometry Pipeline, IRIS Window Manager,

Geometry Engine, Geometry Accelerator, IRIS Graphics Library, and

Geometry Partners are trademarks of Silicon Graphics, Inc.

WF-3030-01 Printed in U.S.A. 2/86

WORKSTATION SOFTWARE

PROM MONITOR

- Hard Disk Interface
- Initial System Diagnostics
- Initial System Configuration
- Booting UNIX

AT&T

UNIX SYSTEM V

- Berkeley 4.2 Enhancements
- Local Enhancements

4.2 BSD

IRIS GRAPHICS LIBRARY II

- By SGI

C COMPILER

WINDOW MANAGER

- MEX

- Multiple-Exposure

NO!

FORTRAN and PASCAL COMPILERS (optional)

EMACS TEXT EDITOR

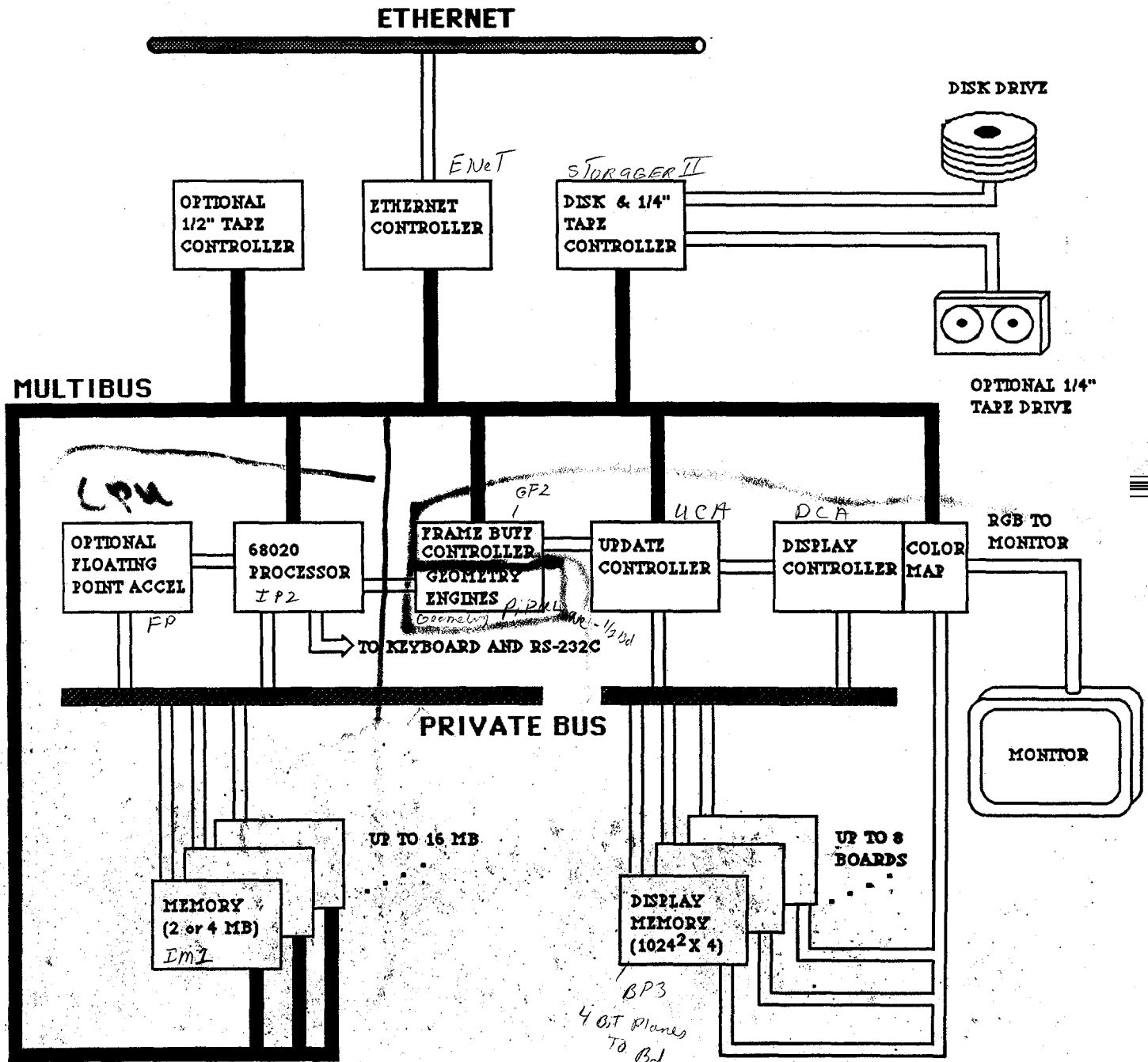
- of VI Editor

TCP/IP ETHERNET (optional)

/ XNS

IBM 327X EMULATION (optional)

BASIC SYSTEM BLOCK DIAGRAM



SUPPORT DOCUMENTATION - page 1

Type *MANn* *command - online help*

There are several manuals available that support the IRIS workstations. Following is a list of the manuals that a Field Engineer will most likely need.

Each system is shipped with these manuals plus some additional manuals not listed.

- **UNIX PROGRAMMER'S MANUAL**
VOL 1A: This manual contains **CHAPTER 1** of the eight chapter programmer's manual. It contains descriptions of all the basic shell commands.
- **UNIX PROGRAMMER'S MANUAL**
VOL 1B: This manual contains **CHAPTERS 2-8** of the eight chapter programmer's manual.
 - **Chapter 2:** Contains descriptions of the system calls. These are commands that cannot be executed individually, but used as code in a C-program.
 - **Chapter 3:** Contains descriptions of available subroutines. These cannot be executed individually, but used in high level compiled programs.
 - **Chapter 4:** Contains descriptions of the format and contents of various files used in the UNIX system.
 - **Chapter 5:** Contains descriptions of miscellaneous facilities such as macro packages, character set tables, etc.
 - **Chapter 6:** Contains descriptions of all UNIX games such as craps, hangman, etc.
 - **Chapter 7:** Contains descriptions of system files that refer to specific hardware peripherals and UNIX System device drivers.
 - **Chapter 8:** Contains some basic information concerning system administration.

SUPPORT DOCUMENTATION - page 2

- **UNIX PROGRAMMER'S MANUAL**
VOL 2A: This manual contains information and tutorials on various UNIX features such as ED, VI, Troff, etc.

- **UNIX PROGRAMMER'S MANUAL**
VOL 2B: This is the high level language reference manual. Contains information on various UNIX languages and programs such as FORTRAN, Fsock, Lint, etc.

- **IRIS OWNER'S GUIDE**
SERIES 3000: This manual contains information concerning Install, Test, System Administration, and workstation operation.

APPENDIX A
DAILY SCHEDULE
HARDWARE MAINTENANCE COURSE
IRIS 68020 BASED WORKSTATIONS

DAY	LESSON	LESSON TITLE	HOURS
1		Orientation	2.0
1	1	Course Introduction	1.0
1	2	Workstation Introduction	1.0
1	3	UNIX: Survival Skills & VI Editor	4.0
2	3	UNIX: Survival Skills & VI Editor	2.0
2		Lab Project 3.1: Basic UNIX Commands (pt.1)	2.0
2	3	UNIX: Survival Skills & VI Editor	2.0
2		Lab Project 3.1: Basic UNIX Commands (pt.2)	2.0
3		Quiz #1: Lessons 2 and 3	1.5
3	3	UNIX: Survival Skills & VI Editor	2.5
3		Lab Project 3.2: VI Editor	4.0
4		Test #1: Lesson 3	2.0
4	4	System Administration (Backups)	0.5
4		Lab Project 4.1: Making a Bootable Tape	2.0
4		Lab Project 4.2: Making Backup Tapes	2.0
4	4	System Administration (Restore)	0.5
4		Lab Project 4.3: Disk Drive Restoration	1.0
5		Lab Project 4.3: Disk Drive Restoration	1.0
5	4	System Administration (Configuration)	0.5
5		Lab Project 4.4: Creating New User Accounts	2.0
5		Lab Project 4.5: Adding ASCII Devices	2.0
5	5	Component Location	1.0
5		Lab Project 5.1: Component Location	1.5
6		Test #2: Lesson 4	2.0
6	6	Data Flow: Theory of Operation	6.0

APPENDIX A (continued)

DAY	LESSON	LESSON TITLE	HOURS
7		Quiz #2: Data Flow	2.0
7	7	Troubleshooting Lecture	2.0
7		Troubleshooting Lab	4.0
8		Troubleshooting Lab	8.0
9		Troubleshooting Lab	4.0
9		Final Exam and Performance Lab	4.0
10	8	Adjustments Lecture	1.0
10		Lab Project: Adjustments	7.0

Contents

LESSON 3: UNIX SURVIVAL SKILLS & VI EDITOR	1
WORKSTATION PROM MONITOR	2
BOOTING UNIX	3
IRIS CONFIGURATION SWITCHES	4
PROM MONITOR COMMANDS	5
PROM MONITOR CMDS - continued	6
PROM MONITOR COMMAND SYNTAX	7
PROM MONITOR HELP MENU	8
PROM MONITOR HELP MENU - continued	9
PROM MONITOR COMMAND EXAMPLES	10
THE BOOTING PROCESS: AUTOCONFIGURATION:	11
AUTOCONFIGURATION OUTPUT	12
SYSTEM DEVICE NAMES - page 1	13
SYSTEM DEVICE NAMES - page 2	14
SYSTEM DEVICE NAMES - page 3	15
USER OPERATION MODES	16
SYSTEM SHUTDOWN	17
BOOT EXERCISE - page 1	18
BOOT EXERCISE - page 2	19
BOOT EXERCISE - page 3	20
BOOT EXERCISE - page 4	21
BOOT EXERCISE - page 5	22
UNIX SYSTEM OVERVIEW	23
UNIX LAYERS	24
THE UNIX KERNEL	25
THE UNIX SHELL	26
UNIX UTILITIES	27
UNIX: FUNCTION & FACILITIES	28
UNIX DOCUMENTATION	29
UNIX IMPLEMENTATION	30
IRIS WORKSTATION DISKS	31
DISK STRUCTURE	32
DISK STRUCTURE DIAGRAM	33
UNIX FILE SYSTEM	34
FILE SYSTEM COMPONENTS	35
WHAT IS A STRUCTURED FILE SYSTEM	36
THE UNIX TREE: DIRECTORY LISTING - page 1	37

THE UNIX TREE: DIRECTORY LISTING - page 2	38
THE UNIX TREE: DIRECTORY LISTING - page 3	39
THE UNIX TREE: DIRECTORY LISTING - page 4	40
THE UNIX TREE: DIRECTORY LISTING - page 5	41
CURRENT and PARENT DIRECTORIES: File Hierarchy	42
PSEUDONYMS: Directory and file links	43
HOME DIRECTORY	44
THE C-SHELL (csh)	45
UNIX COMMAND SYNTAX	46
UNIX COMMAND SYNTAX - continued	47
UNIX BASIC SURVIVAL COMMANDS	48
SURVIVAL COMMANDS LISTING	49
UNIX ACCESS COMMANDS	50
UNIX FILE MANIPULATION COMMANDS - page 1	51
UNIX FILE MANIPULATION COMMANDS - page 2	52
UNIX FILE MANIPULATION COMMANDS - page 3	53
UNIX DIRECTORY MANIPULATION COMMANDS	54
UNIX GENERAL COMMANDS - page 1	55
UNIX GENERAL COMMANDS - page 2	56
UNIX PROGRAM CONTROL COMMANDS	57
UNIX COMMUNICATION COMMANDS	58
UNIX I/O COMMANDS	59
RUNNING PROGRAMS IN BACKGROUND	60
UNIX SYSTEM ADMINISTRATION COMMANDS - page 1	61
UNIX SYSTEM ADMINISTRATION COMMANDS - page 2	62
UNIX SYSTEM ADMINISTRATION COMMANDS - page 3	63
UNIX SYSTEM ADMINISTRATION COMMANDS - page 4	64
UNIX SYSTEM ADMINISTRATION COMMANDS - page 5	65
UNIX SYSTEM ADMINISTRATION COMMANDS - page 6	66
UNIX SYSTEM ADMINISTRATION COMMANDS - page 7	67
COMMAND SEARCH RULE	68
FILE AND DIRECTORY PROTECTION	69
FILE CREATION MODES	70
FILE CREATION MODES - continued	72
PROTECTION MODE EXAMPLES	73
FILE PROTECTION SUMMARY	74
STANDARD INPUT & OUTPUT	75
STANDARD INPUT & OUTPUT DIAGRAM	76
REDIRECTING I/O: Output to a file	77
REDIRECTING I/O: Append to a file	78
REDIRECTING I/O: Input from a file	79
THE PIPE OPERATION	80
VI: A TEXT EDITOR	81

LESSON 3: UNIX SURVIVAL SKILLS & VI EDITOR

Upon completion of this lesson, the student, using available documentation and an IRIS workstation, will be able to:

- Given a list of UNIX commands, correctly match the commands to explanations and use of each command, then demonstrate (on the workstation) that each command can be correctly executed.
- Demonstrate that specific files can be located and modified using the vi editor.
- Demonstrate that specific IRIS demonstration programs can be located and correctly run.
- Correctly draw the UNIX directory/sub-directory tree.

WORKSTATION PROM MONITOR

The basic level of software (in the IRIS workstation) is contained on the CPU board; held in proms and called the **PROM MONITOR**.

THE PROM MONITOR:

- Is a dumb terminal.
- Has a hard disk interface.
- Reads configuration switches.
- Will boot UNIX.
- Allows limited diagnostic functions.

*1P2 (7) +
On Back
Panel*

BOOTING UNIX

The IRIS boot environment is determined by the setting of configuration switches 1 through 4 on the cabinet back panel. Two basic boot options are available: **AUTOMATIC BOOT** and **PROM MONITOR BOOT**.

- **Automatic Boot**
At powerup, the IRIS tries to boot from a file called *defaultboot* on the device specified by the configuration switches (see table on next page).
- **PROM monitor boot**
At power up, the IRIS enters the PROM monitor and waits for further boot instructions.

Since the IRIS can be booted from different devices (*hard disks, tape drives, etc.*), the PROM monitor provides the *ls* command, a version of the UNIX *ls(1)* command, for displaying the names of files on the attached devices.

IRIS CONFIGURATION SWITCHES

As the table shows, switches 1 through 4 select the device from which the IRIS is to be booted. Switch 5 specifies whether the IRIS should perform an automatic boot or a PROM monitor boot. Switch 6 determines whether or not system information is displayed on the screen after the IRIS is reset. Switch 7 selects the display monitor type.

NOTE:

Manufacturing sets all the switches to the closed position. All classroom systems are configured as such.

Configuration Switches			
Switch	Switch Name	Position¹	Meaning
1 - 4	Boot environment	CCCC OCCC COCC OCCC OCOC	Hard disk boot Cartridge tape boot Floppy disk boot Network boot PROM monitor
5	Autoboot	C O	PROM Monitor boot Automatic boot
6	Quiet mode	C O	Display system information Don't display system information
7	Monitor select	C O	Display on primary monitor Display on secondary monitor
8-9	Reserved	C	

IRIS Configuration Switches

C means "Closed" and O means "Open" .

PROM MONITOR COMMANDS

The PROM monitor offers several commands: some can be used, others are for in-house personal and should not be used.

Following is a list of the commands you should be familiar with. Later in the course you will get to use the commands while performing a lab project.

COMMAND	DESCRIPTION
b	Load and begin execution of a boot file. The filename can be supplied as an argument with the boot command, or if no filename is given, then a file called <i>defaultboot</i> is searched for on the boot device. The boot device can be named as an argument or if no name is given, then the configuration switches will decide what device to boot from. <i>contains unix startup</i>
n <i>- network</i>	Same as b , but boot is from "XNS" or "TCP". <i>- prom or IP2</i>
ls	Used to list files on the selected device.
h	Print the help message.

PROM MONITOR CMDS - continued

COMMAND	DESCRIPTION
set	Print the current set values.
set debug 0/1	Set debug mode. This is a toggle switch, it allows you to inject <i>noise</i> into the boot process. Noise is a term used by the engineers and if an error is detected during the boot, then the process is halted and an error message is displayed.
set display	Used to enter the state of the configuration switches. The following steps must be followed when you change the switches: <ol style="list-style-type: none">1. reboot.2. Change switch settings.3. Execute set display.4. Depress reset.

PROM MONITOR COMMAND SYNTAX

When you are in the PROM monitor and request the help menu, the display that is printed is somewhat confusing. (The display is reproduced on the next two pages in your workbook)

The load type commands (**b**, **n**, **l**) and the list command (**ls**) may be executed with optional parameters supplied with the command. The optional parameters are: **MEDIA**, **DEVSPEC**, and **file**.

Type of Device

MEDIA: Media is a 2 or 3 character field that defines what type of device you will boot from or list files from. The available commands are: **hd**, **ct**, **fd**, **ip**, **sd**, **md**, **st**, **mt**, **sf**, **mf**, **xns**, and **rom**.

*hard disk
Tape & floppy*

Of these commands, you will only need to use: **hd**, **ct**, **fd**, or **xns**. This is because the PROM monitor will look to see the actual type of device attached and access that device using the correct device type.

i.e. For disk drives the following mnemonics are used: **ip**, **sd**, or **md**. Each type of available disk drive has a different mnemonic, but, the generic mnemonic **hd** can be used to load from any of the disk types. *This would not apply if the system had two disks with the same device number (say **si0** and **ip0**), then you would have to use the specific mnemonic to boot from the desired drive.*

This same principle applies to the tape (use **ct**) or floppy (use **fd**) devices.

Name or Unit # *UD, 0*

DEVSPEC: Devspec will be one of the following:

hostname	Name of the host, the MEDIA must be xns .
unit	The unit number of the device (0, 1, ...).
<unit><fs>	This is the unit number and filesystem (a-h). The MEDIA must be a disk drive.

file: This is the name of the boot file. If not given, the PROM monitor looks for the file *defaultboot*.

PROM MONITOR HELP MENU

iris> h

General Monitor Commands (All numeric values in hex):

MEDIA is one of the following:

- hd - hard disk.(look for ip, sd, then md)
- ct - cartridge tape. (look for st, then mt)
- fd - floppy disk. (look for sf, then mf)
- ip - interphase disk. (474 MB drive) *- SMD Disk 2190*
- sd(or si) - storager disk. (170 MB drive) *Storage II - interphase 3030 / 1/4 or 1/2 Tape*
- md - midas disk. (72 MB drive) *DSII 722 mb - 1/4 Tape*
- st(or sq) - storager cartidge tape. (1/2" tape) *ESDI or ST506*
- mt(or mq) - midas cartidge tape. (1/4" tape) *- ST506*
- sf - storager floppy disk.
- mf - midas floppy disk.
- xns - network. (ethernet)
- rom - EPROM board.

DEVSPEC is one of the following:

- host name - Name of the host. (MEDIA must be xns)
- unit - unit number of device (0, 1, ...).
(MEDIA must be a tape or disk device)
- <unit><fs> - unit number and filesystem (a-h)
(MEDIA must be a disk device.)
- address - multibus address.
(MEDIA must be a EPROM board.)

[MEDIA.DEVSPEC:][file] load and begin execution of the named file
file defaults to defaultboot
SPECS are from switch settings

- B, HDO? Default boot*
b [MEDIA.DEVSPEC:][file] same as above
- n [DEVICE:][file] same as b with MEDIA = xns
- ls [MEDIA.DEVSPEC:][file] list the files on the device
- l [MEDIA.DEVSPEC:][file] load but don't begin execution of the file
- g address [stack] start executing at specified address.
the stack address is a option.

Continue (y or n)?: y

PROM MONITOR HELP MENU - continued

hl?	print this help message.
set	print the current set values.
set media MEDIA	set the default boot media.
set devspec DEVSPEC	set the default boot device spec.
set debug 0/1	set the debug mode.
set display	set display options from switch settings.
set dcr BITS OPTION	set DC4 bits and option.
exit	reset the PROMS.
fm{blwll} ADDR VALUE [INCR] [CNT]	fill memory as byte, word or long starting at ADDR, with initial VALUE, incrementing VALUE by INCR for CNT times. (INCR defaults to 0; CNT defaults to 1)
dm{blwll} ADDR [CNT]	display memory as byte, word or long. (CNT defaults to 1)
em{blwll} ADDR	edit memory interactively as byte, word or long.
dpr	display processor registers.
epr REGISTER	edit the given processor register. (sr, vbr, cacr, caar, sfc, dfc).
iris>	

One the next page are some examples of PROM monitor commands with explanations.

PROM MONITOR COMMAND EXAMPLES

Following are six examples of PROM monitor commands.

1. b ct0:sifex

The above command will boot a file named *sifex* from tape drive 0.

b = monitor command
ct = MEDIA option
0 = DEVSPEC
sifex = file

2. b hd1:vmunix1

This command will boot a file named *vmunix1* from disk drive 1.

b =monitor command
hd =MEDIA option
1 =DEVSPEC
vmunix1 =file

3. n .elvin:/vmunix

This command will boot the IRIS over the Ethernet. The host system is called *elvin* and the boot file is *vmunix*.

n =monitor command
.elvin = DEVICE
/vmunix = Pathname to file *vmunix*.

4. ls ct0:

This command will list the file names of the first file on tape.

5. ls hd0a:

This command lists the contents of disk partition *a* of drive 0.

6. b

This command boots the IRIS using the defaults defined by the switches.

THE BOOTING PROCESS: AUTOCONFIGURATION:

When you enter the boot command from the PROM monitor the UNIX kernel is loaded into memory and UNIX then configures the system and displays a map for you. The map informs the user what equipment is or is not installed on the IRIS, how much memory is available, and what partition the UNIX ROOT is on.

Simple diagnostics are also run against the attached devices.

Following is some of the output displayed when the boot process begins. This display "blinks" on the screen, followed by the display shown on the next page.

```
iris> b
SGI Extent Filesystem
Loading: md:0:defaultboot
  Text: 038318 bytes
  Data: 0113d8 bytes
  Bss: 024d7c bytes
Jumping to loaded program ~ 20000400
```

After this message is displayed, the screen is cleared and the message shown on the next page is output sequentially as the configuration process proceeds.

AUTOCONFIGURATION OUTPUT

Following is the display that occurs at the IRIS console terminal as the configuration process proceeds. Some of the data changes from software release to release, but for the most part it remains the same.

On the next page descriptions of the equipment mnemonics, shown in this display, are given. (Remember, this output occurs after the output shown on the previous page is displayed and the screen is cleared)

SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]

(C) Copyright 1986 - Silicon Graphics Inc.

real = 4194304 *Bytes*

kmem = 561152 *Kernel mem.*

user = 3633152

bufs = 819200 (max=16k)

dspd0 not installed

qic0 not installed

sii0 at mbio 0x07200 ipl 5

si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0

si1 not installed

sf0 floppy (80/2/8) slave 2

siq0 at mbio 0x73fc ipl 5

sq0 (qic02 cartridge tape) slave 0

iph0 not installed

tmt0 not installed

ik0 not installed

nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2

fpa installed

lpen not installed

kernel debugger disabled.

root on si0a

swap on si0b. swplo=0 nswap=64000

INIT: SINGLE USER MODE

NOTE:

After the line "INIT: SINGLE USER MODE" UNIX will display the contents of file *motd*. This is the *message-of-the-day* file. The user can place any message he/she wants displayed every-time UNIX is booted. Once the message is displayed, the SINGLE USER prompt (#) is displayed.

#

Page 8

*Real - operating system
K + USR = Real*

*multi bus - etc. port
interrupt level*

numeric I.D. (address)

on disk

Network

Report on File System where FS mounted or not

SYSTEM DEVICE NAMES - page 1

The following text defines each mnemonic assigned to each component of an IRIS workstation. The autoconfiguration process will inform the user what boards and I/O devices are present; listing what multibus address has been assigned the board and the number of each device.

MNEMONIC	DESCRIPTION
dsd0	<p>This is the <i>Disk Controller</i> for the 72 MB hard disk, the floppy disk, and the 1/4" cartridge tape drive.</p> <p>If this board is installed, a message like the following is displayed next to the mnemonic: at mbio 0x7f00 ipl 1. mbio = Multibus. 0x7f00 = Address per board jumpers. ipl 1 = Interrupt priority as determined by position of board on the bus.</p>
qic0	<p>This is the <i>Quarter inch tape drive</i> that is attached to the dsd controller. Its assigned device address is 0. Only one tape drive can be attached to the dsd controller.</p>
sin	<p>This is the <i>72 MB disk drive</i>. Two of these drives can be attached to the dsd controller: md0 and md1. The example on the previous page informs you that a PRIAM drive is installed.</p>
mf0	<p>This mnemonic indicates the <i>Floppy Disk Drive</i> that could be attached to the dsd controller.</p>
iph0	<p>This is the <i>SMD Disk Controller</i>. If installed, the mbio address and interrupt level would be displayed as it was for the dsd board. This controller can have two 474 MB disk drives attached to it.</p>

SYSTEM DEVICE NAMES - page 2

MNEMONIC	DESCRIPTION
ipn	If the SMD controller was installed, this would indicate the presence of the <i>474 MB</i> disk drives: ip0 and/or ip1 . This mnemonic is not displayed if the controller is not installed.
sii0	This indicates that the <i>Storager II Disk/Tape Controller</i> is installed. This board can handle two <i>170 MB</i> disk drives and several optional <i>1/2"</i> and <i>1/4"</i> tape drives. If installed, the mbio address and interrupt level information would be present.
siqn	This would indicate the presence of the optional <i>1/4"</i> cartridge tape drives that could be attached to the <i>Storager II</i> board. Where <i>n</i> = a number.
tmtn	This is the mnemonic for the optional <i>1/2"</i> tape drives that could be attached to the <i>Ciprico Tapemaster</i> board. Where <i>n</i> = a number.
ik0	This indicates the <i>Color Printer</i> . An <i>IKON</i> board would be installed.
nx0	This mnemonic indicates the presence of the <i>Excelan Ethernet Controller Board</i> . Like any board installed, the mbio address is given and the interrupt level is indicated if installed, but some additional information is displayed that a customer may require, that is, the <i>Ethernet Address</i> . This address will look like the following: (0800.1400.3948) . (used in <i>/etc/host</i> .)

SYSTEM DEVICE NAMES - page 3

MNEMONIC	DESCRIPTION
fpa	This indicates that the IRIS has the <i>Floating Point board</i> installed.
lpen	This is the <i>Light pen</i> option.

USER OPERATION MODES

Three modes of operation are available to the UNIX user: **SINGLE-USER**, **MULTI-USER**, and **SUPER-USER**.

● SINGLE-USER

This mode can only support one user. It is usually used by the system administrator for file system repair and maintenance and other functions where one person requires exclusive use of the computer.

- The system boots into single-user.
- The single-user has full access privileges of all files.
- Only one file system is typically available and that is *ROOT*. Other file systems could be mounted if access is required to them.

● MULTI-USER

This mode supports several users simultaneously.

- Users login using assigned account names. New accounts are created by the system administrator. The system is shipped with six accounts: *root*, *rootcsh*, *rootsh*, *guest*, *demos*, and, *mexdemos*.
- User has full access privileges to files within that account.
- All file systems are generally available; you can read, write, and execute yours, usually read the rest of the system.

● SUPER-USER

Super-user is entered from multi-user using the command *SU*. On most systems entering super-user will be password protected because the super-user has the same file access privileges as the single-user.

- Workstation is *re-booted* from super-user using the command *reboot*.
- You can return to multi-user by using the command *exit*.

SYSTEM SHUTDOWN

Always shut the system down using the command **reboot**.

This "syncs" the disk drives; paging out all "dirty" pages to the disk, updates open files, and gets the heads off of the disk. This insures the integrity of the **super-block**.

- You must be the super-user or single-user to execute **reboot**.
- You should notify other users first using the **wall** command.
- Reboot will kill all running processes.
- Workstation control returns to the PROM monitor.

you are
super user or single
user

Write all!

***** WARNING *****

NEVER shut your system down using the RESET switch. If you do, the file system on disk will probably be corrupted (trashed). Reset should only be used as a last resort when UNIX is hung, and then, you should wait for the syncing of the drives by the CRON DAEMON. Remember, the drives are synced every 10 minutes by this process.

BOOT EXERCISE - page 1

This is a simple exercise that will introduce you to the subjects just discussed by this lesson: Displaying the PROM monitor help menu; booting UNIX using the defaultboot file; entering multi-user mode from single-user mode (where you entered when the boot process is complete); entering super-user mode from multi-user; and rebooting the system.

NOTE:

Two students should pair together to perform the exercise unless there are enough workstations for each student.

Enter only the commands that you are instructed to enter. You will get *free-time* on the workstation later in the course.

1. Check the *configuration switches* on the back panel and verify that all the switches are closed. This will cause a boot from the hard disk when you enter the PROM monitor boot command.
2. Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985

Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)

Configuration Switch: 0x0100

Multibus Window (2mb) at Megabytes 0 and 1.

Multibus accessible memory (1mb) begins

at Physical memory page 300,

at Virtual address 2000000.

iris>

3. Enter **h** at the **iris>** prompt.
This will cause the PROM monitor *help menu* to be displayed. (see previous page entitled PROM MONITOR HELP MENU.)
4. Enter **y** at the **Continue (y or n)?:** prompt.
This will cause the remainder of the help menu to be displayed. (see previous page entitled PROM MONITOR HELP MENU - continued.)

BOOT EXERCISE - page 2

5. Enter **b** at the **iris>** prompt.

This will cause the UNIX kernel to be loaded into memory. The file *defaultboot* will be searched for on the hard disk and it will boot the kernel. (remember, the configuration switches are selecting the hard disk as the boot device)

The data, shown on the page entitled THE BOOTING PROCESS: AUTOCONFIGURATION, will flash on your screen. This is followed by the configuration data shown on the page entitled AUTOCONFIGURATION OUTPUT.

When the prompt **#** appears, you will be in *single-user mode*.

6. Enter **ls** at the **#** prompt.

This will display the names of all the files and directories that make up the *ROOT (/)* directory. (more about this later)

7. Enter **multi** at the **#** prompt.

The following (minus the inserted instructional text) appears on your screen.

#

INIT: New run level: 2

Is the date [day month date time time-zone year] correct? (y or n)

This is asking you if the date is correct. If it is, enter **y**. If the date is incorrect, enter **n**. You would then be prompted to enter the date as the following example shows: *062312308700*. The last four digits (8700) are optional and are not required, but, it is a good idea to enter at least the year (87).

If you had to correct the date, you will again be asked if it is correct.

Do you want to check filesystem consistency? (y or n)

This is asking you if the program **fsck** should be run to check the filesystem consistency (integrity). Enter **y** to the prompt.

BOOT EXERCISE - page 3

We will talk about this check later in the course. If your filesystem is consistent, then the following message is output:

Checking file systems for consistency:

/dev/si0a

File system: root Volume: SGI

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
nnn files nnnnn K used nnnn K free

/dev/rsi0f

File system: usr Volume: SGI

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
nnn files nnnnn K used nnnn K free

Mounting: /usr

Preserved editor files

Cleared /tmp

Resetting locks and logs

Hostname: JULIE

Daemons:

update

cron

xnsd

lpd

lpsched

Daemons started

JULIE login:

BOOT EXERCISE - page 4**NOTE:**

If your `fsck` output does not look like the output on the previous page, then stop this project and call your instructor.

After the check is complete, UNIX informs you that several *daemons* were started. These are background programs that are always running. It also informed you that the `/usr` filesystem was mounted and that various "house-cleaning" tasks were performed.

8. Enter `student` at the **JULIE login:** prompt.
You are logging in to an account called *student*.

9. Enter `whoami` to the **JULIE 1 >** prompt.
The account name (*student*) will echo back to you.

10. Enter `su` to the **JULIE 2 >** prompt.
You are going to *super-user* mode. Notice that UNIX keeps track of the number of commands that you have executed by changing the number portion of the **JULIE** prompt.

The *super-user* account is password protected.

11. Enter the password `password`.

Notice that a new prompt (**JULIE 1 #**) is displayed. By using different prompts, UNIX lets you know what account you are in

12. Enter `whoami` to the **JULIE 1 #** prompt.
You see that the account name is no longer *student*, but, *root*.

BOOT EXERCISE - page 5

13. Enter **reboot** to the **JULIE 2 #** prompt.

You are going to *sync* the disks and re-enter the PROM monitor.

The following message blinks on the screen, the screen is cleared and the PROM monitor message is again displayed.

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985
```

```
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)
```

```
Configuration Switch: 0x0100
```

```
  Multibus Window (2mb) at Megabytes 0 and 1.
```

```
  Multibus accessible memory (1mb) begins
```

```
    at Physical memory page 300,
```

```
    at Virtual address 2000000.
```

```
iris>
```

14. End of exercise. Please power off your workstation using the front panel switch.

UNIX SYSTEM OVERVIEW

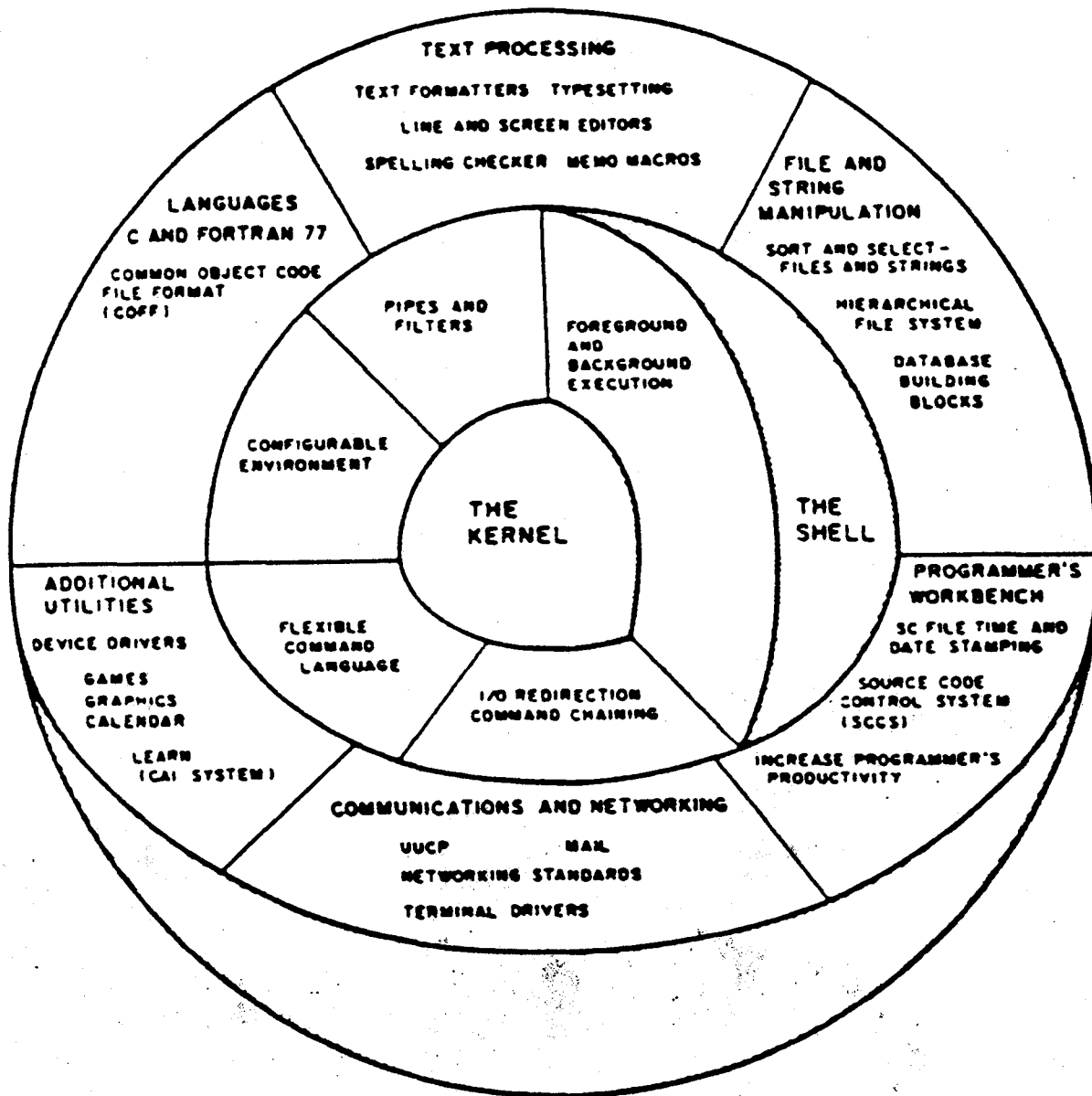
- AT&T UNIX*
- SGI workstations use System V with 4.2 enhancements.

 - Two command interpreters are offered:
 - C-Shell (csh): The Berkeley 4.2 DSB
 - Shell (sh): Bourne/At&T System V
Bourne shell

 - UNIX is a "layered" system:
 - The kernel is at the center
 - The shell is middle layer
 - The utilities make up outer layer

UNIX LAYERS

The following diagram shows the three layers of the UNIX System. The next three pages in the workbook give brief descriptions of each layer. More detailed information will follow as the course progresses.



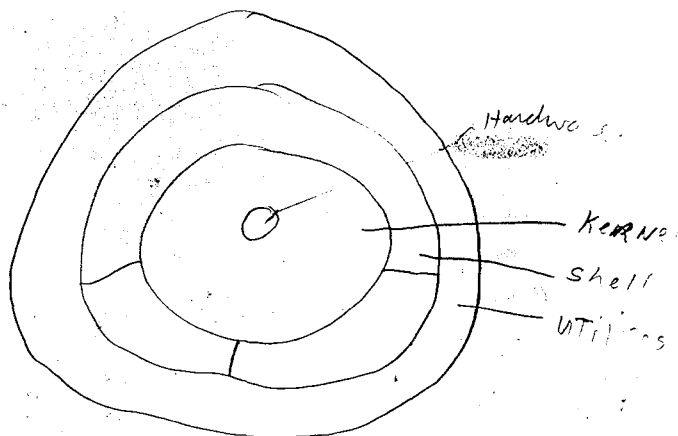
THE UNIX KERNEL

UNIX is a "layered" system with the *KERNEL* at its center.

Certain operating system functions are required many times each second. For example, the part of the UNIX System that is involved in switching from one program to another (time-sharing) is needed many times each second. In the UNIX System all of the functions that are needed immediately are constantly kept in memory. *The memory resident part of the operating system is called the KERNEL.*

- BASIC KERNEL FUNCTIONS:

- Supervise the I/O transactions.
- Create, manage, and control termination of "processes".
- Manage and control the hardware (housekeeping).
- Manage Main Memory.
- Manage file system organization and access.



THE UNIX SHELL

On many systems the command interpreter is a part of the internal structure of the operating system. In the UNIX System, however, the *shell* is just an ordinary program, similar to any other program that runs in the UNIX System.

Two shells are offered, but, only one will be running at a time. The user actually selects the shell he/she wants to use by placing a statement in the *password* file. When the user logs in, the correct shell will be started by the kernel for that user.

The shell is a command programming language that provides an interface to the UNIX Operating System. As a command language it interactively accepts commands from users and arranges for the requested actions to occur. As a programming language it contains control flow and string valued variables; this allows the user to create his/her own special commands, called *shellscripts* or *executable files*.

The program which implements the shell is called `/bin/csh` for the Berkeley version and `/bin/sh` for the Bourne version. It is one of these statements that is placed into the password (`/etc/passwd`) file.

Shell - interpret your commands

commands

who

ls

vi

UNIX UTILITIES*- Applications (Programs)*

Many operating system functions are needed only occasionally, such as the capability to move information from one mass storage device to another. These types of functions are provided by utilities (commands by any other name), standard programs which are invoked upon demand by the user through the command interpreter (the shell).

Like the shell, the utilities are programs and run as processes, managed by the shell.

Some utilities are:

- The editors.
- Language programs (C and FORTRAN).
- Text processing programs (xroff and spell)
- Communicatons and networking programs.
- Mail handler.
- All the basic commands (Vol. IA).
- All the user created shellscripts, etc...

UNIX: FUNCTION & FACILITIES

- **GENERAL OPERATING SYSTEM FUNCTIONS:**
 - The task is to orchestrate (manage) the use of system resources.
 - Provide user access to system facilities.

- **UNIX FUNCTIONS:**
 - Handle devices: Disk(s), tape(s), ethernet, graphics, terminals, etc.
 - Maintain the file system (on the disks).
 - Provide a standard mechanism for accessing all UNIX files and devices.
 - Provide process management: Process creation and process scheduling.
 - Provide virtual memory. (Appearance that there is more memory than there is physical memory. The SWAP area of the system disk is used for virtual addressing.)
 - Provide a user interface - via Command Interpreter: CSH or SH.
 - UNIX tools: An assortment of simple, useful UNIX programs - i.e. ls, more, rm, cp, mv, etc...

UNIX DOCUMENTATION

- UNIX PROGRAMMERS MANUAL VOL 1A & 1B.
 - Table of contents - An alphabetical listing of utilities (commands), pages i-ix.
 - Permuted index - This is a *keyword(s)* form of indexing utilities in the UNIX manuals, pages 1-65.

- THE USER LOCATES INFORMATION PER THE CHAPTER IDENTIFIER (n):
 - i.e.
 - passwd(1)
Locate information in chapter 1 concerning use of the password utility (command).
 - passwd(4)
Locate information in chapter 4 concerning the format of line entries in the password file.
 - cp(1)
Locate information in chapter 1 concerning the use of the *copy* utility (command).
 - sii(7)
Locate information in chapter 7 concerning the *special device file* that serves as the interface for the Interphase Storager 2 control board.

UNIX IMPLEMENTATION

- UNIX is comprised of thousands of files that are arranged in a *tree-like* structure.
 - The files are grouped together in several directories; grouped files usually have some similarity of function or use.
 - The top directory is called **ROOT** directory and has a designation of **(/)**.
- The UNIX kernel is booted from the system disk and lives in main memory.
- As additional UNIX files and/or utilities are required, they are paged into memory from the disk.
- Storage capacity of disks:
 - 3010 has **20 MB** hard disk and **Internal floppy**.
 - 2400 Turbo has **72 MB** or optional **144 MB** hard disks.
 - 3020 has the same disk configuration as the 2400T.
 - 3030 has **170 MB** or optional **474 MB** hard disks.

IRIS WORKSTATION DISKS

- Disk drive space can be divided into partitions:
 - 8 partitions are possible, labeled:
si0a-h for drive 1.
si1a-h for drive 2.
 - 3 partitions are standard for drive 1:
si0a - holds *root* filesystem.
si0b - holds *swap area* used for paging.
si0f - holds *usr* filesystem.
 - 1 partition is standard for drive 2.
si1g - holds user files.
- Sector size = 512 bytes.

- The *Extent File System* (efs) uses variable length blocks.

- Two methods are used to access the drives:
 - **Block mode:** Data is buffered as it comes off the disk.
 - **Raw mode:** Data is passed directly to the user.

DISK STRUCTURE

Each partition on the disk has the following structure: (See diagram on next page)

- **Block 0 - The Boot Block:**

- This block is generally used to boot UNIX systems.
- Block 0 is **NOT USED** on IRIS systems.

- **Block 1 - The Super Block:**

This block contains information about the filesystem:

- Size of the filesystem.
- The filesystem name.
- The filesystem Volume name.
- The filesystem Free Block List.
- The time of last backup.
- The Free i-node list.
- The total number of i-nodes.
- The time of last update.

sync
updates
super block

- **Blocks 2 through n - The i-nodes Block:**

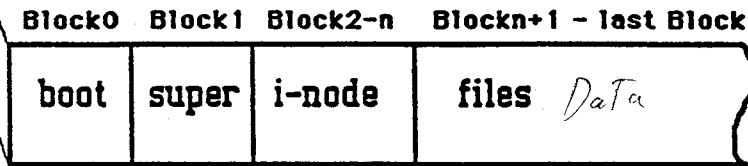
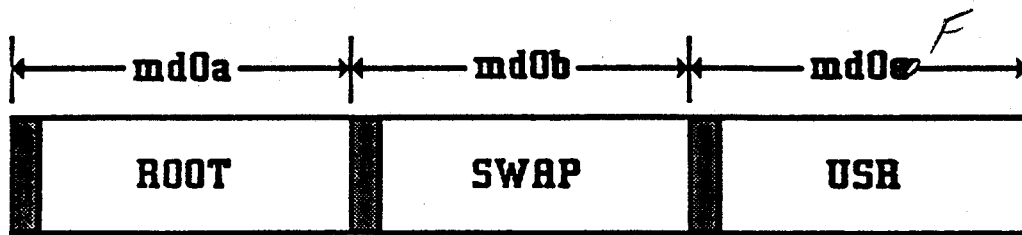
- Each i-node is a description of a *directory* or a *file*.
- The i-node contains access rights and number of links for the file or the directory.
- The i-node contains the block numbers (disk map pointer) to the physical location of the file or directory on the disk.
- There is a *one-to-one* relation of i-nodes to the total number of files and directories.

- **Blocks n+1 through last block - The Data Blocks:**

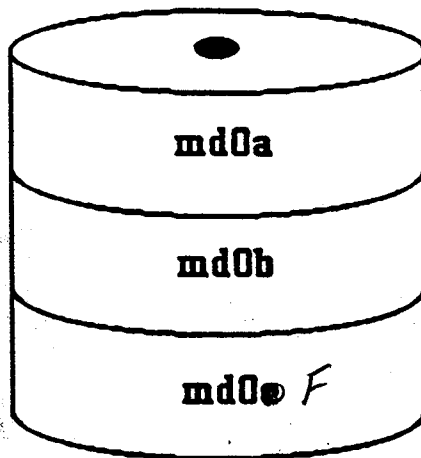
This is the actual user data blocks and directories.

DISK STRUCTURE DIAGRAM

Each filesystem has the same disk structure.



1 Block = 512 Bytes = 1 sector on (3000 series only)



Partition

UNIX FILE SYSTEM

The UNIX operating system is a structured system that has all its files grouped by the related functions they perform or the type of data that the files contain.

The operating systems files are structured into compartments called *directories*. Each directory holds related files. The directories are arranged in a *tree-like* structure with a top directory called **ROOT**. Actually, the structure will resemble the form of an inverted tree, with the **ROOT** directory at the top. (more about directories later)

WHAT KIND OF DATA DOES A FILE HOLD?

- **Program files contain:**

Execute {
- Managers.
- Editors.
- Compilers.

- **Command files (utilities):**

Execute {
- Shellscripts.
- Compiled programs.

- **Helpfiles (man pages).** *Documentation*

- **Message files:**

- Prompts. *TEST*
- Error.

- **I/O device files (provide interface for individual devices).**

- **Configuration files.**

- **User data files.**

FILE SYSTEM COMPONENTS

- **DIRECTORY:**

A compartment or space (disk or memory) that holds related files and/or other directories.

- **FILE:**

A file is a data record:

i.e.

- Source code.

- Shellsript.

- A data table.

- **PATHNAME:**

This is a pointer to a desired directory and/or file within the tree structure. It defines the position of the file or directory within the tree.

It is the job of the kernel to determine if the pathname is valid and if access to the object directory or file will be allowed.

- **FILE PROTECTION CODE:**

Each file and directory has a protection code that defines the following parameters:

- If the file or directory is read only.

- If the file or directory can be modified.

- If the file is executable.

Read
Write
execute

WHAT IS A STRUCTURED FILE SYSTEM

This drawing attempts to show that the way a book (*lets call it the root directory*) is divided into sections (*lets call these sub-directories*), with the sections divided into chapters (*we will call these files*), is very similar to that of the UNIX operating system structure.

I. LIFE

A. Vertebrates

1. Mammals
 - a. Great Apes
 - b. Homo Sapiens
2. Birds
 - a. Winged
 - b. Flightless
3. Fish
 - a. Fresh Water
 - b. Salt Water

B. Insects

1. Spiders
 - a. Web Spinning
 - b. Ground living
2. Ants
 - a. Red Ants
 - b. Black ants

ROOT DIRECTORY (Life)

Sub-directory of Life

Sub-directory of Vertebrates

File in mammals

File in mammals

Sub-directory of Vertebrates

File in Birds

File in Birds

Sub-directory of Vertebrates

File in fish

File in fish

Sub-directory of life

Sub-directory of Insects

File in Spiders

File in Spiders

Sub-directory of Insects

File in ants

File in ants

PATHNAME:

This is a pointer to a desired directory and/or file. It describes its position within the tree. We want the chapter (file) containing information concerning *Winged Birds*.

i.e.

I.A.2.a = This is pointer into the book.

`/life/vertebrates/birds/winged` = This is pathname for the system. The *slash (/)* is used to separate the directories and file.

REMEMBER:

- Directories contain *files* and/or other *directories*.
- Files contain *data*. (ascii or binary)
- Pathname is a *pointer* to a specific file and/or directory.

THE UNIX TREE: DIRECTORY LISTING - page 1

The **ROOT** file system and all its sub-directories are shown below. Sub-directories are *indented* from their parent directory. (more on parent directories later)

This listing only shows the pathnames for *directories*. The file names were omitted to save space. The **USER** directory (designation **/usr**) only has a few accounts other than the accounts supplied with the basic system. An actual customer listing would be much larger than this listing because there would be many more user accounts in the directory **/usr**.

The **ROOT** file system is contained on disk partition **si0a**. Its pathname designation is (**/**).

```
/
/Versions
/bin
/d
/dev
  /dev/EXOS
/etc
  /etc/tabset
/kernels
/lib
/lost+found
/stand
/tmp
```

The **/usr** file system is a sub-directory of **ROOT**, but it lives on disk partition **si0f**.

```
/usr
  /usr/adm
  /usr/bin
  /usr/diag
  /usr/dict
  /usr/games
    /usr/games/lib
      /usr/games/lib/quiz
```

THE UNIX TREE: DIRECTORY LISTING - page 2

```
/usr/include
  /usr/include/g1
  /usr/include/g12
  /usr/include/gpib
  /usr/include/ipII
  /usr/include/machine
  /usr/include/multibus
  /usr/include/pmII
  /usr/include/sys
  /usr/include/xns
/usr/lib
  /usr/lib/acct
  /usr/lib/emacs
    /usr/lib/emacs/bin
    /usr/lib/emacs/databases
    /usr/lib/emacs/doc
    /usr/lib/emacs/maclib
  /usr/lib/font
  /usr/lib/g12
  /usr/lib/help
  /usr/lib/lex
  /usr/lib/lint
  /usr/lib/macros
  /usr/lib/me
  /usr/lib/refer
  /usr/lib/sa
  /usr/lib/spell
  /usr/lib/term
  /usr/lib/tmac
  /usr/lib/uucp
/usr/local
  /usr/local/bin
  /usr/local/etc
  /usr/local/include
  /usr/local/lib
/usr/lost+found
/usr/mail
```


THE UNIX TREE: DIRECTORY LISTING - page 3

```
/usr/man
  /usr/man/a_man
    /usr/man/a_man/cat1
    /usr/man/a_man/cat7
    /usr/man/a_man/cat8
    /usr/man/a_man/man1
    /usr/man/a_man/man7
    /usr/man/a_man/man8
  /usr/man/u_man
    /usr/man/u_man/cat1
    /usr/man/u_man/cat2
    /usr/man/u_man/cat3
    /usr/man/u_man/cat4
    /usr/man/u_man/cat5
    /usr/man/u_man/cat6
    /usr/man/u_man/man1
    /usr/man/u_man/man2
    /usr/man/u_man/man3
    /usr/man/u_man/man4
    /usr/man/u_man/man5
    /usr/man/u_man/man6
/usr/news
```

The user accounts under `/usr/people` are highlighted for easy identification.

```
/usr/people
  /usr/people/demos
    /usr/people/demos/robotlib
  /usr/people/gifts
    /usr/people/gifts/arch
    /usr/people/gifts/archmex
    /usr/people/gifts/dbx_tutorial
      /usr/people/gifts/dbx_tutorial/C
        /usr/people/gifts/dbx_tutorial/C/src
      /usr/people/gifts/dbx_tutorial/adv_fortran
        /usr/people/gifts/dbx_tutorial/adv_fortran/src
      /usr/people/gifts/dbx_tutorial/fortran
        /usr/people/gifts/dbx_tutorial/fortran/src
```

THE UNIX TREE: DIRECTORY LISTING - page 4

```
/usr/people/gifts/examples
  /usr/people/gifts/examples/Fortran
  /usr/people/gifts/examples/Pascal
  /usr/people/gifts/examples/misc
  /usr/people/gifts/examples/tcp
  /usr/people/gifts/examples/xns
/usr/people/gifts/lib
/usr/people/gifts/mextools
  /usr/people/gifts/mextools/images
  /usr/people/gifts/mextools/imglib
  /usr/people/gifts/mextools/imgtools
  /usr/people/gifts/mextools/include
  /usr/people/gifts/mextools/mexrcs
  /usr/people/gifts/mextools/portlib
  /usr/people/gifts/mextools/tools
/usr/people/guest
/usr/people/mexdemos
  /usr/people/mexdemos/bin
  /usr/people/mexdemos/hemelib
  /usr/people/mexdemos/surflib
  /usr/people/mexdemos/zshadelib
/usr/people/rich
  /usr/people/rich/bin
  /usr/people/rich/course
    /usr/people/rich/course/2000
      /usr/people/rich/course/2000/ig
        /usr/people/rich/course/2000/ig/lab
        /usr/people/rich/course/2000/ig/test
      /usr/people/rich/course/2000/workbook
        /usr/people/rich/course/2000/workbook/appendix
        /usr/people/rich/course/2000/workbook/lab
    /usr/people/rich/course/3000
      /usr/people/rich/course/3000/ig
        /usr/people/rich/course/3000/ig/lab
        /usr/people/rich/course/3000/ig/test
      /usr/people/rich/course/3000/workbook
        /usr/people/rich/course/3000/workbook/appendix
```

THE UNIX TREE: DIRECTORY LISTING - page 5

```
    /usr/people/rich/course/3000/workbook/lab
  /usr/people/rich/course/clover
/usr/people/student
/usr/people/student1
/usr/preserve
/usr/pub
/usr/spool
  /usr/spool/at
  /usr/spool/colord
  /usr/spool/lp
    /usr/spool/lp/class
    /usr/spool/lp/interface
    /usr/spool/lp/member
    /usr/spool/lp/model
    /usr/spool/lp/request
  /usr/spool/lpd
  /usr/spool/uucp
  /usr/spool/uucppublic
    /usr/spool/uucppublic/receive
/usr/tmp
```

NOTE:

This listing was created using the `ls` utility (command). The actual command used looked like the following:

```
ls -R > temp
```

The `-R` argument informs the `ls` command to *Recursively list subdirectories encountered*. The output of the command (which would normally go to the screen, was *re-directed* into a file called *temp*. The file was then edited using the `vi` editor, and all the file names were removed.

Later in the course we will cover using commands and re-directing command output. When you get into your *free-time* using the workstation, you may want to try this command.

CURRENT and PARENT DIRECTORIES: File Hierarchy**● CURRENT DIRECTORY:**

This is the directory that you are currently in. All of the files are directly accessible to you by just using the *filename* with all your commands. The full *absolute* pathname is not needed when accessing files in this directory.

i.e.

Say you are in a directory called */usr/people/student* and you want to look at the contents of a file called *xyz* (that is in the directory *student*) using the *more* command. The command would look like the following:

```
more xyz
```

You are still in the directory */usr/people/student*, lets now assume you want to look at a file called *print* in the directory */usr/people/student1*. You must use the *absolute* pathname to access the file. It would look like the following:

```
more /usr/people/student1/print
```

● PARENT DIRECTORY:

This is the directory *immediately above* the current directory.

● EXAMPLES:

- If */usr/people/student* is the current directory,
- Then */usr/people* is the parent directory.

- The root directory has NO parent directory.

PSEUDONYMS: Directory and file links

- **PSEUDONYM (links):**
The **parent** and **current** directories are linked together with a fictitious named called a *pseudonym*. The pseudonyms for the current and parent directories are:
 - Dot (.) is the *current* directory.
 - Dot Dot (..) is the *parent* directory.

NOTE:

This linking of the *current* and *parent* is consistent throughout the tree structure. You will see the (.) and (..) filenames in each directory. The system will not let you remove the *link* files, for if it did, you could destroy the tree structure.

Files can also be linked together. We have already seen one such file, and that is *default-boot*. This is a pseudonym for the real file called *vmunix*. (more about this later)

UNIX COMMAND SYNTAX

When you want to find information concerning a specific command, use *Volume 1A or 1B* of the *UNIX Programmer's Manual*.

Each command is followed by a number or a number and a letter in *parenthesis*.

i.e. `cd(1)`, `sgilabel(1m)`, `xlogin(1c)`

The number tells you what chapter of the manual you can find information concerning the command, and the letter qualifies the command as to specific purpose:

- (1) Commands of *general utility*.
- (1C) Commands used for *communication* with other systems.
- (1D) *Demonstration* programs.
- (1G) Commands used primarily for *graphics & computer-aided design*.
- (1M) Commands used chiefly for *system maintenance & administration* purposes.
- (1W) *Color editing* commands and other tools for the window manager.

Unless otherwise noted, commands will accept options and other arguments according to the following syntax:

`name [option(s)] [command arg(s)]`

WHERE:

name The name of the command.

option There are two types of options:
1. *-noargletter(s)*.
2. *-argletter <> optarg*.
where <> represents white-space,
and - indicates a minus sign.

UNIX COMMAND SYNTAX - continued

noargletter A single letter representing an option without an argument.

i.e. `ls -a`

argletter A single letter representing an option requiring an argument.

optarg An argument (character string) satisfying the preceding *argletter*.

i.e. `sgilabel -n "GL2-W3.5" -s 13256`

cmdarg A pathname (or other command argument) not beginning with - or, by itself indicating the standard input.

i.e. `diff -e file1 file2`

NOTE:

All commands do not adhere to the aforementioned syntax.

UNIX ACCESS COMMANDS

These commands allow the user access to UNIX, moving from multi-user to super-user and back, and rebooting UNIX.

COMMAND	CHAPTER	DESCRIPTION
login	<i>cs(1)</i>	Log into UNIX and start a shell per <i>.login</i> & <i>.cshrc</i> files.

NOTE:

The lesson always assumes that C-shell (*cs*) is running. If you are using Shell (*sh*), then *.profile* is used instead of *.cshrc* (more on this later).

logout	<i>cs(1)</i>	Terminate your login shell.
--------	--------------	-----------------------------

exit	<i>cs(1)</i>	Exit a shell.
------	--------------	---------------

multi	<i>multi(1M)</i>	Enter <i>multi-user</i> mode from <i>single-user</i> mode.
-------	------------------	--

su	<i>su(1)</i>	Enter <i>super-user</i> mode from <i>multi-user</i> mode.
----	--------------	---

NOTE:

Use *exit* to return to *multi-user* mode.

reboot	<i>reboot(1M)</i>	This <i>syncs</i> the disk (pages out dirty files) and puts you into the PROM monitor.
--------	-------------------	--

NOTE:

You must be *super-user* to execute *reboot*.

UNIX FILE MANIPULATION COMMANDS - page 1

These commands allow the user to look at, copy, move, search, remove, compare, and create files.

COMMAND	CHAPTER	DESCRIPTION
more	<i>more(1)</i>	Read a file page-by-page. (The <i>space bar</i> pages the file and <i>ctrl-c</i> terminates the read).
cat	<i>cat(1)</i>	Read a file. This command does not allow paging; the complete file is scrolled past the screen. A more useful use is appending one file to another (<i>catenate</i>).
cp	<i>cp(1)</i>	Copy one file to another.
mv	<i>cp(1)</i>	Move a file or rename a file.
ln	<i>cp(1)</i>	Create a <i>pseudonym</i> (fictitious name) for a file or directory. This allows the user to maintain one copy of a file and have pseudonyms in other directories (this saves space) that are <i>linked</i> to the original file. Both names have equal weight; either name can be used when accessing the file.

i.e.

The file *defaultboot* is a pseudonym for *vmunix*, which is an executable file used to boot the UNIX root.

UNIX FILE MANIPULATION COMMANDS - page 2

COMMAND	CHAPTER	DESCRIPTION
ln - continued		Another place where links are important is in the <i>directory hierarchy</i> . The name (..) always references the parent directory. When a directory is created (using <i>mkdir</i>), the system <i>links</i> the name (..) to the parent directory and also <i>links</i> the name (.) to the current directory. The complete directory hierarchy is maintained with links between all the directories. You cannot use the "ln" command to change these links.
rm	<i>rm(1)</i>	Remove (delete) a file or contents of a directory.
diff	<i>diff(1)</i>	Compare two files. If the files are equal, the prompt is returned. If the files are unequal, each line that is different (for each file), the line is printed.
file	<i>file(1)</i>	Used to determine what kind of file the file is.
grep	<i>grep(1)</i>	Search a file for a specific pattern (character string) and print each line that contains the pattern.
sort	<i>sort(1)</i>	Arranges a file (line-by-line) into numeric and alphanumeric strings.

UNIX FILE MANIPULATION COMMANDS - page 3

COMMAND	CHAPTER	DESCRIPTION
sum	<i>sum(1)</i>	Print a <i>checksum</i> and block count of a file. This is a great tool to use when transmitting files. The check sum should be the same after a file is transmitted.
vi	<i>vi(1) or Vol IIA</i>	Full screen editor.

UNIX DIRECTORY MANIPULATION COMMANDS

These commands allow the user to enter a directory, print the pathname of the current directory, list the contents of a directory, remove or move a directory, and push or pop directory pathnames onto or off of a memory stack.

COMMAND	CHAPTER	DESCRIPTION
ls	<i>ls(1)</i>	List the contents of a directory.
cd	<i>cd(1)</i>	Change to another directory.
pwd	<i>pwd(1)</i>	Print the pathname of the current directory.
mkdir	<i>mkdir(1)</i>	Create a new directory. When you do this, the new directory is linked to the parent directory.
rmdir	<i>rm(1)</i>	Remove a directory name from the tree.
mvdir	<i>mvdir</i>	Move a directory.
pushd	<i>cs(1)</i>	Push the pathname of the current directory onto a stack and change to the specified directory. (like a <i>jump subroutine instruction</i>)
popd	<i>cs(1)</i>	Pop the top of the stack, return to the directory per the popped pathname. (like a <i>return subroutine instruction</i>)

UNIX GENERAL COMMANDS - page 1

These commands allow the user to perform various utilities that are useful, but, could not be placed into specific categories.

COMMAND	CHAPTER	DESCRIPTION
alias	<i>cs(1)</i>	Allows the user to rename, redefine, and rearrange commands in a way that suits their needs. i.e. Say you frequently use a command that contains many characters: options, arguments, or long pathnames. You could create an <i>alias</i> name, for the command, that was just a few characters in length. When you entered the alias name, the actual command desired is executed.
date	<i>date(1)</i>	Print the date.
echo	<i>echo(1)</i>	Echo a supplied message.
find	<i>find(1)</i>	Find a file. (this is a very useful command)
history	<i>cs(1)</i>	Display the last n commands executed, where n is defined at login-time by the <i>.cshrc</i> or <i>.login</i> file.
mail	<i>mail(1)</i>	Send or receive electronic mail.

UNIX GENERAL COMMANDS - page 2

COMMAND	CHAPTER	DESCRIPTION
man	<i>man(1)</i>	Display the manual page for the <i>named</i> command.
pr	<i>pr(1)</i>	Paginate, title, and format a file for printing.
spell	<i>spell(1)</i>	Check the spelling of words in a file.
who	<i>who(1)</i>	Print a list of all the users on the system and what port they are attached to.
whoami	<i>who(1)</i>	Print the name of the account that the user is logged into.
write	<i>write(1)</i>	Allows the user to talk to another terminal.
wall	<i>wall(1)</i>	Allows the user to talk to all users on the system (broadcast messages)
tty	<i>tty(1)</i>	Print the name of the port that a terminal is attached to.
wsiris	<i>wsiris(1C)</i>	Emulate an IRIS terminal on an IRIS workstation.

UNIX PROGRAM CONTROL COMMANDS

These commands allow the user to start a job (process), suspend a job, kill a job, and gather status information about processes already running.

COMMAND	CHAPTER	DESCRIPTION
at	<i>at(1)</i>	Submit a job to be run at a later time (batch).
nice	<i>nice(1)</i>	Run a job at a reduced priority.
kill	<i>kill(1)</i>	Terminate a job.
sleep	<i>sleep(1)</i>	Suspend a job for a specified amount of time or execute a job every so often.
ps	<i>ps(1)</i>	Print the status information for jobs.

NOTE:

Remember, commands are nothing more than programs (jobs or processes). They are called utilities, but run as jobs or programs.

UNIX COMMUNICATION COMMANDS

These commands allow the user to communicate with another workstation, terminal, or computer system, using the XNS protocol. (XNS is standard on the IRIS)

COMMAND	CHAPTER	DESCRIPTION
xcp	<i>xcp(1C)</i>	Remote copy. Copy files between machines.
xlogin	<i>xlogin(1C)</i>	Allows the user to connect his/her terminal or workstation to a remote host system.
xx	<i>xx(1)</i>	Allows the user to enter commands from the workstation or terminal that will be executed on another machine on the network.

If you are using the optional *TCP/IP Protocol*, equivalent commands as described above exist.

- **rsh** same as *xx*.
- **rlogin** same as *xlogin*.
- **rcp** same as *xcp*.

UNIX I/O COMMANDS

These commands allow the user to read or write files to tape or disk, and print files on a printer.

COMMAND	CHAPTER	DESCRIPTION
cpio	<i>cpio(1)</i>	Read or write files to tape or disk.
tar	<i>tar(1)</i>	Read or write files to tape.
lpr	<i>lpr(1)</i>	Spool files to the print queue for printing.
lp	<i>lp(1)</i>	Same as lpr , but, has a cancel option for removing requests from the queue.
lpq		Print the printer queue.
mt	<i>mt(1)</i>	This command allows the user to rewind, forward space, backspace, erase, write end-of-file markers, etc., on mag tape.

RUNNING PROGRAMS IN BACKGROUND

If you want to run a program in *background mode*, then follow the command with the **&** character.

i.e. `printit file1 &`

The above command (`printit`) will run detached from the keyboard. The user could enter other commands at the keyboard. If the program takes a long time to run and its not running in *background mode*, then the keyboard will be *locked-out* (dead) until the program completes.

The above mentioned parameter applies to any program you run.

UNIX SYSTEM ADMINISTRATION COMMANDS - page 1

These commands allow the user to perform various system maintenance or system administration tasks. These tasks include: backing up UNIX to tape; restoring UNIX; rebuilding corrupted disk drives; adding new user accounts; adding additional or optional hardware.

COMMAND	CHAPTER	DESCRIPTION
chgrp	<i>chown(1)</i>	Used to change the group affiliation of a file.
chmod	<i>chmod(1)</i>	Used to change the access modes of a file or directory (read, write, or executable).
chown	<i>chown(1)</i>	Used to change the ownership of a file.
cu	<i>cu(1)</i>	Used to call another UNIX System when a serial transmission of files is going to occur.
devport	<i>devport(1)</i>	Used to assign IRIS serial ports (1-3) to an external graphics device like the dial and button box or digitizing tablet.
df	<i>df(1)</i>	Reports the status of all attached filesystems on a disk drive. Status includes the size of the filesystem in <i>Kbytes</i> , the number of <i>Kbytes</i> used, available and percentage of <i>Kbytes</i> used.
du	<i>du(1)</i>	Reports the number of <i>Kbytes</i> used for all files in a directory and the total for the directory.

UNIX SYSTEM ADMINISTRATION COMMANDS - page 2

COMMAND	CHAPTER	DESCRIPTION
env setenv	<i>env(1)</i> <i>cs(1)</i>	<p>These commands are used to read (<i>env</i>) or modify (<i>setenv</i>) the workstation environmentals. The environmentals define the environment for command execution. Several of the environmentals are set when the user logs in. The <i>.login</i> and <i>.cshrc</i> files will define these for the life of the shell. Some environmentals are set by the system at login time.</p> <p>The user can set or change specific environmentals within a <i>shellscript</i> and these environmentals will live for the length of that shellscript.</p> <p>Environmentals define such things as: the time zone that the user is in; the type of terminal that the user is on; where the command interpreter is to look for commands that the user enters (this is called the search rule); what the prompt will look like; where the user home directory is; what shell the user will be using; what the user login name will be; where the user mail box is located.</p>
fsck	<i>fsck(1M)</i> <i>Vol IIB</i>	<p>This is a UNIX diagnostic for maintaining the integrity of the filesystem on the disk. FSCK audits and interactively repairs inconsistent conditions for system files. If the filesystem is consistent, then the number of files, number of blocks used, and number of blocks free are reported.</p>

UNIX SYSTEM ADMINISTRATION COMMANDS - page 3

COMMAND	CHAPTER	DESCRIPTION
fsock - continued		If the filesystem is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. (more later)
labelit	<i>volcopy(1M)</i>	Used to print the existing filesystem label or create a new label.
mkboot	<i>mkboot(1M)</i>	Used to create a <i>bootable</i> backup tape. This tape contains the stand-alone utility program used to rebuild a corrupted disk drive. It also contains the ROOT and USR filesystems.
mkfs	<i>mkfs(1M)</i>	Used to construct a new filesystem on the disk.
mknod	<i>mknod(1M)</i>	Used to build special device files. These files serve as interfaces for the I/O equipment. When the user adds I/O devices (such as a modem), the user must create a device file for that new device.
nohup	<i>cs(1)</i>	This command allows a program (once started) to complete even after the user logs out (hangup). When the user runs a job detached (supplying the & character as an argument to the command), it is effectively <i>nohuped</i> .

UNIX SYSTEM ADMINISTRATION COMMANDS - page 4

COMMAND	CHAPTER	DESCRIPTION
passwd	<i>passwd(1)</i>	Used to change an account login password.
mount	<i>mount(1M)</i>	Used to mount (attach) a removable filesystem to a named directory. When this is done, the files of the filesystem are accessible through the special device file named as an argument of the mount command. i.e. mount /dev/siOf /usr
umount	<i>mount</i>	Used to unmount a filesystem.
set	<i>csh(1)</i>	Used to print the value of shell variables or set shell variables to some value. i.e. set history=20 The above example would allow the history file to contain 20 entries. The <i>.login</i> and <i>.cshrc</i> files may contain several set commands, therefore, everytime the user logs in, the desired values would be set to specific variables.
sgilabel	<i>sgilabel(1M)</i>	Used to read or change the SGI label of a disk drive.

UNIX SYSTEM ADMINISTRATION COMMANDS - page 5

COMMAND	CHAPTER	DESCRIPTION
source	<i>cs(1)</i>	<p>This command will read and execute commands contained in a shellscript.</p> <p>An example of this would be: source .login.</p> <p>The above command would be executed if the user had modified the contents of the .login file (changing some of the environmentals) and wanted the new environmental conditions set for the remaining life of the shell.</p>
stty	<i>stty(1)</i> <i>termio(7)</i>	<p>This command is used to read or set options for terminals, modems, or printers. It defines things like: start and stop characters; eof; tab stops; carriage return. etc...</p>
sync	<i>sync(1)</i> <i>cron(1M)</i>	<p>Update the super block. This command is executed when the user logs out. It pages-out dirty files and insures filesystem integrity.</p> <p>When a user goes from single-user to multi-user, several programs are started and run in the background (by the INIT process). These processes run for the life of the shell and are called <i>daemons</i>.</p> <p>One daemon, the <i>cron daemon</i>, executes commands at specified dates and times according to the instructions in the file <i>/usr/lib/crontab</i>.</p>

UNIX SYSTEM ADMINISTRATION COMMANDS - page 6

COMMAND	CHAPTER	DESCRIPTION
<code>sync</code> - continued		One of the commands in <code>crontab</code> is <code>atrun</code> . This program runs every 10 minutes (default), and one of the things it does is execute <code>sync</code> .

Therefore, every 10 minutes the filesystem is updated. This is good to know if your system is hung and the keyboard locked-out. You could wait for the disk access light to blink on (indicating `sync`), then depress `reset` on the back of the IRIS. You will now be assured that no critical open files exist; using `reset` will not trash your filesystem if done this way.

<code>telinit</code>	<i>init(1M)</i> <i>inittab(4)</i> <i>getty(1M)</i>	<p>Init is a general process spawner (it starts processes). Its primary role is to create processes from script stored in the file <code>/ect/inittab</code>. This file usually has <code>init</code> spawn <code>getty</code> processes for each terminal line that a user may be using. The <code>getty</code> process will set the terminal type, modes, speed, and line discipline.</p> <p>Init is executed once; when the user goes from single-user mode to multi-user mode.</p> <p>If the user ever changes the file <code>inittab</code>, say because an <code>ASCII</code> terminal is added to port two of the IRIS, and the user wants access (to the system) from this terminal, then the user would execute <code>telinit q</code>.</p>
----------------------	--	--

UNIX SYSTEM ADMINISTRATION COMMANDS - page 7

COMMAND	CHAPTER	DESCRIPTION
telinit - continued		Telinit is <i>linked</i> to init, and it directs the actions of init. <i>Telinit q</i> tells init to re-examine the <i>/etc/inittab</i> file. The terminal just added would now be seen and a getty process would be started for that line. (more later)
umask	<i>umask(1)</i> <i>csh(1)</i>	Set file-creation-mode-mask. When this command is placed into the user .login file, the default file protection characters can be changed to what the user wants them to be. (more later)

NOTE:

All of the commands outlined in the lesson will be covered by one or more of the various lab projects that you required to complete. You will get a chance to execute the commands and experience their output and results.

COMMAND SEARCH RULE

When you enter a command on the command line, or use a command in a shellsript, the command interpreter (csh or sh) will go looking for the command.

Where and how the command interpreter looks for the command is called the *search rule*. The *search rule* is defined by the system variable called **PATH**. **PATH** is usually defined by the user's *login file*. The search rule will remain in effect for the life of the shell.

The user can change the search rule (while in the shell) using the **set** command. The new rule would remain in effect until the *.login* file was executed again using the **source** command or **login** command.

The following is an example of a typical search rule:

PATH = . /bin /usr/local/bin /bin /usr/bin

WHERE:

dot (.)	Implies the current directory.
/bin	This is the <i>bin</i> directory of the user. i.e. Assume an account named <i>rich</i> . The pathname for this account would be: <i>/usr/people/rich</i> . The directory <i>/bin</i> is where the user places his/her own special commands. <i>/bin = /usr/people/rich/bin</i> .
/usr/local/bin	This is where special commands created by the user are placed. The commands are not UNIX supplied commands, but, ones maintained locally by the customer or user. i.e. 2700 or q.
/bin	This directory holds UNIX System executable programs.
/usr/bin	Additional UNIX executable programs.

FILE AND DIRECTORY PROTECTION

Files and directories in the UNIX System are protected by a 3 digit code. The code will determine who can Read or write a file and if a file is executable.

Access to directories and files is allowed to the following class types:

FILE PERMISSIONS - Three classes

- User: The owner or creator of the file.
- Group: Members of a specified group.
- Others: The general population.

DIRECTORY ACCESS - Three types:

- Read: The directory may be read as a file.
- Execute: Names in the directory are accessible.
- Write: Directory entries may be created or removed.

FILE CREATION MODES

The protection code is actually a 12 bit value that is located in a file or directories *i-node*.

The 12 bits are divided into 4 groups, with each group having 3 bits. For illustration each bit in the groups has been assigned the decimal weight of its binary equivalent. The following symbols have also been assigned for this illustration:

s4s2s1 - u4u2u1 - g4g2g1 - o4o2o1

sss These bits are used by the *system*.

uuu These bits are for the *user*.

ggg These bits are for the *group*.

ooo These bits are for *other*.

s4 Set user ID bit.

s2 Set group ID bit.

s1 This is the sticky bit.

u4 File is readable.

u2 File can be modified (write).

u1 File is executable.

g4 File is readable.

g2 File can be modified (write).

g1: File is executable.

FILE CREATION MODES - continued

- o4** File is readable.
- o2** File can be modified (write).
- o1** File is executable.

When a new account is created on the workstation, the system assigns the following default values to the protection codes: **777** for new directories and **666** for new files.

By placing an *umask 022* command into the user *.login file*, the file creation modes will be *masked* to be set for **755** for new directories created in that account.

If after the user creates a file and later wants to change the protection code, the *chmod* command must be used.

i.e. `chmod 744 file1`

The above command would change the protection code for file1 to:

- user** Read, write, and file is executable.
- group** File can only be read.
- other** File can only be read.

PROTECTION MODE EXAMPLES

Below is the output produced as a result of executing two commands: `pwd` and `ls -al`.

The `pwd` command prints the pathname of the current directory (`/usr/people/student`) and the `ls -al` command produces a long listing of the directory contents.

The long listing supplies the user with information about the directory and its contents. Such as: who owns the directory, who owns the various files and any sub-directories, the protection codes for all the files and sub-directories, the group that the individual files and directories belong to.

JULIE 1# `pwd`

`/usr/people/student`

JULIE 2# `ls -al`

total 26

<code>drwxr-xr-x</code>	2	student	user	160	Jun 5 16:53	/
<code>drwxr-xr-x</code>	10	bin	bin	160	May 29 14:23	../
<code>-rw-r--r--</code>	1	root	sys	200	May 19 13:38	.cshrc
<code>-rw-r--r--</code>	1	root	sys	180	May 19 12:02	.cshrcDEFAULT
<code>-rw-r--r--</code>	1	root	sys	209	May 19 07:50	.login
<code>-rw-r--r--</code>	1	student	user	209	May 19 11:39	.loginDEFAULT
<code>-rwxrwxrwx</code>	1	student	user	17408	May 19 13:19	print*
<code>-rw-rw-r--</code>	1	root	sys	87	May 19 12:53	print.c
<code>-rwxrwxr-x</code>	1	root	sys	25	May 19 12:53	xyz*
^		^	^		^	^
modes		owner	other		date	name
	^		^	^		
	link	group	size (in bytes)			

Mode field format: **d** = directory; **-** = file; **r** = read; **w** = write; and **x** = executable.

FILE PROTECTION SUMMARY

- **ACCESS MODES**
 - Read, Write, & Execute (rwx)

- **ACCESS CLASSES**
 - User, Group, & Others (ugo)

- **MODIFY ON COMMAND (after creation)**
 - Use chmod(1)
 - Syntax: chmod [mode] [file or directory name]

- **MODIFY AT CREATION TIME**
 - Use umask(1) in .login
 - Syntax: umask [mode]

- **CHANGE GROUP ID**
 - Use chgrp(1)
 - Syntax: chgrp [group] [filename]

- **CHANGE OWNERSHIP**
 - Use chown(1) (must be super-user)
 - Syntax: chown [owner] [filename]

STANDARD INPUT & OUTPUT

Many programs need to read commands and data from the user and write messages to the user. Therefore, the shell prepares some *standard-I/O connections* for each program: **Standard Input** and **Standard Output**.

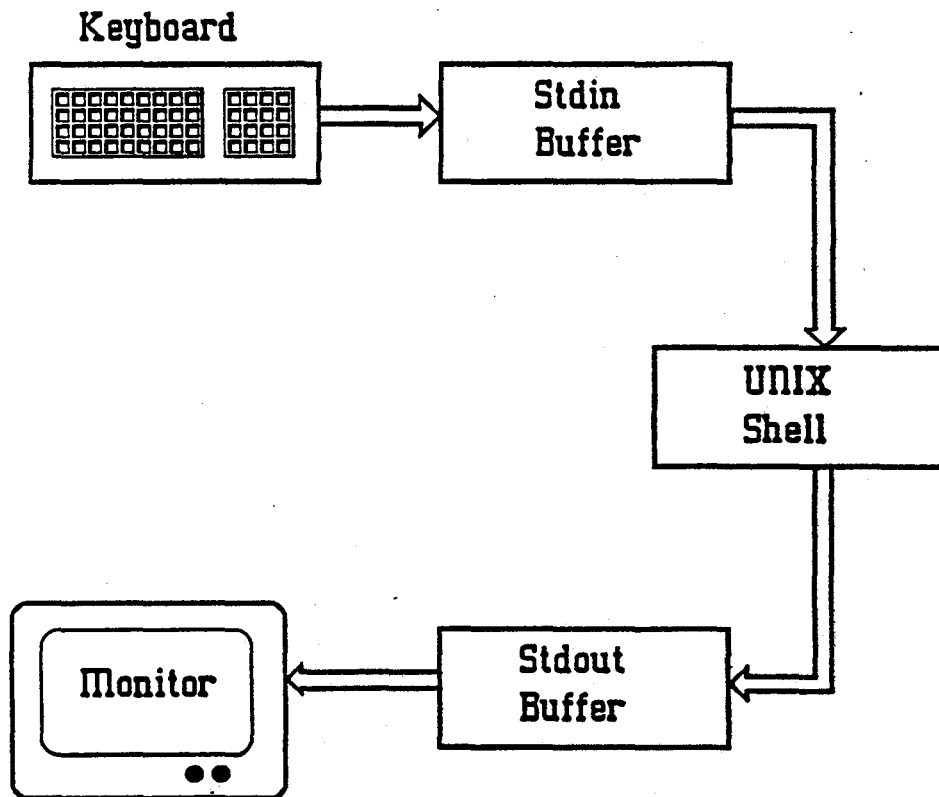
- *stdin* and *stdout* are buffered stream files.
- At login, *stdin* and *stdout* files are opened for transactions.
- *stdin* is a read-only file.
- *stdout* is a write-only file.
- By default, *stdin* is received from the terminal keyboard, and *stdout* is flushed to the terminal screen.
- Both *stdin* and *stdout* can be redirected by the shell.

i.e.

- `ls > file1`
- `more file1 > file2`
- `mail rich < letter`

STANDARD INPUT & OUTPUT DIAGRAM

The diagram below illustrates the data flow for *stdin* and *stdout*.



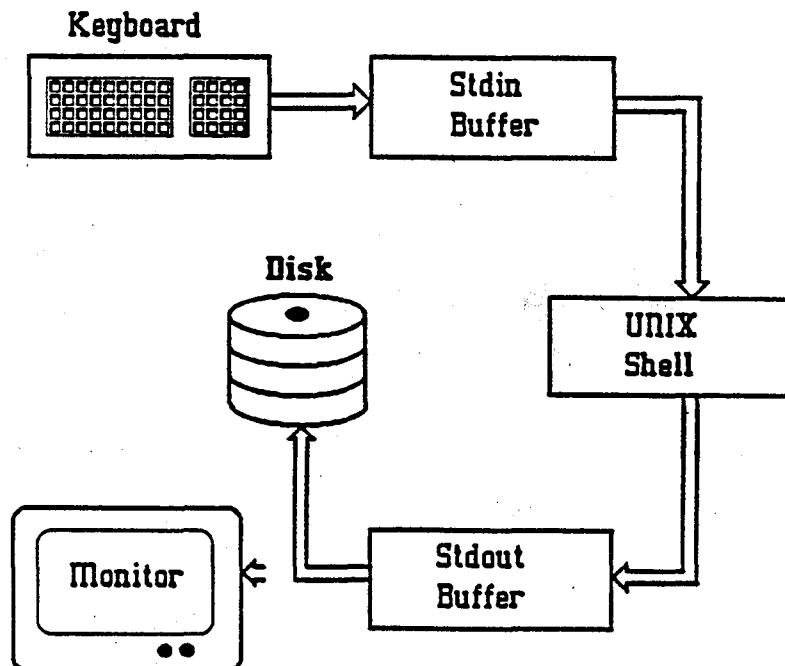
REDIRECTING I/O: Output to a file

By using the "greater-than" character (>), the *stdout* of a command or program can be redirected into a file instead of the terminal screen.

The below text illustrates this operation. First the contents of a directory (*/usr/people/student*) is listed using the `ls` command. Next the `ls` command is again executed, but, this time it's output is redirected into a file. The contents of the new file is printed using the `more` command to show that the output from the `ls` command did indeed go into the file. The `ls` command is once again executed to show the new file in the directory.

```
JULIE 1 > ls
print print.c xyz
JULIE 2 > ls > file1
JULIE 3 > more file1
file1
print
print.c
xyz
JULIE 4 > ls
file1 print print.c xyz
```

The diagram below illustrates the data flow for the above operation.



REDIRECTING I/O: Append to a file

By using two "greater-than" characters (`>>`), the *stdout* of a command or program can be redirected and appended to a file. If the file already exists, then the data is placed at the end of the file, and if the file does not exist, then the shell will create the named file.

For illustration of the operation the example shown on the previous page will be expanded. The pathname of the current working directory will be printed using the `pwd` command, but, its output will be appended to the file that was created in the previous example (`file1`). The file will then be printed using the `more` command to show the result of the redirection.

```
JULIE 5 > pwd
/usr/people/student
JULIE 6 > pwd >> file1
JULIE 7 > more file1
file1
print
print.c
xyz
/usr/people/student
```

As you can see, the output from the second `pwd` was redirected and appended to the file.

REDIRECTING I/O: Input from a file

By using the "*less-than*" character (<), the *stdout* of a file can be redirected to the *stdin* of another command.

The same approach used on the preceding pages will be used to illustrate this operation. The contents of a shellscript file (*xyz*) will be printed using the **more** command. You will see that the shellscript (an executable command) file contains two commands: **echo** and **ls**. Next, the *stdout* of the shellscript file will be redirected to the *stdin* of the **write** command. This command will send a message to another user on the system; the user in this case will be *student*, therefore, the output from the shellscript will be sent to *student*, who by the way, is the assumed user in these examples.

The following is what *student* would see at his/her screen if the following was performed.

```
JULIE 8 > more xyz
echo Hey, this works!
ls
JULIE 9 > write student < xyz
```

```
Message from student (ttyd1) [date] . . .
echo Hey, this works!
ls
<EOT>
```

NOTE:

You will get an opportunity to experiment with these exact examples and others later in the course when you are performing the various lab projects.

THE PIPE OPERATION

The (`|`) character is the operator for a special UNIX Shell operation called *piping*. The *stdout* from an *executing* command on the left side of the *pipe*, is sent as *stdin* to the command on the right side of the pipe.

i.e.

```
JULIE 10 > pwd | wall
```

```
Broadcast Message from student (ttyd1) [date] ...  
/usr/people/student
```

The *stdout* of the above `pwd` command was *piped* to the *stdin* of the `wall` command. The `wall` command will broadcast a message to all users on the system. In this case the message was the output from the `pwd` command.

The pipe concept allows programs or commands to be *coupled* together. The output of one program becomes the input to another, etc.

i.e.

```
JULIE 11 > sort file1 | lpr
```

The example above: `sort` the contents of the file `file1` and pass the output to the printer command `lpr` for printing.

VI: A TEXT EDITOR

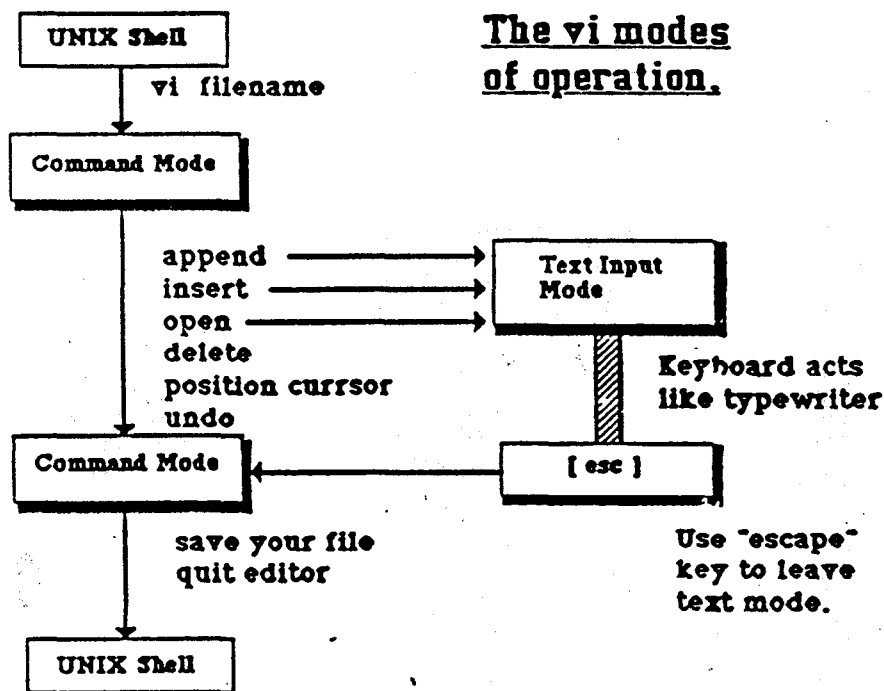
A text editor is a general purpose program used to prepare text files. A text editor enables a user to enter, change, and correct text. Most editors contain commands to locate specific lines or words in the text and commands to add, delete, change, and print lines in the text.

Vi is an interactive text editor designed to be used with a crt terminal. It produces a *window* into the file you are editing. This window lets you see about 40 lines of text of the file at a time, and you can move the window up and down through the file. You can move to any part of any line on the screen and can make changes there.

Although vi is a sophisticated editor with a large number of commands, the basic structure of vi is very simple. There are two modes of operation, the *Command Mode* and the *Text Input Mode*, as shown in the following diagram.

COMMAND MODE: Most of the commands are used to position the cursor or find text. The rest of the commands either delete something or place you into the *Text Mode*.

TEXT MODE: This mode allows you to enter new text into the file.



Contents

LAB 3.1: Basic UNIX Commands	1
SECTION 1: Instructional Procedure	2
I. BASIC UNIX COMMANDS	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT.	2
V. PROCEDURE	2
SECTION 2: Review	64

LAB 3.1: Basic UNIX Commands

This lab project contains two sections:

- Section 1:** This section is used to teach you how the UNIX Survival commands function and how to use each command. It allows you to practice many of the commands and see what type of output occurs as a result of executing the various commands.

- Section 2** This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. BASIC UNIX COMMANDS

These are the commands defined by lesson three of this course. Of the hundreds of UNIX commands, these are the commands that a Field Engineer must have a good understanding of in order to survive on the IRIS workstation. These commands will allow the Field Engineer access to UNIX, allow movement within UNIX, and teach the Engineer how to access required data files.

What commands are not covered by this lab project will be covered in subsequent lab projects of this course.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly execute any of the *Basic Commands* presented by this lab project.
- B. Correctly explain what a command argument is.
- C. Correctly explain what a command option is and how it is invoked.

III. PURPOSE

Teach the student how to use the *Basic Commands* and allow each student sufficient time to practice each of these commands.

IV. EQUIPMENT.

A functional IRIS 68020 based workstation.

V. PROCEDURE

Two students will be assigned one IRIS workstation. Each student will perform the lab project from one of the two ASCII terminals attached to *ports two or three*.

Special accounts have been created for the classroom environment to be used by the students.

The account names are *student* and *student1*. Both accounts are identical, so choose one and keep that account name throughout the remainder of the course.

It is very important that you follow the lab project precisely and do not wander off in your own direction. You will have ample time for *free-time* later in the course.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

1. **iris >** The prom monitor.
2. **JULIE n >** Student account prompt when in Multi-user mode, where n = command number.
3. **JULIE n #** Student account prompt when in Super-user mode, where n = command number.
4. **#** Single user mode.
5. **JULIE login:** This prompt appears when you enter Multi-user mode.

The following text will lead you through the execution of many of the **BASIC COMMANDS**. When you are required to enter a command, the text recreates the **STANDARD OUTPUT** for that command. Just keep reading and following the flow of the lab project.

Also throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **JULIE 1 > pwd**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in normal print.

BEGIN ACTUAL PROCEDURE HERE:**STEP 1: Power on the workstation.**

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)
Configuration Switch: 0x0000
Multibus Window (2mb) at Megabytes 0 and 1.
Multibus accessible memory (1mb) begins
  at Physical memory page 300,
  at Virtual address 2000000.
iris>
```

STEP 2: Boot the UNIX System.

The workstation must be booted from the main workstation terminal that is connected to port 1. Once UNIX is up and in multi-user mode, each student will continue the project using his/her individual ASCII terminal.

Before the following command is issued, power on each ASCII terminal. The power-on switch is located on the back of the ASCII monitor. If the monitor is already on, then power it off and then back on. This will clear the screen and a single cursor should appear. Now, back at the primary workstation terminal, enter the following command:

```
iris > b
```

This will cause the UNIX kernel to be loaded into memory. The file *defaultboot* will be searched for on the hard disk and it will boot the kernel. (remember, the configuration switches are selecting the hard disk as the boot device)

The following output "blinks" on the screen.

SGI Extent Filesystem**Loading: md:0:defaultboot****Text: 038318 bytes****Data: 0113d8 bytes****Bss: 024d7c bytes****Jumping to load program ~ 20000400**

After the above message is displayed, the screen is cleared and the message shown next is output sequentially as the configuration process proceeds.

SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]**(C) Copyright 1986 - Silicon Graphics Inc.****real = 4194304****kmem = 561152****user = 3633152****bufs = 819200 (max=16k)****dsd0 not installed****qic0 not installed****sii0 at mbio 0x07200 ipl 5****si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0****si1 not installed****sf0 floppy (80/2/8) slave 2****siq0 at mbio 0x73fc ipl 5****sq0 (qic02 cartridge tape) slave 0****iph0 not installed****tmt0 not installed****ik0 not installed****nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2****fpa installed****lpen not installed****kernel debugger disabled.****root on si0a****swap on si0b. swplo=0 nswap=64000****INIT: SINGLE USER MODE**

**Welcome to the world of the
IRIS personal workstation.**

**The SGI education department
hopes that you have a very
pleasant and informative stay**

with us.

#

The UNIX Kernel has been loaded and the workstation is in *Single-user* mode. Remember, single-user is reserved for system maintenance and system administration tasks. We will not perform any of these type tasks at this time, but, cover them in another lab project.

STEP 3: Going to multi-user mode.

```
# multi
```

The following output and prompts will appear at the workstation monitor.

```
#
```

```
INIT: New run level: 2
```

```
Is the date [day month date time time-zone year] correct? (y or n)
```

This is asking you if the date is correct. If it is, enter **y**. If the date is incorrect, enter **n**. You would then be prompted to enter the date as the following example shows: *062312308600*. The last four digits (8600) are optional and are not required, but, it is a good idea to enter at least the year (86).

If you had to correct the date, you will again be asked if it is correct.

```
Do you want to check filesystem consistency? (y or n)
```

This is asking you if the program *fsck* should be run to check the filesystem consistency (integrity). Enter **y** to the prompt.

We will talk about this check later in the course. If your filesystem is consistent, then the following message is output:

Checking file systems for consistency:

```
/dev/si0a
```

```
File system: root Volume: SGI
```

```
** Phase 1 - Check Blocks and Sizes
```

```
** Phase 2 - Check Pathnames
```

```
** Phase 3 - Check Connectivity
```


**** Phase 4 - Check Reference Counts**

**** Phase 5 - Check Free List**

nnn files nnnnn K used nnnn K free

/dev/si0f

File system: usr Volume: SGI

**** Phase 1 - Check Blocks and Sizes**

**** Phase 2 - Check Pathnames**

**** Phase 3 - Check Connectivity**

**** Phase 4 - Check Reference Counts**

**** Phase 5 - Check Free List**

nnn files nnnnn K used nnnn K free

Mounting: /usr

Preserved editor files

Cleared /tmp

Resetting locks and logs

Hostname: JULIE

Daemons:

update

cron

xnsd

lpd

lpsched

Daemons started

JULIE login:

At this point the system is up in multi-user mode, the background daemons are running and the system is waiting for users to login to their accounts.

The INIT process (started when you entered *multi*) started the daemons. One of the functions of INIT is to look in a file called *inittab* and determine what terminals are attached to the workstation. Any defined terminal will be sent a login prompt. Since the two ASCII terminals are attached to the workstation and defined in the *inittab* file, the system will now attempt to display the login prompt at each terminal.

INIT starts a process called *getty* for each attached and defined terminal. The *getty* process will access another file called *gettydefs*, this file is used to determine what the *baud rate* of the terminal is and insure that the correct rate is selected.

The *getty* process assumes an initial baud rate of 9600 baud (this is because this baud rate is defined by the first line in the *gettydefs* file). If the baud rate of the ASCII terminal is not 9600

baud, then *garbage* would appear at the terminal when the login message is sent to the terminal. Depressing **THE BREAK KEY** will force the *getty* process to access the next line in the *gettydefs* file. The second line defines a baud rate of 4800 baud. If garbage still appears at the terminal, depressing the *break key* again will force the *getty* process to go access the next line of *gettydefs*. This process continues until the *getty* process finds the correct baud rate for that terminal.

Additional information will be given later in the course about this process. The contents of these files will be looked at then. This information is given here to help you understand how the system works.

Each student should now move onto his/her own ASCII terminal to complete the lab project.

STEP 4: Login to the student(1) account.

JULIE login: student or student1

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

TERM = (v50am)

The last line of the above output is actually a prompt. It is asking you if this terminal is a *Visual 50* (the *Visual 60* functions the same as the *Visual 50*). If the terminal is a *Visual 50* or *60*, by depressing **return** you are answering **yes** to the question.

If the terminal is of another type, you must enter the name for that type of terminal. All of the terminal types that the IRIS workstation supports are defined by the file *termcap*.

If you do not remember the mnemonic name for the type of terminal that you are using, then you would have to print the contents of *termcap* using the *more* command from the primary workstation terminal. Once you obtained the correct mnemonic, then you could return to the ASCII terminal and enter the correct terminal type to the prompt.

Since the classroom terminals are either Visual 50 or 60 models, depress return to answer *yes* to the prompt. If the terminal was another IRIS terminal, then you would enter *rwsiris*.

Once you have responded to the prompt, the system accesses the file *termcap* to fetch all the configuration data that defines the terminal. This way the system will understand how to communicate with the monitor and how to interpret the individual keystrokes from the keyboard.

Once you depress return or enter the correct terminal type, the following prompt appears at the terminal:

```
JULIE 1 >
```

STEP 5: Where am I; who am I; how do I move around.

When you want to determine what directory you are currently in, use the **pwd** command. This will print the working directory.

```
JULIE 1 > pwd
```

```
/usr/people/student
```

NOTE:

The above output indicates the student account. If you are *student1*, your *stdout* (standard output) would be the same except the message would be: */usr/people/student1*. This difference of the *stdout* will be consistent throughout the lab project since I was logged in as *student* when I produced this project.

When you login, UNIX places you into your *home* directory. This is the point that you are now at.

NOTE:

If you repeat the command just executed, or execute another command on your own, the prompt number shown by the lab project and your prompt number will get out of sync. No big deal, just keep this in mind so that you will not get confused when the two numbers get out of sync.

The next command will show you who you are. This would be useful if you approached a workstation that was in a shell and you wanted to determine who's account it was.

JULIE 2 > whoami

student

The next command is useful when you want to determine who is on the workstation network with you.

JULIE 3 > who

```
student ttyd1    Jun 7 16:20  
rich    console  Jun 7 12:13
```

At this point open your UNIX PROGRAMMER'S MANUAL, Vol 1A, to the manual page for the **who** command.

The **who** command may be executed with any of a number of options. To invoke an option or options, enter the mnemonic for the desired option preceded by the minus (-) character. *White space* (at least one space) must separate the command and the option. More than one option can be selected, but, you only need one minus character. **DO NOT** place spaces between the options if you are using more than one.

The options are optional entries to the command as indicated by the *brackets* [] surrounding the available options for this command: i.e. [-uTlpdbrtas].

In a moment you will be instructed to execute the **who** command again using one of the options, but, lets now look at the output that is produced by this command.

The general format for output is:

name [state] **line** **time** **activity** **pid** [comment] [exit]

The lab project highlighted the three fields that were output as a result of executing **who** without any options.

Lets now execute the **who** command using the **u** option.

```
JULIE 4 > who -u
```

```
student  ttyd1  Jun 7 16:20 0:08  415  
student1 ttyd2  Jun 7 12:13 .    76
```

Notice the difference in the output of the two commands. Now take a few minutes and read over the descriptions of the options, returning to the flow of the lab project when finished.

Of all the options, the **u** is the one most useful to you. You could use it to get the *process ID number* of a terminal that is locked up. You could kill the process from another terminal using the **kill** command and the process ID number. (more on this later)

Do not concern yourself with the optional argument [file] for this command.

Now, if you want, try executing the **who** command using some of the other options. Try using more than one option at a time to see what happens. When you are finished, continue with the lab project.

.... NOW, ON WITH THE SHOW.

NOTE:

As the lab project progresses, many additional commands will be demonstrated. You should always open to the correct manual page when a new command is discussed. The lab project will make reference to options and arguments of the command being discussed and it will assume that you are open to the correct page.

The next command is used to print (display) a list of a directory contents. The command is called **ls**, which stands for *list*. It has many useful options. If you use **ls** without a *name* argument, then it lists the contents of the current directory. If you supply a pathname, then you can list another directory (i.e. /etc) or an individual file (i.e xyz).

Lets list the contents of the current directory (your home directory).

```
JULIE 5 > ls
```

```
print print.c xyz
```

You see that the directory contains three names: *print*, *print.c* & *xyz*. Are these files or directories or a combination of both? Are the files executable or data files? Are the names linked to other names? Who owns the files? When was the last time they were modified? Are the files special files? Lets answer a few of these questions.

```
JULIE 6 > ls -l
```

```
total 20
```

```
-rwxrwxrwx 1 student user 17408 May 19 13:19 print
-rw-rw-r-- 1 root sys 87 May 19 12:53 print.c
-rwxrwxr-x 1 root sys 25 May 19 12:53 xyz
```

The above command `ls -l` will answer many of the questions. (refer to the workbook page entitled *Protection Mode Examples* in section three)

We can see that all the names are filenames (the first - in the mode field). Files *print* & *xyz* are executable (the last x in the mode field). We can see who owns the files and to what group they belong. The size of the files in Kbytes is shown and the date they were last modified.

The next command lists all the files in the current directory. The previous `ls` commands did not list filenames that begin with a period (.).

```
JULIE 7 > ls -a
```

```
..          .cshrc          .login          print          xyz
..          .cshrcDEFAULT  .loginDEFAULT  print.c
```

We now see that several other files populate the directory. These are the *.login* & *.cshrc* files that have been mentioned many times throughout the course. These two files are accessed every time you start your shell (login). They contain statements that define parameters and configurations that your shell will operate under.

Later in the lab project we will look inside these files, and later in the course they will again be discussed.

Notice that there are two files that have **DEFAULT** after the first part of the name. These are two files placed into this directory by the education department and would not exist in an actual directory. They are here for illustration and will be discussed later when the *.login* & *.cshrc* files are discussed.

Lets now print a long listing of all the files in the directory.

JULIE 8 > ls -al

```
total 26
drwxr-xr-x  2 student user   176 Jun  7 16:21 .
drwxr-xr-x 10 bin      bin   160 May 29 14:23 ..
-rw-r--r--  1 root    sys   200 May 19 13:38 .cshrc
-rw-r--r--  1 root    sys   180 May 19 12:02 .cshrcDEFAULT
-rw-r--r--  1 root    sys   209 May 19 07:50 .login
-rw-r--r--  1 student user   209 May 19 11:39 .loginDEFAULT
-rwxrwxrwx  1 student user 17408 May 19 13:19 print
-rw-rw-r--  1 root    sys    87 May 19 12:53 print.c
-rwxrwxr-x  1 root    sys    25 May 19 12:53 xyz
```

Notice the first two directories in the above listing: (.) and (..). Dot is the *current* directory and is owned by you and its in the *user* group. Dot Dot is the *parent* directory and it belongs to *bin*. This is a special account used by the system.

The next example shows you how to get the corresponding *i-node* of a file or directory. (The *i-node* numbers you see may be different from the ones illustrated, and why you may want these numbers will be covered later in the course)

JULIE 9 > ls -i

```
4309 print          4310 print.c       4311 xyz
```

The next example will mark each name in the directory with a special character that identifies the type of file that it is: Executable file = * ; directory names = / ; if the file is linked to another file = ~ or !.

JULIE 10 > ls -F

```
print*           print.c          xyz*
```

You should now try a few of the *ls* options on you own, but, let me suggest a few examples to try and make some additional comments.

All of the commands prior to this point in the lesson have been executed *relative* to the current directory. Most of the UNIX commands can also be executed using an *absolute pathname* to effect files and/or directories other than the current directory; This means that you do not have to be in a directory to execute, modify, move, copy, etc., files. (Note: file and directory permissions may inhibit you from doing some of these things)

While you are experimenting with the `ls` command, try the following two commands, they illustrate the point about absolute pathnames.

1. `ls /etc` (List the directory *etc*)
2. `ls -l /etc/motd` (List the file *motd* in the *etc* directory)

Return to this point in the lab project when you are finished experimenting with the `ls` command.

Up to this point you have been in your home directory where the system placed you when you logged on. We will now learn how to move around within the UNIX tree.

The next command will take you to the top of the UNIX tree by using the `cd` command. This is the Change Directory command. The directory you are going to is called *ROOT* and its path-name is `/`.

```
JULIE 11 > cd /
```

Notice that no *stdout* is produced by this command. If the directory exists, then the shell takes you there, else, an error message is displayed indicating that the directory does not exist. You should trust the shell, but, for those skeptics, lets print the working directory name.

```
JULIE 12 > pwd
```

```
/
```

You see, we are in the *ROOT* directory (`/`).

Now to make a point, enter the following `cd` command; this directory does not exist.

JULIE 13 > `cd /abc`

/abc: No such file or directory

Lets now list the contents of the UNIX ROOT directory.

JULIE 14 > `ls`

<i>Versions</i>	<i>defaultboot</i>	<i>kernels</i>	<i>ovmunix</i>	<i>usr</i>
<i>bin</i>	<i>dev</i>	<i>lib</i>	<i>stand</i>	<i>vmunix</i>
<i>d</i>	<i>etc</i>	<i>lost+found</i>	<i>tmp</i>	<i>vmunix1</i>

The next command will give us a long listing of the directory.

JULIE 15 > `ls -al`

```
total 1347
drwxr-xr-x 13 root sys 384 May 29 15:54 .
drwxr-xr-x 13 root sys 384 May 29 15:54 ..
-rw-r--r-- 1 root sys 41 May 6 17:41 .cshrc
-rw-r--r-- 1 root sys 144 May 16 10:33 .login
-rw-r--r-- 1 root sys 121 May 16 10:33 .profile
drwxr-xr-x 2 root sys 128 May 16 10:47 Versions
drwxr-xr-x 2 bin bin 1856 May 16 10:51 bin
drwxr-xr-x 2 bin bin 32 May 2 19:40 d
-rwxr-xr-x 2 root sys 336896 May 16 10:51 defaultboot
drwxr-xr-x 3 bin bin 3008 May 16 11:06 dev
drwxr-xr-x 3 bin bin 1424 Jun 7 12:13 etc
drwxrwxrwx 2 root sys 48 May 16 10:35 kernels
drwxr-xr-x 2 bin bin 160 May 16 10:36 lib
drwxrwxrwx 2 root sys 5120 May 16 08:45 lost+found
-rwxr-xr-x 1 root sys 336896 May 7 07:46 ovmunix
drwxr-xr-x 2 bin bin 80 May 16 10:36 stand
drwxrwxrwx 2 bin bin 3808 Jun 7 16:20 tmp
drwxrwxrwx 19 root sys 304 May 29 14:23 usr
-rwxr-xr-x 2 root sys 336896 May 16 10:51 vmunix
-rwxr-xr-x 1 root sys 336896 May 16 10:42 vmunix1
```

The following text gives general information concerning the files and directories contained in ROOT directory.

- **Dot (.)**
This is the current directory (ROOT).
- **Dot Dot (..)**
This is the parent directory, which, for ROOT is the same as the current directory. ROOT is the only directory that does not have a parent directory. (Notice that the size of the current and parent directory both equal 384 Kbytes. Look back at the output for your student directory and see that there is a difference in the Kbyte size of these two directories.)
- **.cshrc**
This is the file used (by the shell) when a user logs into the system administration accounts of: *root* or *rootcsh*. Every account will have a *.cshrc* file if that user is using the *C-shell* (*csh*). This file defines parameters under which the shell will operate.

Remember, the shell that a user uses, is determined by a statement in a file called *passwd*. (more on this later in the course)

- **.login**
This is the file used by the shell at login time to set additional parameters. The information in the *.cshrc* & *.login* files could be combined since they are both accessed by the shell at login time, and indeed, some users only maintain the *.login* file. (more later)
- **.profile**
This file serves the same purpose as *.cshrc*. It is used when the account calls for the *Shell* (*sh*) command interpreter to be used. Only one account uses *sh* in the basic IRIS software release, and that's *rootsh*.

If a user creates a new account and wants to use *sh*, then the *sh* statement must be placed into the *passwd* file.

- **Versions**
This is a directory that contains coded strings that define the version levels of UNIX and any options installed on the workstation. Later in the course another lab project will access this directory and you will see the type of data that it holds.
- **bin**
This directory holds a majority of the UNIX utilities (commands). This is one of the directories that an account's *search rule* (the *PATH* system variable) should instruct the shell to look into when the user enters a command. This variable is defined in either the *.login* or *.cshrc* files.

- **d**
This is a *generic* directory used to attach (mount) the filesystem called */dev/s1g*. This filesystem is defined by default when a second hard disk is shipped with a workstation. The user could change the name of the filesystem and/or the name of the directory to which it attaches.

This directory could hold whatever files the user wants.

- **defaultboot**
This is actually a pseudonym name (link) for the file *vmunix*. Remember, the PROM monitor looks for this file on the specified boot device if no other filename is supplied with the *boot* command. Notice the size of the file and compare that to the file *vmunix*.
- **dev**
This is the directory that holds all the *special device* files. Remember, these files serve as *interfaces* between UNIX and the I/O devices.
- **etc**
This directory is where most of the system configuration files are maintained. Many of the files have already been mentioned by the lesson material, some of which are: *passwd*, *motd*, *inittab*, *termcap*, *gettydefs* and *many more*. When we get into system administration, we will look at this directory in greater depth.
- **kernels**
This directory will hold different copies of the *kernel*. These copies would be the ones used if certain options were installed on the workstation: *TCP/IP*, *IEEE*, *ect..*
- **lib**
This directory will hold various files needed by the system to support the various options and/or programs: *C*, *FORTTRAN*, *etc*.
- **lost+found**
This directory is used by the filesystem maintenance program *fsck*. *FSCK* places severed directories and/or files here (these are unreferenced files or directories). These are error conditions that *fsck* will attempt to correct if they are detected. (more on *fsck* later)
- **ovmunix**
This is a copy of the old UNIX system. It is created when a new software release is installed on the workstation. It is usually removed when the install process is complete. If you see it on your workstations, ignore it.
- **stand**
This directory holds the *stand-alone* utilities that the Field Engineer needs when rebuilding a corrupted disk drive.

- **tmp**
This is the *temp* directory. The user and UNIX place files here that are temporary. This directory is usually cleared when the system is booted. (look at page seven of this lab project and see the message *Cleared /tmp* in the initialization process)
- **usr**
This is the directory that the *user filesystem* is attached (mounted) to when the system goes into multi-user mode. This filesystem holds the bulk of the system files and directories.

User accounts are created and maintained under this directory; this is where the *student* accounts live.

- **vmunix**
This is the bootstrap program that loads the UNIX kernel. This program is used to load the system from the first hard disk.
- **vmunix1**
This is a link of the file *vmunix*. It is used to load the system when a copy of the kernel is maintained on the second hard disk and you are booting from that disk. (note the file sizes of files *defaultboot*, *vmunix*, & *vmunix1*, they are all the same because one is the actual file, and the others are links to that file)

The next few commands will show you how to move around in the UNIX tree using some *shorthand* methods. First lets return to your home directory. When the lab project instructs you to enter a command containing the account name (*student*), if you are *student1*, then substitute *student1* for *student*.

```
JULIE 16 > cd /usr/people/student(1)
```

The next command proves that you are now home.

```
JULIE 17 > pwd
```

```
/usr/people/student
```

We will now return to ROOT so that I can show you other ways to get home.

```
JULIE 18 > cd /
```

```
JULIE 19 > pwd
```

```
/
```

OK, we are back at ROOT. The following command will take you to your home directory.

```
JULIE 20 > cd ~student(1)
```

```
JULIE 21 > pwd
```

```
/usr/people/student
```

That's one short method to get home. I will now show you another method. The next command will take you to another account (called *rich*) and then we will return to your home.

```
JULIE 22 > cd ~rich
```

```
JULIE 23 > pwd
```

```
/usr/people/rich
```

The `cd ~rich` command took you to the home directory of the account *rich*. You can use (`~`) to get to the home directory of any account.

Before we again return to your home directory let's see what's in the directory */usr/people/rich*.

```
JULIE 24 > ls -F
```

```
bin/          course/      tree
```

This directory contains two subdirectories (*course* & *bin*) and one file called *tree*.

OK, thats nice, lets now return home using the simple method.

```
JULIE 25 > cd
```

```
JULIE 26 > pwd
```

```
/usr/people/student
```

Notice that when you use the `cd` command without an argument, that the shell returns you to the home directory. This will always be the home directory that you logged into, in this case student or student1.

The next series of commands will show you how to use the memory stack for *pushing & popping* directory pathnames. The *pushd & popd* commands are like *jump-to-subroutine & return-from-subroutine* instructions that you might use in a computer program.

The *pushd* command will push the pathname of the *current* directory into a memory stack and then *change directories* to the directory that you name as an argument to the *pushd* command.

```
JULIE 27 > pushd /
```

```
/
```

```
JULIE 28 > pwd
```

```
/
```

The *pushd* command above took you to the ROOT directory and the `pwd` verifies that it worked.

Lets see if the *popd* will return us to the directory that we jumped from.

```
JULIE 29 > popd
```

```
JULIE 30 > pwd
```

```
/usr/people/student
```

OK, so it worked. Why would you want to use these commands?

Some users trees can be very deep with subdirectories. Lets say that you are in one of these trees, in fact, lets go to one:

```
JULIE 31 > cd ~rich/course/3000/workbook
```

```
JULIE 32 > pwd
```

```
/usr/people/rich/course/3000/workbook
```

```
JULIE 33 > ls -F
```

```

appendix/      lesson2      print*      student.job*
lab/           lesson2.copy print.c      student.stuff
lesson1       lesson3      print.c.copy xyz*
lesson1.copy lesson3.copy print.copy*  xyz.copy*

```

The `cd` command brought you to this directory and the `ls -F` lists the contents. As you can see, other directories (*lab* & *appendix*) are available for moving further down in the tree.

OK, back to the question. You are here and want to go somewhere else in the system tree; maybe go look at a file in another user's account.

Getting to the other directory may not be too difficult, but returning here requires much typing when entering the long pathname; it's the perfect place to use the *pushd* & *popd* commands. Lets go to your account's home directory.

```
JULIE 34 > pushd ~student(1)
```

```
~/usr/people/rich/course/3000/workbook
```

```
JULIE 35 > pwd
```

```
/usr/people/student
```

```
JULIE 36 > popd
```

```
/usr/people/rich/course/3000/workbook
```

We jumped to account *student* using `pushd` then returned using `popd`.

It is now time for you to practice moving around the UNIX tree on your own. Go and investigate some of the directories and their contents. **ONLY USE THE COMMANDS WE HAVE LEARNED TO THIS POINT: *ls, pwd, cd, pushd, popd***.

When your curiosity has been satisfied, return to this point in the lab project.

STEP 6: Reading files; rename files; move files; comparing files; and finding files.

The first thing we want to do is make sure that you are back in your home directory.

JULIE 37 > *cd*

JULIE 38 > *pwd*

/usr/people/student

Lets once again list the contents of your home directory.

JULIE 39 > *ls*

print *print.c* *xyz*

The following command is useful if you want to determine what type of file a file is. It is more specific than the *ls* command. For curiosity, lets look at the three files in your directory using the *file* command.

JULIE 40 > *file xyz*

xyz: *commands text*

JULIE 41 > *file print.c*

print.c: *c program text*

JULIE 42 > *file print*

print: *pure executable not stripped*

The first file command tells us that the file *xyz* is a commands text. This means that it is a *shellscript file*. The user can create shellscripts to perform specific tasks. These files contain UNIX commands or other shellscripts.

Lets read the file *xyz* using the *cat* command.

```
JULIE 43 > cat xyz
```

```
echo Hey, this works!  
ls
```

As we can see, the file *xyz* is comprised of two UNIX commands: *echo & ls*.

Lets now execute this command (shellscript).

```
JULIE 44 > xyz
```

```
Hey, this works!  
print print.c xyz
```

Notice that the *stdout*, for the two commands that comprise *xyz*, is what you would expect if you executed the two commands individually.

The second file command (executed at JULIE 41) told you that the file *print.c* contains *C program text*. This means that it contains *C language source code*. Once the file is compiled, it will become an executable file. (the file *print* is the compiled version of the file *print.c*)

Lets look at this file using the *cat* command.

```
JULIE 45 > cat print.c
```

```
#include <stdio.h>
```

```
main ()
```

```
{
```

```
    printf("A simple message is printed.");  
    printf("\n");
```

```
}
```

The third file command (executed at JULIE 42) indicates that the file *print*, is a *pure executable not stripped* file. This means that the file can be executed, so lets try it:

JULIE 46 > print

A simple message is printed.

I have held off reading the file *print*, because, when you read the file, your screen will be filled with what appears to be garbage and your terminal will beep. This is normal when you attempt to read this type of file.

This brings up the point, how are jobs aborted or interrupted?

In a moment I will ask you to read the file called *print*, but, before you attempt this, I must tell you how to stop the process once it starts. At this point two methods could be used:

1. Depress the key marked *CTRL* and while holding it depressed, depress the key *c* (*ctrl-c*). The UNIX and IRIS documentation refer to this function using the symbol (*^c*).

This will stop the process, but, your screen will be full of junk. Use the *clear* command to clear your screen.

2. The second method, depressing the *break* key.

This may cause your terminal to lock-up. If it does, then depress and hold the *function* key followed by depressing the *set up* key. This will cause numbered output to appear at the bottom of your screen if you are on a visual 50 or 60, if you are on a vt100, see your instructor)

1 of several commands would be accepted at this point, you would want to *RESET* your terminal, therefore, you would depress the *seven* key on the *numbers pad*, followed by return. (I will ask you to perform this operation later)

The above mentioned functions could be used to kill other processes, but, they only kill the process that you are attached to, not ones that you started in the background; these processes are killed using the *kill* command. (we will look at this later)

The process that you are attached to is the current process that has a hold of your keyboard and has it locked out. Most commands that you execute run very quickly and you do not care if the keyboard is locked out, but, if using your keyboard is important to you during long running commands or jobs, then, running them in background mode is what you must do. (more on this later)

Now, execute the following `cat` command and clear your keyboard and terminal when the junk appears, then continue the lab project.

```
JULIE 47 > cat print
```

junk will appear.

The command `cat` is not always the best way to read a file. It will print the complete file non stop, scrolling the data past the screen much faster then you could ever read it. It is OK to use when reading small files. Try the following command and watch what happens:

```
JULIE 48 > cat ~/rich/course/3000/workbook/lesson2
```

The contents of this file will now scroll across your screen, use the (`^c`) to stop the process.

A better command to use when reading a file, is the `more` command. Lets again read the three files in your home directory.

```
JULIE 49 > more xyz
```

```
echo Hey, this works!  
ls
```

```
JULIE 50 > more print.c
```

```
#include <stdio.h>
```

```
main ()
```

```
{
```

```
    printf("A simple message is printed.");  
    printf("\n");
```

```
}
```

JULIE 51 > more print

I should not have to tell you what to do at this point.

The **more** command will display a page at a time of the file. By depressing the *space bar* you can scroll down the file by pages. By using the *return key* you can scroll a line at a time. Merely type **q** to terminate the read.

Lets again read the long file attempted at JULIE 48, this time using the **more** command; controlling the read using the space bar and return keys.

JULIE 52 > more ~rich/course/3000/workbook/lesson2

The contents of the file will now appear.

The next series of commands will show you how to move files and how to rename a file.

First, lets again list the contents of your home directory.

JULIE 53 > ls

print print.c xyz

The next command shows you how to *rename* a file. The command used to do this is the **mv** command. This stands for *move*. It will rename the file *xyz*, giving it a new name that you will call *new.name*.

NOTE:

To find the command description and syntax for the **mv** command, see **cp(1)** in your Vol 1A book.

JULIE 54 > mv xyz new.name

Lets see the change.

JULIE 55 > ls

new.name print print.c

Lets now change the name back to what it was.

```
JULIE 56 > mv new.name xyz
```

```
JULIE 57 > ls
```

```
print print.c xyz
```

The following command will move a file into your directory from another directory in the tree. You will bring over the file using the same name, but, you could have also renamed the file. **Note that there are two steps for JULIE 58. Execute the one that applies to you. Student1 will bring over a copy of lesson1 and rename it lesson1.**

```
(if student) JULIE 58 > mv ~rich/course/3000/workbook/lesson1 lesson1
```

```
(if student1) JULIE 58 > mv ~rich/course/3000/workbook/lesson1.copy lesson1
```

```
JULIE 59 > ls
```

```
lesson1 print print.c xyz
```

NOTE: The above move happened because the file permissions allowed it. If this file had been a protected file, then a message would be displayed informing you that the move could not take place.

Try moving the system file called *inittab*.

```
JULIE 60 > mv /usr/inittab inittab
```

```
mv: cannot access /usr/inittab
```

Lets now read the new file we just moved.

```
JULIE 61 > more lesson1
```

```
.h1 "LESSON 1: COURSE INTRODUCTION"
```

Upon completion of this lesson, the student will be able to describe to his/her own satisfaction, the following:

```
.b1
```

What will be taught during this course.

.b1

How the course will be presented.

.b1

How he/she is expected to perform.

.pg

Thats nice, lets now look at another command; one that will allow you to compare files.

JULIE 62 > diff lesson1 print.c

```

1c1
< .h1 "LESSON 1: COURSE INTRODUCTION"
---
> #include <stdio.h>
3,20c3,6
<
<
< Upon completion of this lesson, the student will be able to
< describe to his/her own satisfaction, the following:
<
< .b1
< What will be taught during this course.
<
<
< .b1
< How the course will be presented.
<
<
< .b1
< How he/she is expected to perform.
< .pg
---
> main ()
> {
>     printf("A simple message is printed.");
>     printf("\n");

```

The above output is an example of what you get when two files are not equal. For each line that is different (for both files), those lines are printed with leading characters < or >.

The lines beginning with < are for those lines in the first file that do not compare to the second file, and lines beginning with > are for those lines in the second file that do not compare to the first file.

If the two files are equal, then no *stdout* will occur.

The next command would be useful any time you transfer a file over the Ethernet network and want to be sure it arrived unchanged. The command is the *sum* command and it will calculate a check sum character for a file and give the size of the file in Kbytes.

If you ran this program against a file before transmission, and then ran it on the file after being received, the check sum characters must match to indicate a good transmission. This could become a quick and dirty way to verify transmission lines.

```
JULIE 63 > sum lesson1
```

```
23833 1 lesson1
```

Now, some information about the *find* command before I ask you to execute some of these commands on your own. Find can be a very useful tool for the Field Engineer.

Lets say that you know that a certain file exists somewhere in the tree (system), but, you have forgotten in what directory the file resides in. The following command will find the file for you. (It will take awhile)

```
JULIE 64 > find / -name inittab -print
```

```
find: cannot chdir to /usr/spool/at
```

```
/etc/inittab
```

The file that I was looking for was *inittab*, as defined by the *-name* option. The filesystem that I was looking in was ROOT, as defined by the */* argument. The *-print* option instructs the shell to print the pathname if it finds this file.

The *find* command will search from the directory (the pathname you supply as an argument) down throughout the tree. If you supply it with (*/*) like I did, it will search the complete tree since every thing lives under ROOT. Therefore, if you know the general area of the file, you should supply some other pathname further down in the tree. The reason for doing this, would be to save time.

Notice that the `find` command also indicates that he could not change directory (`chdir`) to the `/usr/spool/at` directory. Lets go find out why.

```
JULIE 65 > cd /usr/spool
```

```
JULIE 66 > ls -l
```

```
total 6 No Per.
drwx----- 3 root      sys      64      May 29 14:47  at
drwxrwxrwx  2 bin        bin      64      May 29 14:47  colord
drwxr-xr-x  7 lp         lp       272     Jun  7 12:14  lp
drwxrwxrwx  2 root      sys      80      Jun  9 09:10  lpd
drwxr-xr-x  3 uucpadm   uucpadm  208     Jun  9 10:35  uucp
drwxrwxrwx  3 uucpadm   uucpadm  48      May 29 14:23  uucppublic
```

There is an answer for almost every condition; the directory (`at`) can only be accessed by the user, and that's `ROOT`. There are no permissions for the classes of `group` and `other`. Since you are logged in as `student`, that puts you into the `other` class. Even if you were a member of the group (`sys` for this directory), you would not be allowed access, since both `group` & `other` are denied permission. (later in the course we will discuss this further)

Lets go back to your home directory.

```
JULIE 67 > cd
```

Since the `find` command can be so important, lets look at another example.

```
JULIE 68 > find ~rich -name lesson2 -print
```

```
/usr/people/rich/course/3000/workbook/lesson2
```

The above command found a file called `lesson2` for you. The shell was instructed to begin looking in the directory `~rich`, which, if you remember, is short hand for `/usr/people/rich`. The file is located at `/usr/people/rich/course/3000/workbook`.

The following example shows you how to find a file when all you have is an *i-node number*. This would be the case when you ran `fsck` against the disk filesystem. If `fsck` detects error conditions, for many of those error conditions, only the *i-node* of the file in error is printed. You would then have to use `find` to tell you where the file was. (more on this later in the course)


```
JULIE 69 > find /usr -inum 7 -print
```

```
/usr/adm/cronlog
```

```
find: cannot chdir to /usr/spool/at
```

The above command instructs the shell to look for *i-node #7* in the */usr* filesystem. The file *cronlog* was found in the directory */usr/adm*.

Now, take some additional time and experiment with only the commands covered to this point. Resume when you are satisfied. **Be sure to leave your home directory looking like it did before you started experimenting; restore (move) any files that you may have moved into your home directory, back to where you moved them from. Your directory should look like the output of the command at JULIE 72, minus the file *your.name*.**

STEP SEVEN: How to copy a file; how to remove a file.

The `copy` command is used to create duplicate copies of files. You can copy files within the current directory, copy files from outside the current directory, you can copy a file from the current directory to any directory in the system. Also, when you copy a file, you can specify any name you want for the new file.

Before you continue with the lab project, execute the following command to insure that you are back in your home directory.

```
JULIE 70 > cd
```

The next command copies the file *print.c* to a new file. Use your actual name (i.e. Robert, Bill, Susan, etc.) for the new filename.

```
JULIE 71 > cp print.c your.name
```

Now, list the directory and see the new file.

```
JULIE 72 > ls
```

```
lesson1 print print.c your.name xyz
```

Now, just for drill, lets compare the two files.

JULIE 73 > diff print.c your.name

Since no *stdout* was return, we are assured that the files are the same.

Lets copy another file in your home directory.

JULIE 74 > cp xyz abc

JULIE 75 > ls

abc lesson1 print print.c your.name xyz

The next command demonstrates how to copy a file from the current directory to another directory in the tree. (We will restore lesson1 to its rightful place in the tree)

(if student) JULIE 76 > cp lesson1 ~rich/course/3000/workbook/lesson1

(if student1) JULIE 76 > cp lesson1 ~rich/course/3000/workbook/lesson1.copy

Lets go see the result of the copy.

JULIE 77 > pushd ~rich/course/3000/workbook

/usr/people/rich/course/3000/workbook ~

JULIE 78 > ls

appendix/	lesson2	print*	student.job*
lab/	lesson2.copy	print.c	student.stuff
lesson1	lesson3	print.c.copy	xyz*
lesson1.copy	lesson3.copy	print.copy*	xyz.copy*

Lets return to your home directory.

JULIE 79 > popd

JULIE 80 > ls

abc lesson1 print print.c your.name xyz

The next command shows you how to remove a file using the `rm` command.

```
JULIE 81 > rm your.name
```

```
JULIE 82 > ls
```

```
abc lesson1 print print.c xyz
```

The remove command can also remove multiple files. Just supply the list of files that you want removed, separated by spaces.

```
JULIE 83 > rm lesson1 abc
```

```
JULIE 84 > ls
```

```
print print.c xyz
```

The next series of commands will show you that you can remove files outside of the current directory and list the contents of other directories from the current directory. **NOTE** that there are two steps for JULIE 85, execute the one that applies to you.

```
(if student) JULIE 85 > rm ~rich/course/3000/workbook/print.c
```

```
(if student1) JULIE 85 > rm ~rich/course/3000/workbook/xyz
```

```
JULIE 86 > ls ~rich/course/3000/workbook
```

Depending on who you are (student or student1), the output will look like the output produced by the `ls` command at JULIE 78, with the exception of the file that you removed (`print.c` or `xyz`).

```
JULIE 87 > pwd
```

```
/usr/people/student
```

The `pwd` command proves that we executed the `rm` and `ls` commands from the current directory. Most UNIX commands can be executed using absolute pathnames.

If you would like to try a few copies and removes on your own, do so, and return to step 8. Make sure you are home when you resume. **Also, make sure only the three files `print`, `print.c`, & `xyz` are left in your home directory when you are finished. DO NOT remove files in the directory `~rich/course/3000/workbook` or critical system files.**

STEP 8: Creating a new directory; creating a file with *cat*; and removing a directory.

Lets print a long listing of your directory.

```
JULIE 88 > ls -l
```

```
total 20
```

```
-rwxrwxrwx 1 student user 17408 May 19 13:19 print
-rw-rw-r-- 1 root sys 87 May 19 12:53 print.c
-rwxrwxr-x 2 root sys 25 May 19 12:53 xyz
```

Your home directory at this point contains only files. We will now create a new directory beneath this directory and call it *temp*. To do this, use the *make directory* command *mkdir*.

```
JULIE 89 > mkdir temp
```

```
JULIE 90 > ls -l
```

```
total 21
```

```
-rwxrwxrwx 1 student user 17408 May 19 13:19 print
-rw-rw-r-- 1 root sys 87 May 19 12:53 print.c
drwxrwxrwx 2 student user 32 Jun 10 09:04 temp
-rwxrwxr-x 2 root sys 25 May 19 12:53 xyz
```

You now have the new directory *temp*. Notice the two permission classes: **user = student** (you own the directory) and **group = user**.

Also, notice the default permissions, they are all on (rwxrwxrwx). Later we will change them using the *chmod* instruction.

Lets now switch to the new directory.

```
JULIE 91 > cd temp
```

```
JULIE 92 > ls
```

Notice that the directory is empty, we are now going to put some files into it.

```
JULIE 93 > mv ~student(1)/print print
```

```
JULIE 94 > ls
```

print

The next command introduces a new concept to you. You are going to use the `cat` command to create a file. Remember, `cat` reads a file with the *stdout* going to the screen. The following command will redirect *stdout* into a new file; we will call the file *redirect*, for lack of a better name.

```
JULIE 95 > cat ~/student(1)/xyz > redirect
```

```
JULIE 96 > ls
```

print redirect

The above command read the file `xyz` from your home directory and the shell redirects (`>`) the *stdout* into the named file.

```
JULIE 97 > more redirect
```

```
echo Hey, this works!
```

```
ls
```

The `more` command verifies that the redirection function did work. The data displayed is a copy of the data that is in the file `xyz`.

Let me now show you another *short hand* method for moving around in the tree.

If you remember, the name of the current directory has a *pseudonym* of (`.`) and the parent directory one of (`..`). To go up one level in a tree, you can use the *pseudonym* as the *name argument* to the `cd` command.

First, just a reminder of where you are.

```
JULIE 98 > pwd
```

```
/usr/people/student/temp
```

Now, lets go the parent directory of this directory.

```
JULIE 99 > cd ..
```

```
JULIE 100 > pwd
```

```
/usr/people/student
```

The next command will create another directory called *bin*.

```
JULIE 101 > mkdir bin
```

```
JULIE 102 > cd bin
```

Lets now place some files into this directory.

```
JULIE 103 > cp ~student(1)/temp/print print
```

```
JULIE 104 > mv ~student(1)/xyz xyz
```

```
JULIE 105 > ls
```

```
print xyz
```

OK, go home again.

```
JULIE 106 > cd ..
```

Print a long listing of your home directory.

```
JULIE 107 > ls -l
```

```
total 5  
drwxrwxrwx 2 student user 64 Jun 10 09:11 bin  
-rw-rw-r-- 1 root sys 87 May 19 12:53 print.c  
drwxrwxrwx 2 student user 64 Jun 10 09:08 temp
```

You can now see the result of executing the above commands: You now have two directories, and the files *xyz* & *print* have been moved into the new directories.

The next command shows you how to remove a directory.

JULIE 108 > rmdir temp

rmdir: temp not empty

The above message is informing you that the shell will not remove the directory because it is not empty. Lets go empty it.

JULIE 109 > cd temp

JULIE 110 > ls

print redirect

JULIE 111 > rm print redirect

JULIE 112 > cd ..

JULIE 113 > rmdir temp

JULIE 114 > ls -l

```
total 5
drwxrwxrwx 2 student user 64 Jun 10 09:11 bin
-rw-rw-r-- 1 root sys 87 May 19 12:53 print.c
```

One more example, then you can experiment on your own.

The following command will list the contents of the tree from the current directory down: Files and directories are both listed, with directory names proceeded by a *slash*.

JULIE 115 > ls -R

bin print.c

./bin:

print xyz

This is a great command to execute from the ROOT directory, redirecting the *stdout* into a file, and then printing the file. You would then have a complete listing of the system. Thats how the listing in section three of the workbook, pages entitled *THE UNIX TREE: DIRECTORY*

LISTING was produced. The filenames were removed; leaving only the directory listing.

Now, if you want, go practice what you have learned. Remember, when you return to the lab project, be in your home directory. Also, remove any files you created, leave the tree as shown by the *stdout* of the previous command, (*JULIE 115*).

Stop Here

STEP 9: Changing file ownerships; file permissions; setting account password; set and alias commands.

Lets look a little closer at the file permissions.

```
JULIE 116 > ls -F
```

```
bin/      print.c
```

```
JULIE 117 > cd bin
```

```
JULIE 118 > ls -l
```

```
total 18
```

```
-rwxrwxrwx 1 student user 17408 Jun 10 09:10 print
-rwxrwx r-x 1 root   sys   25    Jun 10 14:38 xyz
```

The file *xyz* belongs to ROOT. The permissions for the class of *other* (which is you as far as this file is concerned) are set to the value of 5. Remember, each class has a three bit code that is decoded as follows: The *4-weight bit* establishes the *readability* of the file (*r*); the *2-weight bit* establishes if the file can be modified (*w*); and the *1-weight bit* defines if the file is an executable file (*x*). A value of 5 in binary = **1 0 1**, therefore, the *r - x*, as seen in the field that defines *other*.

The file *xyz* can be *read* or *executed* by you, lets see what happens when you try to modify it.

We will attempt to remove the file; this would be a modification.

```
JULIE 119 > rm xyz
```

```
xyz: 775 mode
```

The shell is telling you that you do not have the right to remove this file.

Lets try to change the permissions for the class of *other*.

To change file or directory permissions you must own the file or directory. The command used to change the permissions is called change mode (*chmod*).

Since you do not own the file *xyz*, changing the permissions should not be allowed....lets see.

```
JULIE 120 > chmod 777 xyz
```

chmod: can't change xyz

If you attempted to change the ownership of the file, that also would fail.

The responsibility for changing file permissions usually belongs to the *system administrator*.

Remember, this does not apply to files *owned* by you.

In order to change the permissions of the file *xyz*, you would need to be the *super-user*.

Lets become the *super-user*.

```
JULIE 121 > su
```

Password:

At this point you must enter the password. The password for our training systems is: **password**.

Enter the password. (the characters are not echoed to the screen as you enter the password)

If the password is accepted, then the *super-user* prompt will be displayed.

```
i.e. JULIE 1 #
```

Are you now the *super-user*?....Lets see.

```
JULIE 1 # whoami
```

root

when you entered super-user, where were you placed?....Lets see.

```
JULIE 2 # pwd
```

```
/usr/people/student/bin
```

So, you are in the same directory that you were in before you became the super-user.

Lets now try to change the permissions and ownership of the file *xyz*.

```
JULIE 3 # chmod 777 xyz
```

That appeared to work....lets look.

```
JULIE 4 # ls -l
```

```
total 18
```

```
-rwxrwxrwx 1 student user 17408 Jun 10 09:10 print  
-rwxrwxrwx 1 root sys 25 Jun 10 14:38 xyz
```

As you can see, the permissions were changed for the class of *other*.

You could now go back to *student(1)* and remove the file, but while we are here, lets also change the ownership of the file.

This is done with the command *chown*.

```
JULIE 5 # chown student(1) xyz
```

Lets again see if the change occurred.

```
JULIE 6 # ls -l
```

total 18

```
-rwxrwxrwx 1 student user 17408 Jun 10 09:10 print
-rwxrwxrwx 1 student sys 25 Jun 10 14:38 xyz
```

Lets not stop now, next we will change the *group*.

```
JULIE 7 # chgrp user xyz
```

```
JULIE 8 # ls -l
```

total 18

```
-rwxrwxrwx 1 student user 17408 Jun 10 09:10 print
-rwxrwxrwx 1 student user 25 Jun 10 14:38 xyz
```

Now, lets return to student(1) using the *exit* command.

```
JULIE 9 # exit
```

```
JULIE 122 >
```

Notice that the prompt again indicates that you are in multi-user mode.

Now, try removing the file *xyz*.

```
JULIE 122 > rm xyz
```

```
JULIE 123 > ls
```

print

Now lets do something that most of you have probably been wanting to do; change your account password.

At this time your account does not have a password, so you are now going to establish one, using the command *passwd*.

```
JULIE 124 > passwd
```

Changing password for student

New password: (enter a password and write it down)

*can edit
This out*

Re-enter new password:

Now to check the new password, I want you to login into your account.

NOTE:

You do not have to logout to again login. Execute the login command.

```
JULIE 125 > login student(1)
```

Password: (enter you new password)

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

TERM = (v50am)

Depress return, and you are back in your account.

Notice that the prompt is again = 1.

```
i.e. JULIE 1 >
```

Next I will discuss the *.login* & *.cshrc* files. These files are accessed by the login program when you login. The commands contained inside these files are executed. What this does is set up the *environmentals* for your shell. Also, any special commands that you want executed prior to beginning your session with UNIX are executed. (i.e. starting the window manager, setting up aliases, etc.)

First, lets print your established *environmentals*, using the command *env*.

```
JULIE 1 > env
```

```
HOME=/usr/people/student
```

```
PATH=./usr/people/student/bin:/usr/local/bin:/usr/uch/bin:/usr/bin
```

```
LOGNAME=student
SHELL=/bin/csh
MAIL=/usr/mail/student
TZ=PST8PDT
USER=student
TERM=v50am
```

We will discuss what we are looking at in a moment, but, first I want you to list the contents of your home directory, specifying that all the files are listed (`ls -al`), then display the contents of the `.login` & `.cshrc` files.

JULIE 2 > `ls -al`

```
total 8
drwxr-xr-x  2 student user  176 Jun  7 16:21 .
drwxr-xr-x 10 bin     bin  160 May 29 14:23 ..
-rw-r--r--  1 root   sys  200 May 19 13:38 .cshrc
-rw-r--r--  1 root   sys  180 May 19 12:02 .cshrcDEFAULT
-rw-r--r--  1 root   sys  209 May 19 07:50 .login
-rw-r--r--  1 student user  209 May 19 11:39 .loginDEFAULT
-drwxrwxrwx  2 student user   80 Jun 10 16:44 bin
-rw-rw-r--  1 root   sys   87 May 19 12:53 print.c
```

set env var JULIE 3 > `more .login`

```
setenv SHELL /bin/csh prevents you from using ctrl d to logout
set ignoreeof noglob
set tmp=('tset -S -Q') Set Type of Term you using for port 112
setenv TERM $tmp[1]
unset tmp noglob
stty erase "H" kill "U" intr "C" echoe
set path = ( . /bin /usr/local/bin /usr/uch /bin /usr/bin)
```

JULIE 4 > `more .cshrc`

```
setenv TZ 'cat /etc/TZ' list last 20 commands
set history = 20
if ( $?prompt ) then
    if ( -o /bin/su ) then
        set prompt="hostname' #'
    else
```

TO Set Prompts can change

```

                set prompt="'hostname' > "
                endif
endif
alias m more
alias ll 'ls -al'

```

OK, what's it all mean?...The following discusses each environmental:

- **HOME = /usr/people/student**
This is set by the system.

- **PATH = . /usr/people/student/bin /usr/local/bin /usr/uch /bin /usr/bin**
This is set by the (set path) instruction located in *.login*

This is the *search-rule*. The shell (csh) will look in all the specified directories, looking for a match of a command you enter. It begins the search with the directory listed first, (.), and continues searching *left to right*.

The *search-rule* remains in effect for the life of the shell or until you change it using the set command. (we will do this later)

- **LOGNAME = student**
This is established by the entry in the */etc/passwd* file.

- **SHELL = /bin/csh**
This is also established by a statement contained in the */etc/passwd* file. This defines what shell the user wants.

- **MAIL = /usr/mail/student**
This is set by the system at login. It establishes an electronic mail box for the user. If the user receives mail, then a message is displayed on his/her screen that mail exists in the mail box. The user could then use the *mail* commands to read, save, forward, or discard the individual pieces of mail.

- **TZ = PST8PDT**
This is the *time-zone* for the clock. This is set by an instruction in the *.cshrc* file.

- **USER = student**
This is set by the system.

- **TERM = v50am**

This is set as a result of two commands in the file `.login`. The first command `set tmp=(tset -S -Q)` establishes a variable named `tmp`, with the value of the variable being equal to the output of the `tset` command. This process will determine the type of terminal that is attached.

The contents of the variable `tmp` is then passed to the `(setenv TERM)` command, thus, setting the environmental `TERM` to the correct terminal type.

Other commands in the `.login` file do the following:

- Define the special keyboard control characters. This is done by the `stty` command. You have already used the `ctrl-c` key sequence to interrupt the scrolling process started by reading an executable file, using the `cat` command.
- Sets others variables: `ignoreeof` and `noglob`. (see `cs(1)` in Vol IA for description)

Commands in the `.cshrc` file establish the size of the *history* memory stack; what the *prompts* will look like for your account (multi and super-user prompts); alias commands are established in this file, etc.

The two files `.login` & `.cshrc` could be combined into one file since they are both accessed when your shell is started. The user can place all kinds of special commands and set additional variables from these files.

NOTE:

The information given about these files is for your general knowledge. For the most part, you should not have to worry about the contents of these files, just be aware why they exist.

Lets now play with the *PATH* environmental.

JULIE 5 > print

A simple message is printed.

The above command lives in your *bin* directory, remember, you created the directory and moved the command into the directory.

Since the *search-rule* is set to look in */usr/people/student/bin*, the shell finds the command and executes it.

We will now change the search rule so that shell will not find the command.

```
JULIE 6 > set path = (/usr /usr/local/bin /bin /usr/bin)
```

Lets now look at the environmentals.

```
JULIE 7 > env
```

```
HOME=/usr/people/student  
PATH=/usr:/usr/local/bin:/bin:/usr/bin  
LOGNAME=student  
SHELL=/bin/csh  
MAIL=/usr/mail/student  
TZ=PST8PDT  
USER=student  
TERM=v50am
```

```
JULIE 8 > print
```

print: Command not found.

Now, one more point.

The effects of either file (*.login* or *.cshrc*) can again be set by using the source command. This command will access the specified file and execute the commands in the file as the login program would. If you edit the *.login* or *.cshrc* files, adding additional variables, alias commands, or the like, you need not logout and log back in to feel the effects of the file. Execute the following command.

```
JULIE 9 > source .login
```

```
JULIE 10 > print
```

A simple message is printed

This ends this section of the lab project. Feel free to try your hand at some of the commands covered. When you are satisfied, return to your home directory and continue with the project. Also, leave your account tree looking the way it does now, before you start experimenting. Make copies of any file you intend to remove, before you remove it.

STEP 10: Network commands.

Three commands exist for communications over the network. These commands are: `r``cp` (`xcp`), `r``sh` (`xx`), and `r``login` (`xlogin`).

NOTE: All network commands preceded with the letter 'r' are TCP related while those preceded with 'x' (in parentheses) are XNS related.

- **r**`cp` (`xcp`)
This command allows the user to copy files between machines. File permissions control this process just as they do within the local workstation.

- **r**`login` (`xlogin`)
This command allows a user to *remotely login* to another workstation or host computer.

You must already have an account established at the other machine or know an account *user ID* and have permission to enter that account.

- **r**`sh` (`xx`)
This command allows the user to remotely execute commands on another machine. It assumes that the user has an existing account on the remote machine, with the same account ID of the machine the user is currently on (the local system). If that account does not exist on the remote machine, then limited access (to the other machine) is allowed through the *guest* account.

NOTE:

The first network command we will try, will be the *remote execution* command, `xx`.

JULIE 11 > `xx elvin date`

Wed Jun 11 14:33:13 PDT 1986

JULIE 12 > `xx elvin whoami`

guest

The remote machine you are communicating with is called *elvin*. Since your account on this workstation is called *student*, the security system allows you access to the *guest* account on *elvin* for the execution of the above two commands. The reason for this is because there is no account called *student* on *elvin*. The reason the security system does not allow you full access should be obvious.

Lets now switch to your *bin* directory.

```
JULIE 13 > cd bin
```

```
JULIE 14 > ls
```

print

The only file in *bin* is *print*, unless, while you were experimenting with the various commands you have learned, you copied or moved some other files into this directory and forgot to remove them when you were finished.

You are now going to copy a file (called *student.stuff*) from *elvin* into this directory. You will also change the name of the file to *xcp.copy*.

```
JULIE 15 > xcp elvin:~rich/student/student.stuff xcp.copy
```

```
JULIE 16 > ls
```

print *xcp.copy*

```
JULIE 17 > more xcp.copy
```

This file exists for only one purpose; allow students performing the UNIX SURVIVAL LAB the oportunity to copy a file over the network into their student accounts.

NOTE:

Opportunity is purposely misspelled, so that later, the student will be able to find it with the spell command.

The remote copy worked because the permissions for the directory (~rich/student), that you accessed, are set to 777 for all classes. The copy would fail if the directory was protected with a different code.

I am now going to have you transmit the *print* file to *elvin*, but, before I do, I want you to perform a *sum check* on the file.

```
JULIE 18 > sum print
```

```
63486 17 print
```

Now, copy the file to *elvin*, using your real name as the new filename for *elvin*.

```
JULIE 19 > xcp print elvin:~rich/student/your.name
```

Now, to prove that you really copied the file, I want you to remove the file *print* from the current directory.

```
JULIE 20 > rm print
```

```
JULIE 21 > ls
```

```
xcp.copy
```

As you can see, it is gone. We will now restore the file by again using the *xcp* command.

```
JULIE 22 > xcp elvin:~rich/student/your.name print
```

```
JULIE 23 > ls
```

```
print      xcp.copy
```

Remember the sum check you performed on *print* before transmitting it to *elvin*? Use the *sum* command again and check if the file was received OK.

```
JULIE 24 > sum print
```

```
63486 17 print
```

The command *xlogin* will now be demonstrated. Before you execute the command, you must obtain a valid workstation name from your instructor. He/she will determine what workstation in the classroom is unused, and it's name will be supplied to you.

JULIE 25 > xlogin ^{DOC} <instructor supplied name>

<workstation name> login:

You can now login to the workstation. Login as ROOT.

<workstation name> login: root

*Silicon Graphics Inc.
IRIS Workstation
TERM = (rwsiris)*

The workstation is asking what type of terminal you are on, this is nothing new to you, see page 7 of this lab project.

This workstation has a different welcome message, but, it still needs to know what type of terminal you are using.

It is asking you if the terminal is a *remote IRIS workstation*, this is what *rwsiris* means. If you were logging in from the primary workstation terminal, then, you would be a *rwsiris* and you would depress return. Since you are on the *Visual 50 or 60*, enter *v50am*.

TERM = (rwsiris) v50am

#

whoami

root

You can now look at a few things in this workstation. **PLEASE DO NOT CHANGE ANY FILES.** You will find this workstation just like your's, but, there are no special directories like the ones you have been performing this lab project from. Take a couple of minutes for looking, then logout.

logout

JULIE 26 >

STEP 11: Running jobs; collecting job status; killing jobs.

This section of the lab project will show you how to look at status that the system collects and maintains for the various jobs (commands, programs, etc.) that are currently running on the workstation.

You will be asked to execute a command that is an executable shellsript file. It is only used to simulate an actual program, the functions that it is performing are only make-believe tasks. The commands used within the shellsript are actual UNIX commands and I will take some time to explain what they are doing in the interest of giving you additional insight as to how specific commands work.

While the command is running we can look at it's status, then, using the *PID* (process ID number) obtained from the status, kill the job. This is handy to know if your system hangs and the hangup is due to a specific job. Knowing the *PID*, you may be able to free the system by killing the offending job.

Before you can run the job, you must first copy it from the *student* directory set up on elvin.

You should still be in your */bin* directory.

```
JULIE 26 > pwd
```

```
/usr/people/student/bin
```

The next command will copy over the make-believe job from elvin.

```
JULIE 27 > xcp elvin:~rich/student/student.job student.job
```

```
JULIE 28 > ls -F
```

```
print*      student.job*  xcp.copy
```

OK, lets look and see what UNIX commands make up this file.

```
JULIE 29 > more student.job
```

```
echo STUDENT JOB has been started and is running
```

```
find /usr -name student -print > temp.1
```

```
echo The first FIND command has finished
```

```
find / -name phony -print >> temp.1
```

LAB 3.1: Basic UNIX Commands

```

echo The second FIND command has finished
find /usr -name phony -print >> temp.1
echo The THIRD FIND command has finished
ls -R /usr >> temp.1
echo The LS command has finished
echo end of listing >> temp.1
grep -n print.c temp.1 > temp.2
grep -n 'end of listing' temp.1 >> temp.2
echo The GREP commands have finished
cat temp.1 temp.2 > temp.3
echo The job has completed, but, a sleep command is waiting 30 seconds before
echo executing the last command
sleep 30
echo This job is finally finished!

```

The following describes each command.

1. **echo STUDENT JOB has been started and is running**
This command will echo the message on your terminal screen.
2. **find /usr -name student -print > temp.1**
This command will search the */usr* filesystem looking for a file called *student*. If it finds the file, then the absolute pathname of the file will be printed, but, the redirection operator (**>**) will redirect the standard output (*stdout*) into a file called *temp.1*.
3. **echo The first FIND command has finished**
When the *find* command finishes, this command will echo the message on your terminal screen.
4. **find / -name phony -print >> temp.1**
This *find* will look for the file named *phony* in the ROOT directory. If it finds it, then it's pathname would be appended (**>>**) to *temp.1*. I put this command in just to keep the CPU busy.
5. **echo The second FIND command has finished**
Prints the message on your screen.

6. **find /usr -name phony -print >> temp.1**
This find will search the */usr* filesystem for phony. Keeps the CPU busy.

7. **echo The third FIND command has finished**
Prints the message on your screen.

8. **ls -R /usr >> temp.1**
This really keeps the CPU busy. The **ls -R** command is instructing the shell to list the complete */usr* tree. The *stdout* is directed into and appended to *temp.1* (this will be a big file when finished).

9. **echo The LS command is finished**
Prints the message on your screen.

10. **echo end of listing >> temp.1**
The message is redirected and appended to file *temp.1*. (we will later search the file for this character string).

11. **grep -n print.c temp.1 > temp.2**
The **grep** is a useful tool for searching files looking for a match of a *character string* that you supply as an argument to the command.

There are many options available to use as part of this command.

The option you are using (**-n**), requests that the *line number* (in the file) be printed for all lines that contain the character string.

The character string you are using is *print.c*, the file being search is *temp.1*, and any *stdout* is being redirected into a file called *temp.2*.

12. **grep -n 'end of listing' temp.1 >> temp.2**
This **grep** command is looking for the string *end of listing* in the file *temp.1*. The *stdout* is redirected and appended to the file, *temp.2*.

We know that this command should find this string, because, we put it into the file with the **echo** command executed earlier. This will tell us the line number where the string is located.

13. **echo The GREP commands have finished**
Prints the message on your screen.

14. **cat temp.1 temp.2 > temp.3**
The `cat` will concatenate (connect) the two files `temp.1` & `temp.2` into one stream. The `stdout` is redirected into a third file called `temp.3`. This is the real function of `cat`. If you remember, earlier in the lab project you used `cat` to read a file, and I told you reading long files would be difficult because the data stream is scrolled past the screen without stopping, therefore, `cat` is not suitable for reading long files.

15. **echo The job has completed, but, a sleep command is waiting 30 seconds before executing the last command.**

This message is actually printed using two `echo` commands because I could not fit the complete message on one line and have it look good.

16. **sleep 30**
This command will kill 30 seconds of time (pause). I put it here just for demonstration.
17. **echo This job is finally finished!**

OK, we will run the job, but before we start it, lets look at the current workstation process status.

To do this you will use the `ps` command. It has many options, but, the ones I am going to ask you to use, produce the most useful output (`a`, `u`, and `x` options).

JULIE 30 > `ps -aux`

USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	TIME	COMMAND
root	445	47.7	3.5	172	116	d1	R	0:01	ps -aux
student	79	0.0	2.1	108	68	d1	I	0:04	-csh (csh)
root	57	0.0	0.6	12	12	w0	S	0:00	/etc/update
root	60	0.0	1.1	44	32	w0	I	0:00	/etc/cron
root	78	0.0	1.2	48	32	w0	I	0:00	/etc/getty console co_9600
root	2	0.0	0.2	640	0	?	D	0:04	pagedaemon
root	1	0.0	1.8	64	56	?	I	0:04	INIT 2
root	0	0.0	0.1	0	0	?	D	0:04	swapper
daemon	72	0.0	1.5	60	44	w0	S	0:00	/usr/lib/lpd
root	69	0.0	1.5	60	44	n31	I<	0:00	/etc/xnsd /usr/local/boot

```

root    80  0.0  1.2  48  32  d2  1  0:00  /etc/getty ttyd2 dx_9600
lp      75  0.0  2.1  76  68  ?   1  0:00  /usr/lib/lpsched

```

Three fields hold information that is useful to us: *USER*, *PID*, and *COMMAND*.

- **USER**

This tells you who started the job.

- **PID**

This is the process ID number assigned (by the system) to the job when it was started. This is what you would use to *kill* a job.

- **COMMAND**

This is the name of the job that is running.

The following gives brief descriptions for the jobs that are currently running:

- **ps -aux**

This is the command that you executed to get the status. This proves that UNIX commands run as jobs (processes).

- **-csh (csh)**

This is your shell program that is started when you login and it runs until you logout.

NOTE:

Killing this job will produce the same effect as logout. You may want to try it later, but, **NOT NOW.**

- **/etc/update**

This is one of the *daemons* that the INIT process started when you entered *multi-user* mode. It is always running. It will update the *super-block* every 30 seconds. This insures that the system is fairly up to date in case of a crash.

- **/etc/cron**

This is the *clock daemon*. It executes commands at specified dates and times according to the instructions in the file */usr/lib/crontab*. We talked about this earlier in the lesson.

One of the programs that this daemon runs is *atrun*. This is the program that syncs the drive every 10 minutes.

- **/etc/getty console co_9600**
This is a *getty* process waiting for someone to log onto the workstation, using the primary terminal (console). Once someone logs in, a process entry (in this table) would exist that looks like the entry detailing your shell (*-csh (csh)*).
- **pagedaemon**
This process manages paging (of the *swap-area*) on the system disk.
- **/usr/lib/lpd**
This daemon is the *line printer spooler*. It causes named files to be queued for printing on a line printer.
- **INIT 2**
This is the *process control initialization* program. It is started when the system is booted and never dies. Its primary role is to create processes from a script stored in the file */etc/inittab*. Some of the processes are the daemons and the gettys, of which, one will be started for each line a user may log in on.
- **/etc/xnsd /usr/local/boot**
Xnsd is a daemon that is the server for the XNS network utilities. This server listens for connections and will start up the appropriate programs.
- **swapper**
This is the *process handler manager*. It is the program that allows multi users to exist on the workstation at the same time.
- **/etc/getty ttyd2 dx_9600**
This is another *getty* process waiting for someone to login to the terminal on port 2 (ttyd2). When I created this document, no other user was on my workstation, but, if you are sharing the classroom workstation with another student, then you will not see this entry, but, one that looks like the entry detailing your shell (*-csh(csh)*).
- **/usr/lib/lpsched**
This daemon will schedule print jobs.

NOTE:

The output from your `ps -aux` command will have different process ID numbers. Also, if you are sharing the workstation you will see additional processes that were started by your partner.

The order that your processes are listed may also be different.

Now, lets run our simulation program. You will instruct the shell to run the program in the *background* using the `&` operator after the command. This will insure that your keyboard will be available while the program is running.

The first time you run the program, just sit and watch the output as it appears on the screen. We will run it again, look at the process status, then kill it using the `kill` command.

JULIE 31 > `student.job &`

446

JULIE 33 > *STUDENT JOB has been started and is running*

find: cannot chdir to /usr/spool/at

The first FIND command has finished

find: cannot chdir to /usr/spool/at

The second FIND command has finished

find: cannot chdir to /usr/spool/at

The third FIND command has finished

The LS command has finished

The GREP commands have finished

The job has completed, but, a sleep command is waiting 30 seconds before executing the last command

This job is finally finished!

NOTE:

The number that first appeared at your screen *446*, is the *process ID number*. Your number will be different, also, you must depress return to get your prompt back.

Lets now look at the results of executing the program.

JULIE 32 > `ls`

print

temp.1

temp.3

student.job

temp.2

xcp.copy

You can now see the three files that were created by *student.job*. Lets look at *temp.1* & *temp.2*.

JULIE 33 > more temp.1

```
/usr/people/student
adm
bin
diag
dict
games
include
.
.
.
.
.
.
.
.
.
end of listing
```

In the interest of saving space, only the first few lines and the last line of *temp.1* are reproduced by this lab project. You can interrupt the more command, using (^c) when you tire of reading the file.

Lets now look at *temp.2*.

JULIE 34 > more temp.2

```
999:print.c
1004:print.c
3152:print.c
3190:end of listing
```

The contents of this file lists three line numbers where the string *print.c* was detected, and one line number where the string *end of listing* was found.

I am now going to show you something that is very useful, but, can be very destructful if used

wrong, and that is, the use of the *meta-characters*. When in doubt about the use of these characters, **DO NOT USE THEM**.

Lets list the contents of your *bin* directory.

JULIE 35 > ls

```
print          temp.1         temp.3
student.job    temp.2         xcp.copy
```

The two *meta-characters* that I will show you are: ? and *. These characters, when used within command arguments, have very special meaning to the shell when executing that command.

- ?

This character is called the *wild-card*. When used in the command argument it becomes a wild-card for the position that it occupies in the string.

i.e. rm te?

This command would remove any files that began *te* and had any character as the *third* character. (tef, te3, tem, etc...)

- *

This is the *multiple character wild-card*. It will match any string (be careful with this one).

i.e. rm temp*

This command will remove all files beginning with *temp* and any number and type of characters following temp.

The next example shows one use of the (*) meta-character.

JULIE 36 > rm te*

JULIE 37 > ls

```
print          student.job    xcp.copy
```

Get the picture? You could remove the complete contents of a directory with the following command: **rm ***. Don't say I did not warn you.

The meta operators are very useful to programmers. When a programmer wants to search for or find strings of data that are similar, these characters are needed, or at least make the job easier. It is because of this fact, that I even mention these characters; you may see them in arguments of commands of the numerous files you have to deal with while maintaining the IRIS workstation.

OK, on with the show.

We will again run the simulation program; this time we will kill the process using the **kill** before it can complete. Before you use the **kill** command, you must know the *PID*.

JULIE 38 > **student.job &**

479

JULIE 39 > *STUDENT JOB has been started and is running*

NOTE:

If, while you are attempting to enter a command, and screen output appears, interfering with your input, just depress return to get your prompt back.

JULIE 39 > **ps -aux**

USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	TIME	COMMAND
root	445	47.7	3.5	172	116	d1	R	0:01	ps -aux
student	79	0.0	2.1	108	68	d1	I	0:04	-csh (csh)
student	481	30.8	1.7	80	52	d1	D	0:03	find /usr -name student -pri
student	479	0.5	1.4	52	40	d1	S	0:00	/bin/sh student.job
root	57	0.0	0.6	12	12	w0	S	0:00	/etc/update
root	60	0.0	1.1	44	32	w0	I	0:00	/etc/cron
root	78	0.0	1.2	48	32	w0	I	0:00	/etc/getty console co_9600
root	2	0.0	0.2	640	0	?	D	0:04	pagedaemon
root	1	0.0	1.8	64	56	?	I	0:04	INIT 2
root	0	0.0	0.1	0	0	?	D	0:04	swapper
daemon	72	0.0	1.5	60	44	w0	S	0:00	/usr/lib/lpd
root	69	0.0	1.5	60	44	n31	I<	0:00	/etc/xnsd /usr/local/boor
root	80	0.0	1.2	48	32	d2	I	0:00	/etc/getty ttyd2 dx_96
lp	75	0.0	2.1	76	68	?	I	0:00	/usr/lib/lpsched

JULIE 40 > find: cannot chdir to /usr/spool/at
The first FIND command has finished

You can see from the table that *student.job* is running along with one of the *find* commands. Your output will reflect the point in time that you execute the *ps -aux* in relation to the starting of the program. If you waited a long time, you may see one of the other commands.

The next command will kill the program. My *PID #* was 479. You must use the *PID #* shown on your screen.

```
JULIE 40 > kill -9 479
```

The (-9) option is an absolute kill.

Execute the *ps -aux* command again to verify that you killed the process. The lab project will not show the output of the command.

```
JULIE 41 > ps -aux
```

Lets again look at your *bin* directory.

```
JULIE 42 > ls
```

```
print student.job temp.1 xcp.copy
```

The next command shows one example of using the *wild-card* meta-characters. I am removing *temp.1*, it was created by *student.job* before I could kill it.

```
JULIE 43 > rm ??m*
```

```
JULIE 44 > ls
```

```
print student.job xcp.copy
```

Now, run the job in *foreground* and see what happens to your keyboard.

JULIE 45 > student.job

User break or ^c to get your keyboard back.

Well, one more command and this lab project is over as far as the instructional portion is concerned.

Earlier you used the `xcp` command to copy the file *student.stuff*. It has a misspelled word in it, so lets use the command `spell` to find it for us. Remember, you changed the name to *xcp.copy* as you copied it over.

JULIE 46 > spell xc*

oportunity

Notice that I had you use the meta-character (*) to replace the characters after `xc` in the filename.

The `spell` command shows us that *oportunity* is misspelled. If this was a letter or some other type of file that you had created, then you would have to *edit* the file to correct the mistake. That will lead us into the next lab project: **The Vi Editor.**

You are now free to experiment with any or all the commands presented to you during this lab project. When you are finished, complete the questionnaire in section two, and hand it into your instructor.

When you are finished with the lab project perform the following: Enter *super-user*, execute `reboot`, and **power off** your workstation.

SECTION 2: Review

Student Name:

Please complete the review and hand it into your instructor.

1. What is the name of the file that the PROM monitor looks for on the boot device, to be used for booting UNIX, if no filename is supplied with the boot command **b**?

UMUNIX / deFault Boot

2. What mode does UNIX boot into?

Single user

3. When is the program **fsck** run?

boot when you type → multi or FscK

4. What does a **getty process** do?

look at files set for Term definitions

gettydefs also Terms to login

pg

5. If the terminal you login at is not set for 9600 baud, garbage will appear when the login message is received. What must the user do to instruct the **getty process** to try another baud rate?

Break Key

6. What file holds the terminal configuration data for all supported terminals? *P P9.*

gettydef / *etc / Termcap*

7. What command would you use to determine who else is on the workstation with you?

who

8. Show the command used to produce a long listing of a directory.

LS -L / *LS -AL*

9. What is the pseudonym for the parent directory?

..

10. What are the names of the files accessed by the start up process to initialize and set up parameters for a shell?

passwd / *login* / *bin/csh* / *bin/sh*

11. What directory in UNIX holds the majority of the configuration files?

etc / */boot*

12. What command is best suited for reading files?

more

13. Show the syntax for copying a file from a host system to your workstation.

xcp Doc: / Rich / student / File a File

14. Show the command for finding a file named *file1*, known to live somewhere in the *usr* filesystem. You also want to list its complete pathname.

Find ^{-Name} *usr* *file1* -PRINT

15. A file called *student* has the following permissions set for it: *rwrx-rx-x*. What does this mean?

-/owner all / Group = Read + Execute / Other = Read + Execute

16. Show the command for setting the permissions for the file in step 15 to allow only reading by the classes of *group* and *other*.

chmod *744* *student*

17. What command is best for listing process status?

ps -aux

18. What environmental defines the shell *search-rule*?

dk Path from .login File

19. What file instructs the start-up program what shell the user wants to use?

dk *etc/passwd*

20. What is the command used to search for character strings in a file?

grep *ps2*

Contents

LAB 3.2: Vi Editor	1
SECTION 1: Instructional Procedure	2
I. Vi Editor	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT	2
V. PROCEDURE	3
SECTION 2: Vi Editor Reference Guide	25
SECTION 3: Review	30

LAB 3.2: Vi Editor

This lab project contains three sections:

- Section 1:** This section is used to teach you how to use the commands of the **vi editor** for creating and modifying (edit) files. It allows you to practice many of the commands and see what type of output occurs as a result of executing the various commands.

- Section 2:** This section is a quick reference guide for most of the **vi** commands. It shows each function as an illustration and lists the required key strokes to execute the function. Once you are somewhat familiar with the **vi** commands, this guide should prove useful to you.

- Section 3:** This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. Vi Editor

Vi is a general purpose program used to prepare text files. It allows the user to enter, change, and correct text. Once a file is created and becomes part of the operating system, the need to change the file is remote. Although most files will remain unchanged, occasions do occur that necessitates the modification of a file, i.e. when additional equipment is added to the already existing configuration, several files in the directory */etc* must be modified to reflect the additional hardware.

Vi is an interactive text editor designed to be used with a crt terminal. It produces a *window* into the file you are editing. This window lets you see about 40 lines of text of the file at a time, and you can move the window up and down through the file. You can move to any part of any line on the screen and make changes there.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly execute any of the *vi commands* presented by this lab project.
- B. Demonstrate that you can create a new data file in your student directory and make changes to the file.

III. PURPOSE

Teach the student how to use the *Vi editor commands*. The project seeks to develop a basic skill in each student that will allow the student to modify existing system files. The project allows each student sufficient time to practice using the vi editor.

IV. EQUIPMENT.

A functional IRIS 68020 based workstation.

V. PROCEDURE

Two students will be assigned one IRIS workstation. Each student will perform the lab project from one of the two ASCII terminals attached to *ports two or three*.

Special accounts have been created for the classroom environment to be used by the students. The account names are *student* and *student1*. Both accounts are identical, so choose one and keep that account name throughout the remainder of the course.

It is very important that you follow the lab project precisely and do not wander off in your own direction. You will have ample time for *free-time* later in the course.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

1. **iris >** The prom monitor.
2. **JULIE n >** Student account prompt when in Multi-user mode, where n = command number.
3. **JULIE n #** Student account prompt when in Super-user mode, where n = command number.
4. **#** Single user mode.
5. **JULIE login:** This prompt appears when you go into Multi-user mode from Single-user.

The following text will lead you through the execution of many of the VI COMMANDS. When you are required to enter a command, the text recreates the STANDARD OUTPUT for that command. Just keep reading and following the flow of the lab project.

Also throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **JULIE > vi <filename>**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in normal print.

BEGIN ACTUAL PROCEDURE HERE:**STEP 1: Power on the workstation.**

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985  
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)  
Configuration Switch: 0x0000  
Multibus Window (2mb) at Megabytes 0 and 1.  
Multibus accessible memory (1mb) begins  
at Physical memory page 300,  
at Virtual address 2000000.  
iris>
```

STEP 2: Boot the UNIX System.

```
iris > b
```

```
SGI Extent Filesystem  
Loading: md:0:defaultboot  
Text: 038318 bytes  
Data: 0113d8 bytes  
Bss: 024d7c bytes  
Jumping to load program ~ 20000400
```

```
SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]  
(C) Copyright 1986 - Silicon Graphics Inc.  
real = 4194304
```

```
kmem = 561152
user = 3633152
bufs = 819200 (max=16k)
dsd0 not installed
qic0 not installed
sii0 at mbio 0x07200 ipl 5
si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0
si1 not installed
sf0 floppy (80/2/8) slave 2
siq0 at mbio 0x73fc ipl 5
sq0 (qic02 cartridge tape) slave 0
iph0 not installed
tmt0 not installed
ik0 not installed
nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2
fpa installed
lpen not installed
kernel debugger disabled.
root on si0a
swap on si0b. swplo=0 nswap=64000
```

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

#

STEP 3: Go to multi-user mode.

multi

#

INIT: New run level: 2

Is the date [day month date time time-zone year] correct? (y or n) y

Do you want to check filesystem consistency? (y or n) y

Checking file systems for consistency:

/dev/si0a

File system: root Volume: SGI

*** Phase 1 - Check Blocks and Sizes*

*** Phase 2 - Check Pathnames*

*** Phase 3 - Check Connectivity*

*** Phase 4 - Check Reference Counts*

*** Phase 5 - Check Free List*

nnn files nnnnn K used nnnn K free

/dev/si0f

File system: usr Volume: SGI

*** Phase 1 - Check Blocks and Sizes*

*** Phase 2 - Check Pathnames*

*** Phase 3 - Check Connectivity*

*** Phase 4 - Check Reference Counts*

*** Phase 5 - Check Free List*

nnn files nnnnn K used nnnn K free

Mounting: /usr

Preserved editor files

Cleared /tmp

Resetting locks and logs

Hostname: JULIE

Daemons:

update

cron

xnsd

lpd

lpsched

Daemons started

JULIE login:

Each student should now move onto his/her own ASCII terminal to complete the lab project.

STEP 4: Starting vi.

Although **vi** is a very sophisticated editor with an enormous number of commands, the basic structure of **vi** is very simple. There are two modes of operation, *Command Mode* and *Text Input Mode*.

- **Command Mode**

When you enter **vi** you are placed into command mode. If the file already exists, the file contents are copied into a *buffer* for editing. Once in the buffer, the first 40 lines (if there are that many) are displayed at your terminal screen.

If there are not enough lines to fill the screen, **vi** fills the screen with the tilde character (`~`) to the bottom of the screen where the filename, the number of lines, and the number of characters in the file are displayed.

If the file does not exist, **vi** opens a buffer, fills the screen with the tilde character, and displays the filename followed by the comment (in brackets []) *New file*.

In both cases, **vi** enters the buffer with the cursor at the top left of the screen.

To edit a file using **vi** you execute the UNIX command **vi** followed by the name of the file.

i.e. **vi filename**

Command mode allows the user to position the cursor to a point in the file where editing will begin. From command mode the user can delete existing text, search for a specific character string within the text, or enter one of the text input modes.

- **Text Input Mode**

This mode is where you actually enter new text into the buffer. Five Input Modes are available for use:

- **a**

Allows user to *append* new text beginning at the point where the cursor was positioned in the file during command mode. You can type as many lines and [return]'s as you wish.

- **i**

Allows user to *insert* new text beginning at the point before the cursor. You can insert as many lines and [return]'s as you wish.

- **o** (this is lower case o)
Allows user to open a new line below the cursor. You can then input as much text as you want.
- **O** (this is upper case O)
Opens a new line above the cursor and allows new text input.
- **R**
Allows the user to *replace* text, starting at the cursor, with any characters typed.

In this lab project, we will limit ourselves to the minimum number of commands that you will need to start using **vi**. Even though these commands are just a few of the total commands in **vi**, they cover the four major features of editing:

1. Cursor positioning.
2. Text insertion.
3. Deletions and changes.
4. Permanently storing the information.

STEP 5: Moving the Cursor.

OK, lets login to your student account.

JULIE login: student or student1

Password:

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

TERM = (v50am) return

JULIE 1 >

First lets move into your */bin* directory.

```
JULIE 1 > cd bin
```

```
JULIE 2 > ls
```

```
print      student.job  xcp.copy
```

This is what should be in your bin directory. You may have additional files that you copied or moved into this directory when your were practicing with UNIX at the end of lab project 3.1.

If so, remove all but the above named files.

There are *five* basic cursor positioning keys: *k*, *j*, *h*, *l*, and *return*.

- **j**
This key positions the cursor *down* one line in the buffer. Movement to the next line will be to the corresponding character position or to the end of the next line.
- **k**
This key positions the cursor *up* one line in the buffer. Movement to the proceeding line will be to the corresponding character position or to the end of the proceeding line.
- **l**
This key positions the cursor *right* one character position within the line.
- **h**
This key positions the cursor *left* one character position within the line.
- **return**
This key positions the cursor *down* one line in the buffer. The movement is straight down the left margin.

NOTE:

If your keyboard has the *arrow* keys, then you could use them instead of the *j*, *k*, *h*, and *l* keys.

We will now enter `vi`, giving *new.file* as the name argument.

```
JULIE 3 > vi new.file
```

```
~  
~  
~  
~  
~  
~  
/  
/  
/  
/  
~  
~  
~  
~  
~
```

"new.file" [New file]

You just instructed `vi` to open a buffer called *new.file*. Since it did not exist, tilde (~) characters were displayed.

The cursor will be in the upper left corner of the window, try moving it using the cursor position keys.

OK, so you got a *beep*. Lesson number one: You can not move the cursor around in a buffer that is empty.

We must now exit this buffer. Execute the following `vi` command:

```
shift ZZ
```

```
JULIE > 4
```

Shift : Q!

Now we will edit the file *xcp.copy*. If you remember, you brought the file *student.stuff* over from *elvin* renaming it *xcp.copy*.

JULIE 4 > vi xcp.copy

This file exists for only one purpose; allow students performing the UNIX SURVIVAL LAB the opportunity to copy a file over the network into their student accounts.

NOTE:

Opportunity is purposely misspelled, so that later, the student will be able to find it with the spell command.

-
-

"xcp.copy" 9 lines, 284 characters

Now, take a couple of minutes and practice moving the cursor. When you are satisfied, exit vi using shift ZZ.

shift ZZ

JULIE > 5

STEP 6: Text Input Mode

You are now going to create a new file called *new.command* using the "i" input mode. When vi enters *input mode*, you will get no indication that you are in input mode. You just start entering the data. Enter the data exactly as shown below. Later in the lab project we will use this file.

JULIE 5 > vi new.command

When the tilde characters appear, place vi into *input mode* by depressing the "i" key, and then enter the text.

i

clear <return>

more /usr/people/student(1)/bin/welcome <return>

To exit from *input mode*, depress the esc key. This returns vi to command mode.

esc

If you have any doubt that you are in command mode, depressing esc again will cause a *beep* at

your terminal. This tells you that you are in command mode.

The next step will be saving the contents of the buffer into a file on the disk. There are many ways to instruct vi to write out the buffer, you will use the *shift ZZ* command. This writes the buffer to the named file and quits vi.

shift ZZ

JULIE 6 >

Lets now list the directory and see the new file.

JULIE 6 > ls

new.command *print* *student.job* *xcp.copy*

Lets read the contents of the new file.

JULIE 7 > more new.command

clear

more /usr/people/student/bin/welcome

Now we will create another new file and call it *welcome*. Enter the lines of text that are shown below.

JULIE 8 > vi welcome

When the tilde characters appear, place vi into *input mode* using the "i" key.

i

Welcome OH great one! <return>

I am an IRIS 68020 based workstation. <return>

I enthusiastically await your commands. <return>

Remember, to exit from *input mode* use *esc*. To quit vi and write the buffer, use *shift ZZ*.

esc

shift ZZ

JULIE 9 >

Lets again list your directory.

JULIE 9 > ls

new.command print student.job welcome xcp.copy

JULIE 10 > more welcome

*Welcome OH great one!
I am an IRIS 68020 based workstation.
I enthusiastically await your commands.*

At this point I want you to create your own file and put a few lines of text of what ever comes into your head into the file. If you have trouble, ask your instructor for help. Continue the lab project when you are satisfied that you can create a new file.

STEP 7: Deleting and Changing Text.

Before we continue editing, we are going to create one large file from several of the files that exist in your *bin* directory using the cat command.

JULIE 11 > cat new.command student.job welcome xcp.copy > big.file

JULIE 12 > ls

*big.file print welcome
new.command student.job xcp.copy*

JULIE 13 > more big.file

*clear
more /usr/people/student(1)/bin/welcome*

*echo STUDENT JOB has been started and is running
find /usr -name student -print > temp.1*

```
echo The first FIND command has finished
find / -name phony -print >> temp.1
echo The second FIND command has finished
find /usr -name phony -print >> temp.1
echo The third FIND command has finished
ls -R /usr >> temp.1
echo The LS command has finished
echo end of listing >> temp.1
grep -n print.c temp.1 > temp.2
grep -n 'end of listing' temp.1 >> temp.2
echo The GREP commands have finished
cat temp.1 temp.2 > temp.3
echo The job has completed, but, a sleep command is waiting 30 seconds before
echo executing the last command
sleep 30
echo This job is finally finished!
```

*Welcome OH great one!
I am an IRIS 68020 based workstation.
I enthusiastically await your command.*

*This file exists for only one purpose; allow students performing
the UNIX SURVIVAL LAB the opportunity to copy a file over the network
into their student accounts.*

NOTE:

*Opportunity is purposely misspelled, so that later, the student
will be able to find it with the spell command.*

Now, before I show you how to *delete* or *change* text, I want to show you some additional cursor movement commands. While we execute the various commands, refer to page 26. This page illustrates cursor movement and documents the key stroke or key strokes required to perform the illustrated function. Page 25 gives a legend explaining the various symbols used in the document.

Lets open *big.file* for editing. The lab project does not reproduce the output created at your screen when you enter *big.file* using vi because it will look just like the output of the *more* command, also, we are not going to change anything in the file at this moment.

JULIE 14 > vi big.file

OK, you are now in the editor, the cursor is in the upper left corner; lets move around in the file.

NOTE:

Please do not attempt any deletions or additions on your own, just stick to the flow of the lab project. We will get to the deletions in a few minutes.

Lets do a forward search in the file looking for the string **FIND**. (enter **FIND** in upper case)

/FIND

Notice that the search is *case sensitive*, it skipped over the first *find* in the previous line.

Now depress the **n** key.

n

The forward search is again executed using the last string that you supplied it with.

Using upper case **N** will cause a backward search using the last supplied pattern.

N

Now, take a few minutes and execute as many of the cursor movement commands as you want. Practice the commands, but **DO NOT** use the destructive type commands and do not use the key stroke that takes you to the command line, we will do this later. If you need assistance, call your instructor. When you are satisfied, resume the lab project.

We are now going to delete some of the text in *big.file*. The way I will illustrate the deletes will be as follows: First I will have you delete a single line near the beginning of the file; then we will delete specific characters further into the file; and finally we will delete several lines toward the end of the file.

I will have you write the modified buffer out to disk, quit the editor, and then display the file using the **more** command.

The output illustrated by the lab project will show the lines and characters that you deleted (**highlighted**), but still in the relative positions occupied before being deleted; your screen output will show the file as it really exists.

Here we go....Enter the following commands, the first of which takes you to the *top* of the

buffer.

1G

6G

(This positions cursor to line 6, we could have moved here using the **j** or return keys.)(the line begins *echo The first FIND*)

D

(removes text on line 6 to end-of-line)

/listing

(search forward for string *listing*)

dw

(remove the word *listing*)

~f

(displays next page in buffer)

/file

(move cursor down five lines)

3dd

(deletes three lines, beginning with cursor line)

Now, before I have you save the buffer with all it's changes, I want you to do one more delete so that I can show you how to *undo* the last command. This is nice to know if you ever make a delete that was not what you wanted.

2dd

(deletes the next two lines in the buffer)

u

(this is a good one to remember)

ZZ

(quit editor)

JULIE 15 >

Lets now look at the changes.

JULIE 15 > more big.file

clear

more /usr/people/student(1)/bin/welcome

echo STUDENT JOB has been started and is running

find /usr -name student -print > temp.1

```

echo The first FIND command has finished
find / -name phony -print >> temp.1
echo The second FIND command has finished
find /usr -name phony -print >> temp.1
echo The third FIND command has finished
ls -R /usr >> temp.1
echo The LS command has finished
echo end of listing >> temp.1
grep -n print.c temp.1 > temp.2
grep -n 'end of listing' temp.1 >> temp.2
echo The GREP commands have finished
cat temp.1 temp.2 > temp.3
echo The job has completed, but, a sleep command is waiting 30 seconds before
echo executing the last command
sleep 30
echo This job is finally finished!

```

Welcome OH great one!
I am an IRIS 68020 based workstation.
I enthusiastically await your command.

This file exists for only one purpose; allow students performing the UNIX SURVIVAL LAB the opportunity to copy a file over the network into their student accounts.

NOTE:

Opportunity is purposely misspelled, so that later, the student will be able to find it with the spell command.

So you see, you really did change the file. I now want you to practice deleting text in *big.file*, but, before you start, I want you to make a copy of *big.file*. Call the copy *copy.big.file*.

```
JULIE 16 > cp big.file copy.big.file
```

Now go play with *big.file*. Only practice functions that we have learned to this point. If you need help, see your instructor. When you are satisfied that you can delete characters, words, and lines of text, continue with the lab project.

STEP 8: Other Input Modes

Lets now edit the file *xcp.copy* and correct the misspelled word (opportunity). We will also add

some additional text to the file.

JULIE 17 > vi xcp.copy

Execute the next two vi commands.

/op (searchs forward to the string *op*)

a (places vi into *append* mode)

Now that you are in *append* mode, you could enter as much data as you wanted. The new data is placed into the buffer following the cursor. Correct the misspelled word by entering **p**.

p Notice that a **p** is placed into the text)

esc (Exit *append* mode)

Lets now look at *change* mode. Execute the next two commands followed by entering the text shown and watch what happens.

/is (positions forward to string *is*)

i (enter *insert* mode)

was changed using the vi editor. <return>

Exit *insert* mode and clean up buffer.

esc (exit *insert* mode)

D (deletes text to end-of-line)

return (move to beginning of next line)

dd (delete line)

You will now add some additional text.

o (open a new line below the cursor)

This line was added using the vi insert mode. <return>

esc (exit insert mode)

ZZ (quit editor and save buffer)

JULIE 18 >

Lets look at your work.

JULIE 18 > more xcp*

This file exists for only one purpose; allow students performing the UNIX SURVIVAL LAB the opportunity to copy a file over the network into their student accounts.

NOTE:

Opportunity was changed using the vi editor.

This line was added using the vi insert-mode.

Now go and experiment (on your own) with *big.file*. When you are satisfied I will show you some other useful operations of vi.

STEP 9: Text substitution.

Now I want you to remove the file *big.file*.

JULIE 19 > rm big.file

Rename *copy.big.file* to *big.file*.

JULIE 20 > mv copy.big.file big.file

JULIE 21 > more big.file

clear

more /usr/people/student(1)/bin/welcome


```
echo STUDENT JOB has been started and is running
find /usr -name student -print > temp.1
find / -name phony -print >> temp.1
echo The second FIND command has finished
find /usr -name phony -print >> temp.1
echo The third FIND command has finished
ls -R /usr >> temp.1
echo The LS command has finished
echo end of >> temp.1
grep -n print.c temp.1 > temp.2
grep -n 'end of listing' temp.1 >> temp.2
echo The GREP commands have finished
cat temp.1 temp.2 > temp.3
echo The job has completed, but, a sleep command is waiting 30 seconds before
echo executing the last command
sleep 30
echo This job is finally finished!
```

Welcome OH great one!
I am an IRIS 68020 based workstation.
I enthusiastically await your command.

NOTE:

Opportunity is purposely misspelled, so that later, the student will be able to find it with the spell command.

The next commands will show you how to change character strings in a file without going into *Text Insert* mode. The first example will change a string in one line, the second will be a global change; changing the same string in all the lines.

The word *echo* means *reverberate*. So lets change the word *echo* in the first line of *big.file*.

```
JULIE 22 > vi big.file
```

The next series of vi commands make the substitution at the third line.

```
4G          (move to line four)
shift :     (positions cursor to command line
            at the bottom of the screen.
```

```
s/echo/reverberate
```

Notice that the word *echo* was changed to *reverberate* on the line where the cursor was positioned.

Now we will change every occurrence of *echo* to *reverberate*.

```
: (drive cursor to command line)
```

```
g/echo/s//reverberate
```

```
ZZ
```

JULIE 23 >

OK, a few more points and you can experiment on your own with what you have learned. You may also want to try some of the functions illustrated by the reference guide (section 2), but, not covered by this lab project.

Remember the two files we created earlier in the lab project (*new.command* and *welcome*), I am now going to show you how these two files could affect your workstation if you place the file *new.command* into your *.login* file. This is only shown for fun and if it helps you to better understand editing and what kind of things could be done through the *.login* file, then that's a plus.

Lets look at the two files a little closer.

```
JULIE 23 > ls -l new.command welcome
```

```
-rw-rw-rw- 1 student user    46 Aug 15 14:50 new.command
-rw-rw-rw- 1 student user    99 Aug 15 14:51 welcome
```

When the commands were created the default permissions of 666 are assigned. We want to make the file *new.command* an executable file, also, lets set them so that only the owner (you) can modify the file. The permission code for this is 755.

```
JULIE 24 > chmod 755 new.command
```

```
JULIE 25 > ls -l new.command
```

```
-rwxr-xr-x 1 student user    46 Aug 15 14:50 new.command
```

We must now go and edit *.login* adding the command *new.command* to the file.

```
JULIE 26 > cd
```

```
JULIE 27 > pwd
```

```
/usr/people/student
```

```
JULIE 28 > ls -l .login
```

```
-rw-r--r-- 1 root sys 209 May 19 07:50 .login
```

Before we can edit *.login*, we must own the file. When I copied *.login* into the student accounts I did not change the ownership from *root* to *student*. We will now do this.

Since the file is owned by *root*, we must be the *super-user* to change the ownership and group names. (before going to super-user, you may want to attempt an edit of *.login* and see what the editor tells you about the file)

```
JULIE 29 > su
```

Password:

```
Enter the password:password
```

```
JULIE 1 #
```

If you remember, when you enter *super-user*, directory position in the tree is not affected, therefore, we can go right to work on *.login*.

```
JULIE 1 # chown student(1) .login
```

```
JULIE 2 # chgrp user .login
```

```
JULIE 3 # ls -l .login
```

```
-rw-r--r-- 1 student user 209 May 19 07:50 .login
```

```
JULIE 4 # exit
```

```
JULIE 30 >
```

LAB 3.2: Vi Editor

JULIE 30 > more .login

```
setenv      SHELL      /bin/csh
set         ignoreeof noglob
set         tmp=(tset -S -Q)
setenv      TERM       $tmp[1]
unset      tmp noglob
stty erase "H" kill "U" intr "C" echoe
set path = (. ~/bin /usr/local/bin /usr/uch /bin /usr/bin)
```

We will now add the command (*new.command*) to the file using the following editor commands:

JULIE 31 > vi .login

```
/path      (this positions cursor to last line using
            the search operation)

o          (this opens a new line below the cursor
            and places you into Text Input Mode)

new.command <return>

esc        (exit input mode)

ZZ         (save buffer and quit editor)
```

JULIE 32 >

JULIE 32 > more .login

```
setenv      SHELL      /bin/csh
set         ignoreeof noglob
set         tmp=(tset -S -Q)
setenv      TERM       $tmp[1]
unset      tmp noglob
stty erase "H" kill "U" intr "C" echoe
set path = (. ~/bin /usr/local/bin /usr/uch /bin /usr/bin)
new.command
```

Now logout and log back in. The contents of the file *welcome* should be displayed at your

screen after the screen is cleared.

You are now free to practice using the commands of the vi editor. When you are satisfied, complete the questionnaire and turn it into your instructor.

SECTION 2: Vi Editor Reference Guide







Vi Functions: Legend

This document can be used by persons that have used and understand the basics of the **VI EDITOR**.

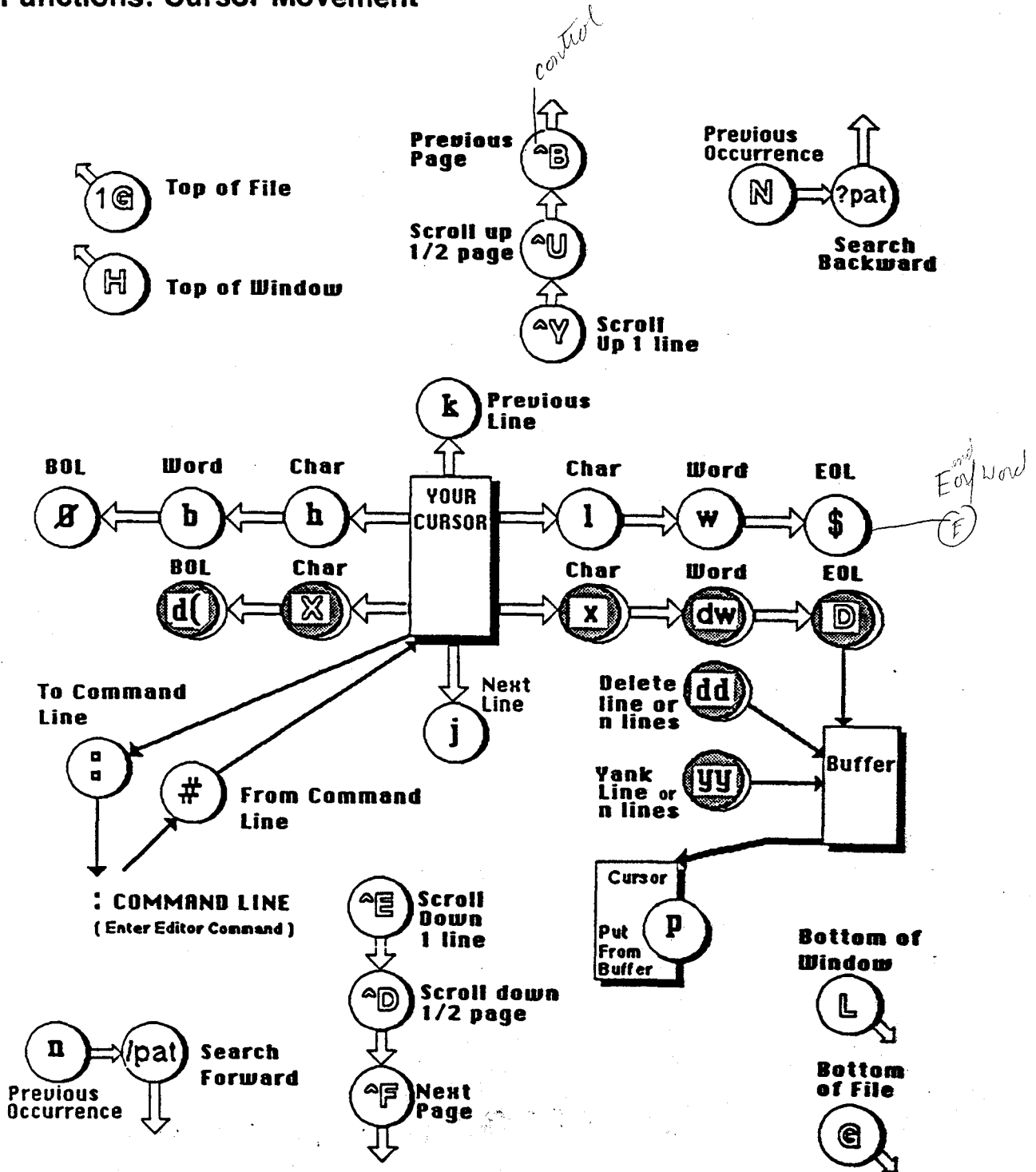
It illustrates the basic functions by using symbols. Each symbol is defined by the following legend. The sequence of key strokes is shown inside of each symbol.

NOTE:

Some functions of Vi can be destructive to your data file. These commands are indicated by using shaded symbols.

Legend	
	Movement only.
	Destructive operation.
<p>NOTE: UPPER CASE KEYSTROKE CHARACTERS ARE SHOWN IN OUTLINE FORM. LOWER CASE IN bold.</p>	
	Cursor position is important.
	This implies the Control key
	This implies the Escape key.
	This implies the Colon key.
<p>A specific command is required. Enter the indicated command that is shown next to the symbol. DO NOT enter the quotation marks.</p>	

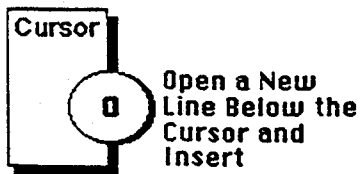
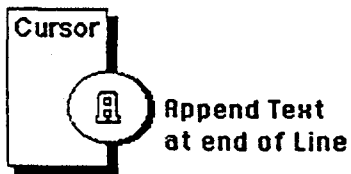
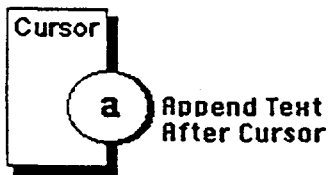
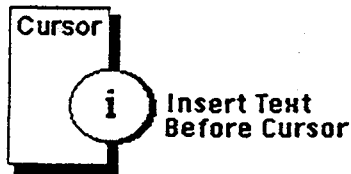
Vi Functions: Cursor Movement



Vi Functions: Input Text Modes

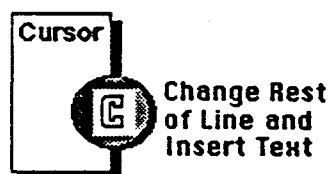
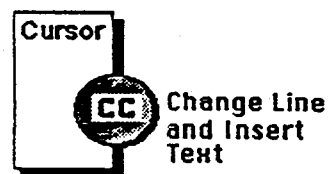
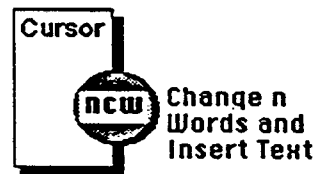
Insert Text Mode

(Escape Terminates)



Change Text Mode

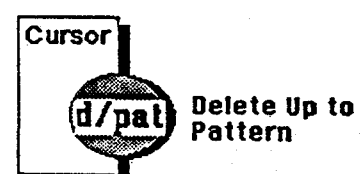
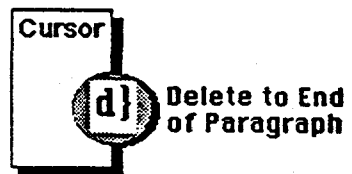
(Escape Terminates)



Note: CR Terminates
Pat = pattern

Delete Text

(CR Terminates)

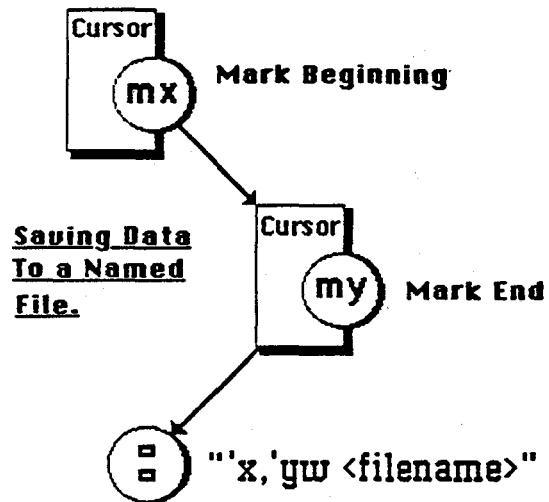


NOTE:

Any function caused by using a command on this page, or any of the other pages, can be un-done by using the UNDO command.



Vi Functions: Buffers and Files



NOTE: A variation of this function is appending the data to another file: `"x, 'yw >> <filename>"`

CHANGING DATA (Substitution)

Single Occurrence

`"s/X/Y /opt "`

Global Change

`"g/X/s//Y /opt "`

WHERE:

X = Search pattern.

Y = Substitution pattern.

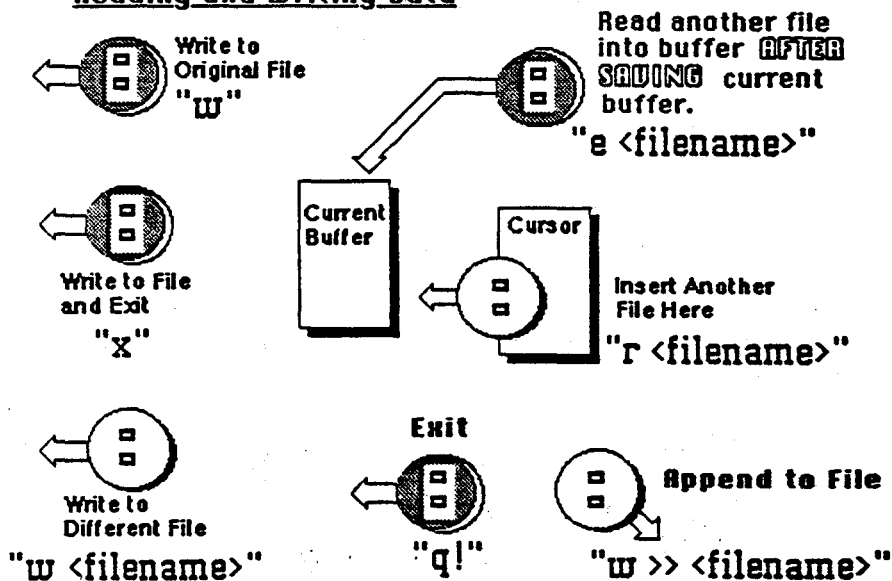
opt = Optional entry.

c = Confirm each change.

p = Print each change.

g = Change every occurrence in the line.

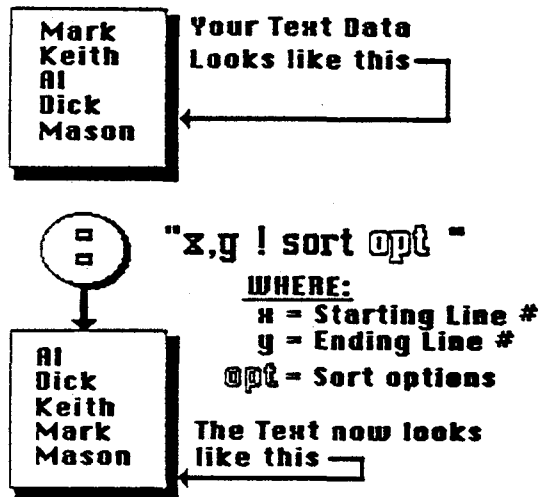
Reading and Writing Data



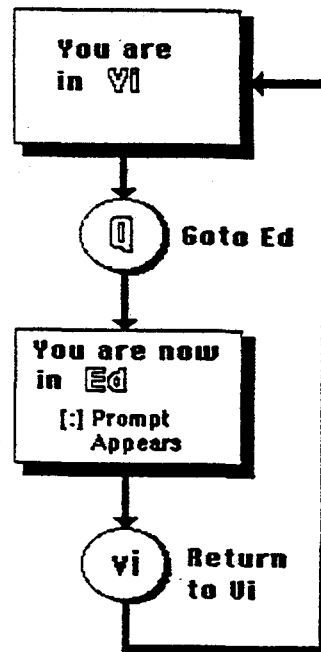
Vi Functions: Misc Functions

Vi Functions: **Misc Functions**

Sorting Text



Enter Editor from Vi



Misc Commands

^| Redraw Display

. Repeat Last Command

>> Shift Line Right One Tab Stop

<< Shift Line Left One Tab Stop

Cursor
J Join Lines

Position in Buffer

nG Goto Line n

m/ Goto Column n

NOTE:

Any function caused by using a command on this page, or any of the other pages, can be un-done by using the **UNDO** command.

u

SECTION 3: Review

Student Name:

Please complete the review and hand it into your instructor.

1. Once in vi, what key would you use to position the cursor right one character position in the line?

L

2. What key do you use to exit Text Input Mode?

ESC

3. What key strokes do you use to quit vi and save the buffer?

Shift ZZ

4. What key places you into Text Input Mode and opens a new line below the cursor?

O

5. If you are in the editor and receive a message at your workstation that was broadcast from another user on the network, how could you clear your screen of the message and see the original data that was on your screen before receiving the message?

Control L ^! ; Set, Re Draw

6. How can you insert another file into the buffer you are currently editing?

cc cP

@ n File Name

7. What keystrokes do you use to delete a word?

dw

8. What keystrokes do you use to move the cursor to line 32 of your buffer?

32 G

9. You want to delete 6 lines of your buffer from the point that you moved the cursor to, show the keystrokes.

6 DD

10. How do you get vi to the command line?

␣

ESC

11. You know that somewhere in your buffer exists the character string *abc*. You want to change the string to *xyz*, show the commands needed:

/ABC a ~~xyz~~ :s/ABC/xyz

12. Show the command for changing every occurrence of *data* in your buffer to *date*.

␣ ^{date} :g/wd /s // ^{date} ~~wm wd~~

vi Reference Card

The vi Screen Editor—a Quick Summary

The command for editing the file "filename" is vi filename.

If no such file exists yet, it will be created when this command is given.

Modes

Command Mode: Lets you use the commands described below.
To Enter: You are placed in a Command Mode when you invoke vi.
To enter the Command Mode from the Text Mode, hit the <esc> key.

Text Mode: Lets you use the keyboard to enter text.
To Enter: Any of the following commands will put you in the Text Mode: a, i, o, O, R, and c.

Using ex Commands

While in the Command Mode, type a colon and follow it with the desired ex command; for example:

:g/dog/s//mango/g

or

:14,42w newfile

You are returned to the regular Command Mode after the ex command is executed.

Cursor Movement Commands

vi commands take place at the cursor location. These commands help you to place the cursor where you want it to be in the text. The cursor will not move beyond the bounds of the existing text.

j Moves the cursor down one line.
k Moves the cursor up one line.
h Moves the cursor left one space.
l Moves the cursor right one space.
<control-d> Moves the screen down a half page.
<control-u> Moves the screen up a half page.
<control-b> Moves the screen back a full page.
<control-f> Moves the screen forward a full page.
nG Moves the cursor to the nth line of file.

Text Entering Commands

a Appends text after cursor position.
i Inserts text before cursor position.
o Opens a new line below cursor position.
O Opens a new line above cursor position.

Text Deletion Commands

x Deletes character under cursor.
dw Deletes from cursor to beginning of next word.
dd Deletes line containing cursor.
d) Deletes rest of sentence.
d} Deletes rest of paragraph.

These commands can be preceded by an integer to indicate the number of characters, words, etc., to be affected.

Text Alteration Commands

The R, cw, c) commands need to be terminated with an <esc>.

r Replace character under cursor with next character typed.
R Write over old text, beginning at cursor position.
cw Change word (beginning at cursor) to new text.
c) Change sentence (starting at cursor) to new text.
J Join next line down to line with cursor.
u Undo last command.
U Undo all changes to line with cursor.

Search Commands

/pattern Search for next occurrence of "pattern."
?pattern Search for preceding occurrence of "pattern."
n Repeat the last search command given.

The Last Command

u Use to undo the last command.
. Use to repeat the last command.
U Use to undo all changes on the current line.

Text Moving Commands (also see Text Deletion Commands)

yy Yank a copy of a line, place it in a buffer. *Number of line*
p Put after the cursor the last item yanked or deleted.
P Put before the cursor the last item yanked or deleted.
"cY Yank a copy of a line, place it in buffer c, where c is any letter from a to z.
"cP Put after the cursor the contents of buffer c.

Saving Text and Quitting the Editor

Editing work takes place in a temporary work area and must be saved by "writing" it into a permanent file.

<esc>:w Write the current text into the permanent file.
<esc>:q Quit, if no changes since last w.
<esc>:q! Emphatic form of quit, no changes written.
<esc>:wq Write and quit.
<esc>ZZ Write and quit.
<esc>:n,kw file2 Write lines n-k into another file.
<esc>:n,kw >> file2 Append lines n-k to another file.

Screen Enhancement Options

<esc>:set wm=k Provides wrap margin at k characters from right.
<esc>:set redraw Keeps screen display current.

PCF
-0
10

Review Test 2: Lessons 4 and 5

Revision A 10/23/86

INSTRUCTIONS:

1. You have 90 minutes to complete this test.
2. **DO NOT** spend too much time on any one question.
3. **DO NOT ASSUME** any facts or conditions about any question. If you do not understand the question, ask your instructor for help before you attempt to answer the question.
4. Some of the questions may have more than one answer. Select the answer that is most correct.
5. The objective of this test is to review the lesson information. Some of the questions you should be able to answer from recall, while others require that you refer to your class notes and/or supporting documentation.

While most of the questions will be on covered material, there will be some that require you to search your documents for the correct answer. For these questions, you will have to use the documents that you have been given, with your ability to extract the correct answer from them.

6. You may use your workstation and UNIX to help you determine the answer to any question.
7. When you complete this test, you may leave the room.

TEST 2

NAME:

1. What mode must the workstation be in to perform the *mkboot* procedure?

Single user / workstation in all 3 modes

2. Before you create the *Bootable* tape, what program should you run to insure that the filesystems are consistent?

Fsck

3. What command do you use to determine the amount of disk free space?

df

4. You want to produce a backup tape of the *usr* filesystem using the *cpio* command. Show the command and its arguments to do this.

cd /usr
IN DIRECTORY /usr + mounted. in root
cpio -ovh1A or *cpio -ohv1A /usr*

5. What command can you use to list the size of files or directories?

du

6. Show the command for restoring a single file called *lesson2* from the *usr* backup tape. The file lives in the directory */usr/people/rich/course*, and this is where it was when the *usr* backup tape was created.

cpio -ivhdum1A /usr/people/rich/course/lesson2

- 7. Show the command for loading the FEX utility program from the Bootable tape. Assume that your workstation is a 3030 with a 170 MB drive.

P 4.3
⑦

1 b A CTO:sifex

- 8. Show the PROM monitor command that you can use to list the headers on a tape.

4.3
31

1S A CTO:*

- 9. You have determined that a new bad spot has developed on your disk drive (either from UNIX error reports and/or results from running the disk exercise tests). What facility of FEX would you use to add the new bad spot location to the bad block table?

Storage II, 170 mb
Under SIFEX Do SHIFT Z, Then PASSWORD n The

Select Parameters equals as select unit + quit. Then Type "b"

For Bad Block edit Then "a" For add The cyl + Head - NOW PRINT IT + compare.

Then quit. Now you have to format the drive "f"

- 10. List the steps (show the commands) that you must take when you want to label a filesystem while using ~~FEX~~ UNIX. Example: For usr File System + Drive Has Been labeled

4.3/2

1. SYNC

2. Labelit n /dev/rsiof n usr n 591

3. Now Depress RESET BUTTON

I would label root after I did user that's a good idea...

- 11. What file would you edit to select your terminals baud rate at login time?

/ETC/INITTAB

4.5
11

12. What file must you edit to add a new account name to your UNIX system?

etc/passwd

13. You want to change a file ownership from *rich* to *robert*. The name of the file is *xyz*. Show the complete command that you would use to make this change.

chown robert xyz

14. What file(s) define an accounts environmentals?

.login .cshrc .login .profile

15. What cable do you use when connecting an ASCII terminal to an IRIS?

RS 232 3 wire Null modem cable.

16. What file would you edit to start a *getty* process against a newly installed ASCII terminal on port 3?

etc/inittab then do a Telinit a q so it would start the process again

17. You installed a VT100 ASCII terminal on port 2. What file do you edit to tell UNIX the terminal type, so that when you login at that terminal, the system will ask you if the terminal is a VT100?

etc/ttytype

18. What command would you execute to inform the *INIT* process that there has been a change to the file *etc/inittab*, and you want the effects of the change to be felt by this shell?

Telinit a q

19. Your system has two winchester disk drives. You have removed the data cables attached to the drives while working on one of the drives. Show what cables connect to what J connectors on each disk drive and the disk controller.

Disk 0 J2 to DSD J_2_4
 J1 to DSD J_4_

Disk 1 J2 to DSD J_2_4
 J1 to DSD J_3_

20. Your workstation has two monitors: A 60 Hrzs non-interlaced (NI) and a 30 Hrzs interlaced (I). The 60 Hrzs is the primary monitor and the 30 Hrzs is the secondary monitor.

Show what switches must be set to tell the system that you have the two monitors and that you are using the 60 Hrzs as the primary.

on IP2 Bd S1 all open + on Backpanel all close esp SW 7

	1	2	3	4	5	6	7	8
<u>ON IP2 Bd S2</u>	C	X	X	X	X	C	C	O

good

Contents

LAB 4.1: Making a Bootable Tape	1
SECTION 1: Instructional Procedure	2
I. MAKING A BOOTABLE TAPE	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT.	2
V. PROCEDURE	2
SECTION 2: Generic Procedure	13
SECTION 3: Review	14

LAB 4.1: Making a Bootable Tape

This lab project contains three sections:

Section 1: This section is used to teach you the procedure and explain to you what you are doing.

Section 2: This section lists just the procedure without all the instructional text.

Section 3: This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. MAKING A BOOTABLE TAPE

The bootable backup tape can be used to boot the workstation and rebuild the disk in case the file system on the disk is damaged beyond use. Make a bootable backup tape as soon as you install a new workstation. Make a new bootable backup tape whenever you install new software or install any new options.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly name the three files that are written on the bootable tape.
- B. Correctly name the data format that each file is written in.
- C. Using your student workstation, create a bootable tape, unassisted and within 30 minutes.

III. PURPOSE

Teach the student how to create a bootable tape for the IRIS workstation.

IV. EQUIPMENT.

- A. A functional IRIS 68020 based workstation.
- B. One blank 1/4" tape cartridge.

V. PROCEDURE

You must be in *SINGLE USER MODE* to create the bootable tape.

Two students will be assigned one IRIS workstation. The two students will work as a team with one student making the actual command entries from the primary terminal.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

1. **iris >** The prom monitor.
2. **JULIE n >** Student account prompt when in Multi-user mode, where n = command number.
3. **JULIE n #** Student account prompt when in Super-user mode, where n = command number.
4. **#** Single user mode.
5. **JULIE login:** This prompt appears when you go into Multi-user mode from Single-user.

The following text will lead you through the execution of the commands necessary to create a bootable tape. When you are required to enter a command, the text recreates the **STANDARD OUTPUT** for that command. Just keep reading and following the flow of the lab project.

Also throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **JULIE 1 > su**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in normal print.

BEGIN ACTUAL PROCEDURE HERE:

STEP 1: Power on the workstation.

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)
Configuration Switch: 0x0000
Multibus Window (2mb) at Megabytes 0 and 1.
Multibus accessible memory (1mb) begins
  at Physical memory page 300,
  at Virtual address 2000000.
iris>
```

STEP 2: Boot the UNIX System.

```
iris > b
```

```
SGI Extent Filesystem
Loading: md:0:defaultboot
Text: 038318 bytes
Data: 0113d8 bytes
Bss: 024d7c bytes
Jumping to load program ~ 20000400
```

```
SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]
(C) Copyright 1986 - Silicon Graphics Inc.
real = 4194304
kmem = 561152
user = 3633152
bufs = 819200 (max=16k)
dsd0 not installed
qic0 not installed
sii0 at mbio 0x07200 ipl 5
si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0
si1 not installed
sf0 floppy (80/2/8) slave 2
siq0 at mbio 0x73fc ipl 5
sq0 (qic02 cartridge tape) slave 0
iph0 not installed
```


LAB 4.1: Making a Bootable Tape

tmt0 not installed
ik0 not installed
nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2
fpa installed
lpen not installed
kernel debugger disabled.
root on si0a
swap on si0b. swplo=0 nswap=64000

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

#

STEP 3: Checking filesystem Consistency

Before we start, just a reminder to show you where UNIX places you after the boot process is complete.

```
# pwd
```

```
/
```

You are at the top of the tree in the *ROOT* directory (*/*).

Your first task is to check the filesystems for consistency using the *fsck* program. If any errors are reported during the check they must be corrected before you create the bootable tape.

```
# fsck
```

```
/dev/si0a
```

```
File system: root Volume: SGI
```

```

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
nnn files nnnnn K used nnnn K free

```

```

/dev/si0f
File system: usr Volume: SGI

```

```

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
nnn files nnnnn K used nnnn K free

```

Lets take a few minutes and look at what the messages from *fsck* mean.

UNIX File System Check (fsck) program interactively checks the integrity of the UNIX File System. If it detects an inconstancy during the checking phases, the administrator is asked whether or not to let *fsck* attempt to fix it. The administrator may choose to ignore the error reports and choose not to have a fix attempted; this would be very risky and total loss of the system could result.

FSCK should be permitted to run every time the system is booted. UNIX is very unforgiving if the filesystem should become corrupted, and there are many ways that corruption could occur (powering off the system without performing sync will always corrupt the system). Corruption tends to spread over time until the system is completely useless.

FSCK is started by the */etc/bcheckrc* script during the *init-cycle* of the boot-up process.

- **** Phase 1 - Check Blocks and Sizes**

- This phase concerns itself with the i-node list.
- Examines each i-node for format, size, and block count (file size actually matches the block-count).
- Checks for bad or duplicate blocks (blocks claimed by more than one i-node).

LAB 4.1: Making a Bootable Tape

- If a duplicate block is found, phase 1b is invoked to locate the i-node that previously claimed the block.
- **** Phase 2 - Check Pathnames**
 - This is used to find and remove directory entries pointing to bad i-nodes from phase one.
 - Unallocated i-nodes (directories pointing no-where).
 - Out of range i-nodes (directory i-numbers pointing beyond those blocks allocated to the directory).
- **** Phase 3 - Check Connectivity**
 - This phase lists error conditions indicating directories severed from the File Structure (lost+found) (unreferenced directories and files).
 - Notes missing or full *lost+found* directories.
- **** Phase 4 - Check Reference Counts**
 - This phase concerns itself with the link count information seen in phase 2 and 3. Error conditions related to link counts for files, bad directories or duplicate blocks, and incorrect free-i-node counts are noted and presented to the administrator (lost+found).
- **** Phase 5 - Check Free List**
 - This phase checks the free block list. It lists error conditions resulting from bad blocks in the free list, incorrect free block counts, duplicate or unused blocks in the free block list, and free blocks not listed in the free block list.

The following is a list of some of the most common errors detected and what to do about them:

- **Initializing fsck:**
 - Almost any errors here are fatal. Generally phase 1 zips through without any problems. If you have the disk write protected, phase 1 will let you know about it with write errors: *CAN NOT WRITE: BLK B*. You should respond (N) to terminate the program and properly set the permissions.
- **Phase 1:**

- **POSSIBLE FILE SIZE ERROR = I**

The i-node size does not match the actual number of blocks used by the i-node. This is a warning and is generally not of real concern.

The problem can usually be fixed by using the `find` command to print the offending i-nodes pathname then removing the file.

● Phase 2:

- **I OUT OF RANGE I-INAME=F(REMOVE)**

A directory entry F has an i-node number I that is greater than the end of the i-node list. A (Y) response will remove the offending reference.

- **DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE)**

Phase 1 has found duplicated or bad blocks in the file F, i-node I. A response of (Y) will have the offending file removed. This could be dangerous.

The directory equivalent to this error condition is identical and is more dangerous.

● Phase 3

- **UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)**

A directory was disconnected from a parent file structure. A (Y) response will have it reconnected to *lost+found*. You can then connect it whenever you wish.

● Phase 4:

- **UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)**

A file was disconnected from its parent directory. A (Y) response will have it put in the *lost+found* directory.

- **SORRY, NO *lost+found* DIRECTORY**

You really should have a *lost+found* directory in every major directory. If this message comes up in response to a *UNREF FILE* or *DIR* error condition, you've lost the directory or file you were trying to save.

- **FREE INODE COUNT WRONG IN SUPERBLOCK (FIX)**

You almost always want to respond (Y). *FSCK* will provide the correct count to the superblock.

● Phase 5:

- **FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)**

Again, answer (Y) to this. The superblock will be correctly updated.

- **BAD FREE LIST (SALVAGE)**

Bad, duplicate or missing blocks from the free list have been found. Normally a (Y) response is the best thing to do.

Additional information can be found in the UNIX Programmer's Manual, Volume IIB, section: FSCK.

The idea of *fsck* is to allow the program to fix the system. If any files were removed that the customer needs, then they can be restored from the back-up tapes. Any files detached and placed into *lost+found* can be moved back to where they belong by the administrator.

If *fsck* can not repair the system, then it would be time to rebuild the disk using the *bootable* tape. Once the system is back up, then the user could restore the rest of the system from the back-up tapes.

If there are no errors or after correcting any listed errors, continue with the procedure.

STEP 4: Mounting the USER filesystem.

Your next step will mount the user file system.....Remember, when you boot unix and the system is placed into single user mode, only the root file system is mounted by the system.

NOTE: If you performed the *df* command (this reports the number of free blocks on the disk) at this time, you may get an indication that /usr is already mounted. This is a bug in the system.

```

# df
File system  Type  kbytes  used  avail  %used  mounted on
/dev/si0a    efs   8925   4886  3898   56%    /
/dev/si0f    efs   39865  18086 21155  46%    /usr

```

The second entry (you may not have it, or you may have several false entries) indicates that the user file system is mounted.

Let me prove to you that this is a bug and *usr* is really not mounted.

By changing the directory to */usr* and executing *ls*, you will see that the directory is empty. This is because it has not yet been mounted.

```
# cd /usr
```

```
# ls
```

```
#
```

Notice that no output is produced by the *ls* command. This is because it is not mounted.

Now, on with the procedure.....

Once again return to the root directory.

```
# cd /
```

Mount the user file system.

```
# mount /dev/si0f /usr
```

The above command mounts the file system */dev/si0f* to the directory */usr*.

NOTE:

If you had not returned to the root directory (*cd /*) before executing the *mount* command and you were still in the */usr* directory, you would get

mount: Device busy message on your terminal when you attempted the mount.

Now if you executed the **DF** command again, you will see a new line in the output indicating the mount.

```
# df
```

```
File system  Type  kbytes  used  avail  %used  mounted
/dev/si0a    efs   8925   4886  3898   56%    /
/dev/si0f    efs   39865  18086  21155  46%    /usr
```

We now have a second line in the table indicating the */usr* mount.

At this time install the tape marked *bootable* into the workstation tape drive. (make sure that it is not write protected)

The following table shows the storage capacity of the different tapes SGI can handle:

Tape Width	Tape Length	Capacity
Quarter-inch	450 ft.	40 Mb
Quarter-inch	600 ft.	60 Mb
Half-inch	2400 ft.	44.6 Mb

Note the size (in kbytes used) of the user file system. If this value, when added to the size of the root file system (kbytes used), is greater the 35 mega bytes when using a *450 foot* tape, then you will need to backup the */usr* filesystem on additional tapes. The procedure to do this is covered in the next lab project and outlined in the IRIS Owner's Guide.

You will now insure that the tape is at load point by executing a rewind command.

```
# mt rew
```

The next command will create the bootable tape by placing the following on the tape:

```
File 1 = /stand      cpio format      (The utilities)
File 2 = /root       dd format
File 3 = /usr        cpio format
```

```
# mkboot /usr
```

Make sure the system is idle: no other user activity.

```
/dev/si0a
```

```
File System: root Volume: SGI
```

```
**Phase 1 - Check Blocks and Sizes
```

```
**Phase 2 - Check Pathnames
```

```
**Phase 3 - Check Connectivity
```

```
**Phase 4 - Check Reference Counts
```

```
**Phase 5 - Check Free List
```

```
403 files 4886 K used 3898 K free
```

```
mkboot: file 1 = cpio of /stand
```

```
mkboot: file 2 = dd of si0a
```

```
150+0 records in
```

```
150+0 records out
```

```
mkboot: file 3 = cpio of /usr
```

```
mkboot complete.
```

The process (when writing both */root* and */usr*) will take about **20 minutes**.

NOTE:

If you could not fit the user filesystem onto the bootable tape, then you would omit the */usr* argument from the above command.

You are now prepared for the worst case failure of your disk system; a filesystem that becomes so corrupted, that *fsck* can not repair it.

SECTION 2: Generic Procedure

This section shows only the actual steps required for creating a bootable tape.

The process is outlined in the IRIS Series 3000 Owner's Guide, the section on Workstation System Administration.

1. Reboot the system into single-user mode.

```
reboot  
b  
fsck
```

2. Correct any errors reported by FSCK.

3. Change your working directory to the root directory.

```
cd /
```

4. Mount the user file system:

For IRIS 3020:

```
mount /dev/md0c /usr
```

For IRIS 3030:

```
mount /dev/si0f /usr
```

5. Mount your blank tape.

6. Rewind the tape.

```
mt rew
```

7. Run the "mkboot" program.

```
mkboot /usr
```

If /usr is too large, back it up onto a separate.

SECTION 3: Review

Student Name:

Please complete the review and hand it into your instructor.

1. What mode must the workstation be in to perform the *mkboot* procedure?

Single user

2. What command would you use to inform any users on the network that you were going to reboot the system?

WALL

3. In what manuals could you find information concerning FSCK?

1A

4. What disk partition is the */usr* filesystem stored in?

F

5. List the formats that the data is written in on the bootable tape for each file:

File 1 = cpio

File 2 = dd

File 3 = cpio

6. How could you create a bootable tape with only the */stand* directory and the */root* file system on the tape?

mkboot

LAB 4.1: Making a Bootable Tape

7. What command is used to determine disk free space?

D.F

8. How can you really tell if a filesystem is mounted?

cd /usr & ls

9. If you are in *multi-user* mode, how do you get to *super-user* mode?

cd / su

10. Show the command for a *tape rewind* operation.

mt Rew

LAB 4.2: Making System Backup Tapes

This lab project contains three sections:

Section 1: This section is used to teach you the procedure and explain to you what you are doing.

Section 2: This section lists just the procedure without all the instructional text.

Section 3: This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. MAKING A BACKUP TAPE

This procedure shows you how to back up the ROOT and USER file systems on separate tapes. These tapes could then be used to restore individual files or directories that may become corrupted.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Explain the difference between BACKUP tapes and the BOOTABLE tape, correctly stating under what conditions each would be used.
- B. Using your student workstation, create a backup tape of the Root and User file systems, unassisted and within 30 minutes.
- C. Using your student workstation, save several specific files onto a backup tape and then restore just one of the files. You will do this unassisted and within 30 minutes.

III. PURPOSE

Teach the student how to create backup tapes and restore specific files or directories of the IRIS workstation Unix file system.

IV. EQUIPMENT.

- A. A functional IRIS 68020 based workstation.
- B. Two blank 1/4" tape cartridges.

V. PROCEDURE

Two students will be assigned one IRIS workstation. The two students will work as a team with one student making the actual command entries from the primary terminal.

You must be in *SINGLE-USER MODE* to create the backup tape of the Root file system.

If you want to backup file systems other than Root, you could be in *multi-user mode*, but if you had to correct a file, you may be competing for the file with other users, therefore, it is best that you be in *single-user mode*.

If you are backing up other file systems (or files) other than Root, you could jump to **STEP 5**.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

- | | |
|-----------------|---|
| 1. iris > | The prom monitor. |
| 2. JULIE n > | Student account prompt when in Multi-user mode, where n = command number. |
| 3. JULIE n # | Student account prompt when in Super-user mode, where n = command number. |
| 4. # | Single user mode. |
| 5. JULIE login: | This prompt appears when you go into Multi-user mode from Single-user. |

The following text will lead you through the execution of the commands necessary to create backup tapes of */root* and */usr* filesystems. When you are required to enter a command, the text recreates the **STANDARD OUTPUT** for that command. Just keep reading and following the flow of the lab project.

Also, throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. JULIE 1 > su

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in **normal print**.

BEGIN ACTUAL PROCEDURE HERE:

If you did not power off your workstation at the end of lab project 4.1, then you should still be in *single-user mode* and you can goto **STEP 3**, but first unmount the */usr* filesystem using **umount /usr**.

If you had gone into *multi-user mode* at the end of lab project 4.1, then you must enter *super-user* and reboot the workstation. If this is the case, then execute **reboot** and goto **STEP 2**.

STEP 1: Power on the workstation.

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)
Configuration Switch: 0x0000
Multibus Window (2mb) at Megabytes 0 and 1.
Multibus accessible memory (1mb) begins
  at Physical memory page 300,
  at Virtual address 2000000.
iris>
```


STEP 2: Boot the UNIX System.

iris > b

SGI Extent Filesystem

Loading: md:0:defaultboot

Text: 038318 bytes

Data: 0113d8 bytes

Bss: 024d7c bytes

Jumping to load program ~ 20000400

SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]

(C) Copyright 1986 - Silicon Graphics Inc.

real = 4194304

kmem = 561152

user = 3633152

bufs = 819200 (max=16k)

dsd0 not installed

qic0 not installed

sii0 at mbio 0x07200 ipl 5

si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0

si1 not installed

sf0 floppy (80/2/8) slave 2

siq0 at mbio 0x73fc ipl 5

sq0 (qic02 cartridge tape) slave 0

iph0 not installed

tmt0 not installed

ik0 not installed

nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2

fpa installed

lpen not installed

kernel debugger disabled.

root on si0a

swap on si0b. swplo=0 nswap=64000

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay*

with us.

#

LAB 4.2: Making System Backup Tapes

STEP 3: Checking filesystem Consistency

Before you backup a filesystem you should run *fsck* against the filesystem to check for any corrupted files.

If *fsck* detects problem files, you would then want to correct the problem before backing up the system. To do this, you must be in *single-user mode* to ensure that no other users are on the system while you are correcting the problem files.

```
# fsck
```

```
/dev/si0a
```

```
File system: root Volume: SGI
```

```
** Phase 1 - Check Blocks and Sizes  
** Phase 2 - Check Pathnames  
** Phase 3 - Check Connectivity  
** Phase 4 - Check Reference Counts  
** Phase 5 - Check Free List  
nnn files nnnnn K used nnnn K free
```

```
/dev/si0f
```

```
File system: usr Volume: SGI
```

```
** Phase 1 - Check Blocks and Sizes  
** Phase 2 - Check Pathnames  
** Phase 3 - Check Connectivity  
** Phase 4 - Check Reference Counts  
** Phase 5 - Check Free List  
nnn files nnnnn K used nnnn K free
```

If there are no errors continue with the procedure, but if *fsck* detected any errors, then ask your instructor for assistance.

STEP 4: Backing up the ROOT filesystem.

Install the tape labeled *ROOT BACKUP*.

You are now ready to create the backup tape of the Root file system.

The next command will write the Root File System onto your tape using the `cpio` command. The (1) following the (h) option will cause a tape rewind before the write is started and after the write is complete. (see *cpio(1)* in UNIX Programmer's Manual, Vol. 1A if you are interested about the functions of the *o*, *v* and *h* options)

```
# cpio -ovh1 /
```

Rewind

The (/) is the *name-list* argument and it indicates the *root* directory. The *root* directory and all sub-directories below it will be saved (stored) to tape.

NOTE:

If you had your tape write protected at this time, the following message would appear:

```
qic0: write protected
cpio: Can't open /dev/rmt1 for writing.
```

The write takes about 5 minutes and when it completes, the (#) prompt is displayed.

Now, remove the tape and install the tape labeled *USR BACKUP*.

STEP 5: Backing up the USR filesystem.

Your next step will mount the user file system.....Remember, when you boot unix and the system is placed into *single-user mode*, only the root file system is mounted by the system.

```
# mount /dev/si0f /usr
```

The above command mounts the file system */dev/si0f* to the */usr* directory.

You will next change directories to the *usr* directory.

```
# cd /usr
```

Lets now look at the size of the *usr* directory. We want to determine if its size (in mega bytes) is too large for the tape. Use the `df` command for this task.

```
# df
```

```
File system  Type  kbytes  use   avail  %used  mounted
/dev/si0a    efs   8925   4886  3898   56%    /
/dev/si0f    efs   39865  18086 21155  46%    /usr
```

The above output shows the size of the *usr* file system. (Remember, you may have duplicate lines because of a system bug reporting the disk free space). If the size is too large for the tape (see table below) then you must split the file system up into smaller parts and save them on additional tapes.

Tape Width	Tape Length	Capacity
Quarter-inch	450 ft.	40 Mb
Quarter-inch	600 ft.	60 Mb
Half-inch	2400 ft.	44.6 Mb

For this project (since we are on the training workstations) you will only need one tape. Following the " `cpio write` " you will be shown how to split the file system if it was too large.

Lets now write the file to tape using the `cpio` command.

```
# cpio -ovh1 .
```

The (.) following the option argument tells the system to use the current directory, which is, */usr*. (Remember, the [1] following the [h] option instructs that a rewind take place before and after the write operation.) The complete tree below *usr* is saved.

When the write is complete (about 20 minutes on your training workstations) the # prompt is displayed.

NOW LETS LOOK AT THE CASE WHEN THE FILE SYSTEM IS TOO LARGE FOR ONE TAPE TO HOLD.

ALTERNATE STEP 5: Backing up USR on multiple tapes.

Lets say, that when you performed the **df** command, the following output occurred:

```
# df

Filesystem Type kbytes use avail %use Mounted on
/dev/si0a efs 9424 6110 3090 66% /
/dev/si0f efs 103196 65162 35510 65% /usr
/dev/si1g efs 128736 82006 43574 65% /d
```

Notice that the **/usr** file system is 65162 K bytes (65 meg) in length. This is too big for a 450 foot tape.

You must now perform the **du** command to get the size of each file and directory. The following output from the **du** command has been modified to show you the size of each directory in the **/usr** directory. The directory names have been **highlighted** with bold print.

```
# du

7/usr/adm
4061/usr/bin
192/usr/d1ct
71/usr/games/lib/quiz
109/usr/games/lib
1156/usr/games
1/usr/include/EXOS/net
1/usr/include/EXOS/sys/mdep
2/usr/include/EXOS/sys
4/usr/include/EXOS
129/usr/include/gl2
26/usr/include/gpib
77/usr/include/multibus
29/usr/include/pmII
126/usr/include/sys
```

LAB 4.2: Making System Backup Tapes

```
60/usr/include/vkernel
19/usr/include/xns
617/usr/include
262/usr/lib/acct
25/usr/lib/font
52/usr/lib/gl2
47/usr/lib/help
9/usr/lib/lex
194/usr/lib/lint
99/usr/lib/macros
18/usr/lib/me
87/usr/lib/refer
24/usr/lib/sa
260/usr/lib/spell
1/usr/lib/tabset
35/usr/lib/term
55/usr/lib/tmac
1/usr/lib/uucp/.XQTDIR
166/usr/lib/uucp
141/usr/lib/emacs/bin
313/usr/lib/emacs/databases
144/usr/lib/emacs/doc
732/usr/lib/emacs/maclib
1332/usr/lib/emacs
4454/usr/lib
1/usr/local/bin
1/usr/local/etc
1/usr/local/include
1/usr/local/lib
5/usr/local
1/usr/mail
1/usr/man/a_man/cat1
1/usr/man/a_man/cat7
1/usr/man/a_man/cat8
208/usr/man/a_man/man1
63/usr/man/a_man/man7
12/usr/man/a_man/man8
287/usr/man/a_man
1/usr/man/u_man/cat1
1/usr/man/u_man/cat2
1/usr/man/u_man/cat3
1/usr/man/u_man/cat4
1/usr/man/u_man/cat5
1/usr/man/u_man/cat6
```

```
1066 /usr/man/u_man/man1
```



```
201/usr/man/u_man/man2
669/usr/man/u_man/man3
108/usr/man/u_man/man4
 99/usr/man/u_man/man5
 51/usr/man/u_man/man6
2201/usr/man/u_man
2489/usr/man
  1/usr/news
102/usr/people/demos/robotlib
2343/usr/people/demos
 50/usr/people/gifts/arch
 52/usr/people/gifts/archmex
 29/usr/people/gifts/examples/1.ProgGuide
 16/usr/people/gifts/examples/2.Append
  4/usr/people/gifts/examples/4.WindMgr
 38/usr/people/gifts/examples/5.ProgExamples
 32/usr/people/gifts/examples/Fortran
121/usr/people/gifts/examples
506/usr/people/gifts/mextools/images
 25/usr/people/gifts/mextools/imglib
 14/usr/people/gifts/mextools/imgtools
  8/usr/people/gifts/mextools/include
  4/usr/people/gifts/mextools/mexrcs
 18/usr/people/gifts/mextools/portlib
 58/usr/people/gifts/mextools/tools
638/usr/people/gifts/mextools
862/usr/people/gifts
  4/usr/people/guest
 65/usr/people/mexdemos/bin/menulib
395/usr/people/mexdemos/bin
140/usr/people/mexdemos/hemelib
153/usr/people/mexdemos/surflib
123/usr/people/mexdemos/zshadelib
1655/usr/people/mexdemos
  1/usr/people/rich/course/clover
  1/usr/people/rich/course/3000
  1/usr/people/rich/course/2000/workbook/appendix
 63/usr/people/rich/course/2000/workbook/lab
 68/usr/people/rich/course/2000/workbook
  1/usr/people/rich/course/2000/ig/lab
  1/usr/people/rich/course/2000/ig/test
  3/usr/people/rich/course/2000/ig
 72/usr/people/rich/course/2000
 75/usr/people/rich/course
```

82 /usr/people/rich

LAB 4.2: Making System Backup Tapes

```

38623/usr/people
  1/usr/preserve
10/usr/pub
  3/usr/spool/colord
  1/usr/spool/lp/class
  1/usr/spool/lp/interface
  1/usr/spool/lp/member
15/usr/spool/lp/model
  1/usr/spool/lp/request
23/usr/spool/lp
  1/usr/spool/lpd
  1/usr/spool/uucp/.XQTDIR
  2/usr/spool/uucp
  1/usr/spool/uucppublic/receive
  2/usr/spool/uucppublic
32/usr/spool
  1/usr/sys/conf
  1/usr/sys/h
  1/usr/sys/kernels
  1/usr/sys/multibus
  1/usr/sys/pmII
  6/usr/sys
  1/usr/test
  1/usr/tmp
65162/usr

```

By looking at the directory sizes, we see that the */usr/people* directory is the largest single directory. You could put this directory on one tape and all the other directories on a second tape.

```

i.e.
# cd /usr/people
# cpio -ovh1 .

```

The commands for the second tape would look like the following:

```

# cd /usr
# cpio -ovh1 /usr/tmp /usr/test /usr/sys ....etc

```

You would name each directory except the */usr/people* directory. Just keep entering the directory arguments to the *cpio* command. The entry will wrap to the next line on the screen. When you enter all the directory names, then depress < return >.

You would now have your system backed up on three tapes: Root on one tape and User on two tapes.

This part of the lab project is for general information. It gives examples of various *store-restore* operations.

QUESTION: What is the difference between the BACKUP tapes and the BOOTABLE tape?

ANSWER:

The Bootable tape is used when the file system is so corrupt that Unix cannot be booted. The utilities in the first file on the tape are used to rebuild the disk system. This totally destroys any existing files on the disk and restores the system to a known good state.

From the point when the bootable tape is created the user should then create the backup tapes for the *root* and *usr* filesystems.

The backup tapes are used when only specific files or directories have been lost or corrupted; UNIX will boot up, the filesystems are consistent, but certain files need restoring. These files could be loaded from the backup tapes and placed into the directories they belong.

The user should create *incremental backups* per some fixed schedule to insure that all filesystems are backed up to a known good point.

QUESTION:

What if you had a backup tape (or any cpio format tape) and wanted to know what was on the tape?

ANSWER:

Mount the tape and execute the following command.

```
cpio -ihvt1
```

or

```
cpio -ihvt1 > list
```

The first command will rewind the tape and then read the files printing a list of the pathnames on the screen for each file on the tape. Using the *t* option causes only a read of the tape without directing the output to any directories on the disk. A rewind will be performed when the read is complete.

The second command will do the same thing, but the output will be re-directed into a file called *list*.

QUESTION:

You have somehow removed a file from your system. How do you restore the file?

ANSWER:

We will assume that the file you lost was */bin/time*. Mount the Root backup tape and perform the following command. (You may want to try this on the training workstation. Go remove the file and then execute the command)

```
cpio -ivhdum1 /bin/time
```

The above command will restore *time* to the */bin* directory and print the pathname on the screen for you when it is done. A rewind will be performed after the restore.

If you wanted to restore several files, just list them as part of the argument to the *cpio* command. If you do not supply a file name, the default for the argument is the WILD CARD *meta character*, and the complete tape is restored.

QUESTION:

What if you wanted to move a complete directory from one workstation to another. You could use the following method.

Lets assume that the name of the directory is */emacs* and that it lives under */usr/lib*. You want to move it to the same relative position in the tree on the other workstation.

ANSWER:

At the workstation that has the directory, execute the following:

```
# cd /usr/lib  
# cpio -ovh1 emacs (this dumps complete tree below emacs)
```

At the other workstation, execute the following:

```
# cd /usr/lib
# cpio -ivhmd1 (without an argument, restore complete
               tape)
```

QUESTION:

What if you had some special files that you wanted saved on a scratch tape. How would you do this?

ANSWER:

For illustration, lets say you had three files that you wanted saved on tape. Lets call them: Files A, B and C. Mount your tape and execute the following commands:

```
# cd <your directory>
# mt rew (rewinds the tape)
# cpio -ohv2 A (writes file A with no rewind)
# cpio -ohv2 B (writes file B with no rewind)
# cpio -ohv2 C (writes file C with no rewind)
# mt rew (rewinds tape)
```

write no rewind.

The **2** after the **v** option (of the cpio commands) indicates that a rewind is not to be performed before or after the operation.

You would now have a tape with three files on it : A, B and C.

QUESTION:

You want to restore only the third file (C) from your tape. How do you do this?

ANSWER:

Execute the following commands:

```
# cd <your directory>
# mt rew (rewind tape)
# mt fsf 2 (forward space two files)
# cpio -ivh2 (read the third file, no rewind)
# mt rew (rewind tape)
```

NOTE:

If you wanted to know what was on the scratch tape, rewind the tape and execute `cpio -ivht2`. The name of the file will be printed on the screen.

If you issued the command again, then the name of the second file (if it exists) will be printed. This could be continued until you ran out of tape or files.

The operation just described could also be done like this:

```
# mt fsf 2          (skip forward two files)
# cpio -ivht2      (read the third file)
```

Read *No Rewind*

This completes the lab project. The next section gives only the generic procedure for creating the backup tapes.

When you are finished, complete the questionnaire in section three and turn it in to your instructor.

NOTE:

This lab project only mentioned and used the `cpio` command to create and restore the backup tapes. Another command, `tar`, could also have been used. This command is used only for tape drives, whereas, `cpio` can be used with tapes and disks to move files. The procedure is outlined in the IRIS OWNER'S GUIDE at the same place the above procedures for `cpio` are found.

SECTION 2: Generic Procedure

This section shows only the actual steps required for creating backup tapes.

The process is outlined in the IRIS OWNER'S GUIDE, section 4: Making Periodic Backups.

1. Reboot the system into single-user mode.

```
reboot  
b  
fsck
```

2. Correct any errors reported by FSCK.

3. Change your working directory to the root directory.

```
cd /
```

4. Install your blank tape.

5. Backup the Root file system.

```
cpio -ovh1 /
```

6. Mount the user file system:

```
mount /dev/si0f /usr
```

7. Install your other blank tape.

8. Change directories to USER directory.

```
cd /usr
```

9. Backup the USER files system.

```
cpio -ovh1 .
```


BRU - Tape copy

SECTION 3: Review

Student Name:

Rewind Before Rewind

Please complete the review and hand it into your instructor.

1. Show the command syntax used to backup the *root* filesystem to tape.

cpio -ovh2 /

2. You want to execute the *cpio* command, but do not want to cause a rewind before or after the operation. What character do you include as part of the options argument to insure this?

.2

3. What command can you use to list the size of files or directories?

DU

4. When would you use the Backup tapes?

To Dump File or Restore Files

5. What command would you use to print a listing of files contained on a tape written in *cpio* format?

cpio -i hvt1 > list

Tape of contents no file created

1 more

6. You want to forward space your tape four files, show the command.

mt -Fsf 4

7. You want to place two directories on a single tape. Each directory to be a separate file on the tape. Show the commands to do this.

```
set cd /usr/people  
- that was under  
CD TO Directory CPIO -0vhz Student  
_____  
CD TO Directory CPIO -0vhz Bin  
_____  
_____  
_____
```

Contents

LAB 4.3: Disk Drive Restoration Procedure	1
SECTION 1: Instructional Procedure	2
I. DISK DRIVE RESTORATION	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT.	2
V. PROCEDURE	2
SECTION 2: Generic Procedure	26
SECTION 3: Review	31

LAB 4.3: Disk Drive Restoration Procedure

This lab project contains three sections:

Section 1: This section is used to teach you the procedure and explain to you what you are doing.

Section 2: This section lists just the procedure without all the instructional text.

Section 3: This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. DISK DRIVE RESTORATION

When your workstation is experiencing disk drive filesystem errors and FSCK fails to correct the error, or you cannot boot UNIX due to disk errors, then its time to format the disk and restore the complete file system.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly list the names of each available FEX (disk formatting and diagnostic program) program and describe when each would be used.
- B. Using SIFEX, rebuild your training workstations disk drive and restore the UNIX file system. You must do this unassisted and within two hours.

III. PURPOSE

Teach the student how to reformat and rebuild a workstation disk drive.

IV. EQUIPMENT.

- A. A functional IRIS 68020 based workstation.
- B. The Bootable tape created in lab project 4.1.

V. PROCEDURE

Two students will be assigned one IRIS workstation. The two students will work as a team with one student making the actual command entries from the primary terminal.

During the project you will see four different prompts; the current prompt is determined by

where you are in the workstation systems.

1. **iris >** The prom monitor.
2. **#** Single user mode.
3. **JULIE login:** This prompt appears when you go into Multi-user mode from Single-user.
4. **sifex 2.4>** This is the top-level prompt for *sifex*. I use the term "top-level" because there are many other prompts that will occur during the process that ask you to select an option, a test, a mode, etc..

The following text will lead you through the execution of the commands necessary to *format the disk* and *restore the disk filesystems*. When you are required to enter a command, the text recreates the STANDARD OUTPUT for that command. Just keep reading and following the flow of the lab project.

Also, throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **sifex 2.4> f**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.

When executing some of the *sifex* commands, you will notice that some prompts and/or messages will appear next to the command, these messages will also be **HIGHLIGHTED USING ITALICS**.

- The instructional comments that makeup the bulk of the lab project will be in **normal print**.

BEGIN ACTUAL PROCEDURE HERE:

If you did not power off your workstation at the end of lab project 4.2, then you should still be in *single-user mode*, possibly in *multi-user* depending on what you did after the instructional portion of the project. For either case you must perform a reboot and be in the PROM monitor to begin this project.

STEP 1: Reboot the workstation.

```
# reboot
```

IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985

Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)

Configuration Switch: 0x0000

Multibus Window (2mb) at Megabytes 0 and 1.

Multibus accessible memory (1mb) begins

at Physical memory page 300,

at Virtual address 2000000.

iris>

STEP 2: Booting sifex from tape.

At this time install the Bootable tape created during Lab 4.1.

FEX is a Formatter and EXerciser (thus its name) for winchester disk drives on IRIS workstations. It will handle any one of the several types of disk drives that an IRIS may be using. Three *fex* programs exist; with the type of drive installed the determining factor as to what *FEX* program you will use.

- **MD FEX:**
Used with small winchester drives. i.e Vertex, Priam, and Maxtor.
- **IP FEX:**
Used when your disk drive is the 474 MB Fusitsu disk drive.
- **SI FEX:**
Used when your disk drive is the 170 MB Hitachi disk drive.

Since the training workstations contain the 170mb winchester type disk drives, this project will illustrate the use of SIFEX.

The method you use for SIFEX is the same for the other two programs. They are all *menu-driven* and if you have a good understanding of SIFEX you will not experience any problems running the IPFEX and SIFEX programs.

Before we actually begin the process, let me give you some general information about the *FEX* programs; command syntax, responding to prompts, and special keyboard control keys.

- Most of your input to *FEX* is performed using a single character without using the return key. When you are entering a command, *FEX* will wait for you to enter a single character and then immediately execute the command. The exception to this is when *FEX* is requesting a " string " of characters or numbers from you.
- Strings may be edited using **backspace** to delete the previous character, and **ctrl- u** to delete all characters back to the last prompt.
- When *FEX* prompts for a value or a response, the default value or response is shown in parenthesis ().
- Using the **DEL** key will return you to the top level command prompt (**sifex 2.4>**) from anywhere in *FEX*.

You must now boot the desired *fex* program from your bootable tape. Remember, the first file on the bootable tape contains the stand-alone *fex* programs (some people call these utilities). Before you can boot the program you have to know what prom monitor " boot command " to use.

We will ask the prom monitor to display the available commands for this prom monitor. We do this using the **help** command.

```
iris > h
```

General Monitor Commands (All numeric values in hex):

MEDIA is one of the following:

hd- hard disk. (look for *ip*, *sd*, then *md*)

ct- cartridge tape. (look for *st*, then *mt*)

fd- floppy disk. (look for *sf*, then *mf*)

ip- interphase disk. (474 MB drive)

sd(or si)- storager disk. (170 MB drive)
 md- midas disk. (72 MB drive)
 st(or sq)- storager cartridge tape. (1/2" tape)
 mt(or mq)- midas cartridge tape. (1/4" tape)
 sf- storager floppy disk.
 mf- midas floppy disk.
 xns- network. (ethernet)
 rom- EPROM board.

DEVSPEC is one of the following:

host name- Name of the host. (MEDIA must be xns)
 unit- unit number of device (0, 1, ...).
 (MEDIA must be a tape or disk device)
 <unit><fs>- unit number and filesystem (a-h)
 (MEDIA must be a disk device.)
 address- multibus address.
 (MEDIA must be a EPROM board.)

[MEDIA.DEVSPEC:][file] load and begin execution of the named file
 file defaults to defaultboot
 SPECS are from switch settings

b [MEDIA.DEVSPEC:][file] *same as above*

n [DEVICE:][file] *same as b with MEDIA = xns*

ls [MEDIA.DEVSPEC:][file] *list the files on the device*

l [MEDIA.DEVSPEC:][file] *load but don't begin execution of the file*

g address [stack] *start executing at specified address.
 the stack address is a option.*

Continue (y or n)?: n

We can see that the command used to boot from tape is ct.

Before we boot *fox*, lets perform a little review of some operations talked about in lesson 3 (see the page entitled PROM MONITOR COMMAND EXAMPLES).

Lets list the contents of the *root* partition.

```
iris> ls si0a:
```

```

.                ..                lost+found         usr
.cshrc           .login            .profile          Versions
etc              kernels           lib               stand
tmp              vmunix             vmunix1          ovmunix
```

This is a very basic command and if you could not read the disk you have serious problems with your disk subsystem. If you have a second disk, then you should try to list the contents of one of it's partitions, and if it fails start thinking *disk controller* problems.

Lets see what is contained on the first file on your Bootable tape.

```
iris > ls ct0:*
```

```
mdfex
ipfex
sifex
```

*- I F Doesn't work
Tape could have Ran of end
Error code No Device or address*

OK, these are the programs that were placed here by the *mkboot* program, so I guess we can now boot the one we want, *sifex*.

```
iris> b ct0:sifex 1
```

```
SIFEX for Hitachi 512-17 Disk Drives and QIC-02 Tape
Interphase Storage II Disk/Tape Controller
** Version: 2.4
```

```
*** Drive: 0 Name: Hitachi 512-17, Serial: 1217
```

```
sifex 2.4>
```

STEP 3: Entering Field Engineer portion of sifex.

Two command sets exist for *sifex*, one for the user and one for the Field Engineer.

- **User Command Set**

This is a limited set of commands that will allow the user access to *sifex* for the purpose of restoring disk files from tape.

- **Field Engineer Command Set**

This is the complete set of commands that allow disk formatting, disk drive testing, and tape-to-disk copying.

In order to gain access to the full set of commands, one of eight passwords must be entered. To get *sifex* to prompt you for the password you must enter an upper case **Z** to the initial prompt.

```
sifex 2.4> Z
```

```
*** SECURITY PASSWORD ***
```

```
enter passWord:
```

You must now enter one of eight passwords in response to the above prompt. Each password corresponds to a letter in the prompt word, *passWord*; the letter that is set in upper case dictates what password you will use.

Following is the list of passwords and the corresponding *password* letter:

- If **P** use *carter*.
- If **A** use *ludwig*.
- If first **S** use *chase*.
- If second **S** use *darrah*.
- If **W** use *donl*.
- If **O** use *bradley*.
- If **R** use *ellis*.
- If **D** use *luttner*.

Since the above prompt has the **W** set in upper case, I used *donl* to gain access to the full set of *sifex* commands.

Now, enter your password. If you did not understand the explanation, see your instructor.

```
enter passWord: donl
```

```
acceptedrifex 2.4>
```

We are now into *sifex*, lets list the available commands using the **help** command.

```
acceptedrifex 2.4> h
```

```
*** SIFEX -- COMMANDS ***
```

```
q    -quit and return to IRIS prom monitor.  
s    -set miscellaneous variables.  
t    -tape copy to disk utility.  
  
b    -enter bad block edit mode.  
c    -copy data.  
e    -run drive read/write/seek tests.  
f    -format the selected drive.  
h    -print the help message.  
m    -map out bad track.  
r    -restart the test & read the label.  
u    -update disk label.
```

Each of the above commands will take you into a different routine where specific functions are executed from. During the course of this lab project we will enter several of the routines and execute various commands within each routine in order to perform the complete task of rebuilding a disk drive.

NOTE:

This program was initially created to allow manufacturing to prepare, test, and name disk drives. You will only learn the commands that you need to rebuild a damaged disk drive. Do not concern yourself with commands not covered by this procedure.

STEP 4: Selecting the Disk Drive.

Now that you have gained access to *sifex*, you must select the drive that you want to work on. To select a drive for formatting or testing you must enter the *set miscellaneous variables* routine using the *sifex* command **s**.

sifex 2.4> s

Set ?

Once into this routine, *sifex* asks you what variable you want to set. Lets list the available *set commands* using the *help* command.

Set ? h

*** Set commands ***

l -change drive file system information
 q -quit and return to main routine
 t -select type of drive (? for list)
 u -select unit for testing (0/1/f)

a -i/o port (and WUB) address
 b -display current drive bad block list
 c -change drive label information
 d -display settings
 f -disable firmware retries
 i -set interleave
 h -this message
 m -mode of disk address entry
 r -reset random number sequence
 v -verbose (on/off)
 w -write lock switch (on/off)
 Q -quiet all extraneous printout (on/off)

To select a drive for testing or formatting use the *u* command.

Set ? u Unit (0)= 0

By entering the *u*, you told *sifex* that you wanted to select the unit. It shows you the default (0), and is waiting for you to accept the default or enter " 1 or f ". You will enter **0** for the training system. (0 = front drive, 1 = side drive)

At this moment all you needed to do is select the drive, which you have done, so we will quit this routine and return to the *sifex* control routine. We will return to this routine later.

Set ? q

Quit

STEP 5: Entering Bad Block Data.

Before you continue, it is recommended that you check your Bad Block Table and verify that it contains the entries that are listed on the sheet of paper attached to the top of the disk drive.

The Bad Block Table is used during formatting to define where bad spots are located on the disk surface. Knowing where the bad spots are, allows the track to be flagged as bad and the assignment of an alternate to be made.

The Bad Block Table is located on track zero of the drive.

NOTE: There is a chance that your Bad Block Table will not match the listing. This is due to the way SGI initially formatted some drives at the factory. If it does not match, then you must enter the data.

To change or display the Bad Block Table, you must enter the *bad block edit* routine using the **b** command.

```
sifex 2.4 > b Bad Block edit, type h for help
bb >
```

Lets do as suggested and request help.

```
bb> h Help -- choose one of
add bad blocks
clear bad block list
edit list
print list
quit
setup alternates
zap alternate assignments
```

We first want to print the list to see if it matches the hard copy attached to the top of the drive.

```
bb > p Print bad blocks:
```

You will get an output of the entries. They are in the format:

```
xxx/y
```

```
xxx = cylinder number
```

y = head number

NOTE:

If your table does not match the hard copy listing, then perform the following NUMBERED steps, else skip the numbered steps and goto **STEP 6** after entering **q** to quit *bad block edit routine*. (The nine hundred numbers are the alternate tracks that the FEX program assigned to the existing bad blocks, ignore them)

1. Clear the bad block table by entering " c ".
2. Enter " y " to the " confirm " prompt.
3. Enter " a " to " Add Bad Blocks ".

The following prompt appears:

Add new entries. Mode cyl/hd/sec, end with a blank line:

bb add:

NOTE:

Enter only the CYLINDER and HEAD information from the hard listing.

i.e.

31/0

921/6

135/3

When you are finished entering the bad block data, terminate the entry by depressing return without giving a " next " entry. You will actually add a blank line to the table this way.

4. Return by itself to terminate the entries.....

Once you are satisfied that your Bad Block Table contains the correct entries, quit the Bad Block Edit mode.

bb > q

Quit

STEP 6: Formatting the Disk.

You will now instruct the program to format the disk. It will take about 8 minutes on the small disk.

```
sifex 2.4 > f
```

```
*** WARNING -- ALL DATA ON UNIT 0 WILL BE LOST!!!
```

```
Drive: Hitachi 512-17 UNIT=0, (970+177/17 (512)) ILV=1
Type 'go <return>' to start...
```

```
go <return>
```

The following output will accumulate on the screen...

```
Starting format....
```

```
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
..... 980
```

```
Formatting bad tracks....
```

```
Writing label....
```

```
Formatting complete.
```

NOTE:

The drive is now formatted and ready to receive your UNIX file systems, but the lab project will now show you how to test the drive for new bad spots. You would not format the pack followed by these tests in the field, since the tests distory the format and you would then have to again run the format routine. This is backwards from the way you would normally work on the drive in the field. (See the flow chart in section two for the actual sequence you should use.)

STEP 7: Disk Exercise Tests.

We are now going to exercise the disk and see if any new bad spots are detected. If any new spots are detected we will again enter the bad block edit mode and enter the new bad spot data.

The test takes about 8 minutes to complete one cycle. A thorough check would take 12 - 24 hours, but run the test for the amount of time that you have.

We will only run the test a couple of passes on the training system. After the test is run, you will access an error table (maintained by sifex) to see if any new bad spots were detected.

You will now run the disk drive exercise tests....

Since complete *read/write* testing will use write operations, the *write lock* must be set off. The *write lock* is set on when you enter *sifex*; this is the default and it prevents any write operations from being performed. (you do not need to change this lock while performing format writes)

To change the *write lock* you must again enter the *set miscellaneous variables* routine.

```
sifex 2.4> s
```

Set ?

Lets again list the menu using the *help* command.

```
Set ? h
```

*** Set commands ***

```
l -change drive file system information
q -quit and return to main routine
t -select type of drive (? for list)
u -select unit for testing (0/1/f)

a -i/o port (and WUB) address
b -display current drive bad block list
c -change drive label information
d -display settings
f -disable firmware retries
i -set interleave
h -this message
m -mode of disk address entry
r -reset random number sequence
v -verbose (on/off)
w -write lock switch (on/off)
Q -quiet all extraneous printout (on/off)
```

The command we want is the *w* command.

```
Set ?w Write lock (on)?
```

The program is telling you that the *write lock* is set on (on), and it is asking you if you want to change it (?). You will respond by entering *off* to the prompt.

?off

You must now quit the *set miscellaneous variables* routine.

Set?q

sifex 2.4>

Lets again list the available *sifex* routines.

sifex 2.4> h

*** SIFEX -- COMMANDS ***

q -quit and return to IRIS prom monitor.
 s -set miscellaneous variables.
 t -tape copy to disk utility.
 b -enter bad block edit mode.
 c -copy data.
 e -run drive read/write/seek tests.
 f -format the selected drive.
 h -print the help message.
 m -map out bad track.
 r -restart the test & read the label.
 u -update disk label.

Now, to select the disk exercise tests routine using the **e** command.

sifex 2.4 > e

Drive: Hitachi 512-17 Unit=0, (970=17/7/17 (512) ILV=1
 Which exercise?

Since you have not been here before, lets list the available tests using the **l** command (stands for list).

Which exercise? l Exercise help -- Choose from:

complete write/read multi pass/multi pattern.

Complete write/read one pass.

disk read or write multiple.

Disk read or write single chunk.

error display/reset

multiple sector write/read repeated.

Multiple sector write/read once.

quit.

random reads.

Random number reset.

seek tests canned.

Seek tests selective.

track test multi pass/multi pattern.

Track test one pass.

xtra disk read write multiple tracks.

To select a test you must enter the *first letter* of the above listed tests. We are going to enter **c** to run the *complete write/read multi pass/pattern* test.

Notice that some of the tests begin with an *upper case* letter.

OK, lets run the test.

Which exercise? **c** *Complete Exercise -- track writes and reads*

Repeat how many times?

It is asking you for a number, if you depress **<return>**, the test will loop until you kill it by using the **a** key.

Repeat how many times? **<return>**

Alternate units?

Here the test is asking you if you want to alternate between the unit you have selected (unit 0) and the other disk drive. The default for this question is **no**, and thats what is selected if you depress **return**.

Alternate units? **return**

The following message is sequentially output and continues until you terminate the test using the **a** key. Let the test run two passes and then terminate it.

```
START LOOP 1  UNIT 0: Pattern 0xb1b6dbbd
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
..... 980
```

The next message will look the same except for the loop count.

After two passes, terminate the test using the **a** key.

a

Which exercise?

While the test was running, a log is maintained by the program to record any disk errors detected during the test. You must now display the log to check if any errors were detected.

NOTE:

The longer the test is allowed to run, the greater the chance of detecting new bad spots, if they exist.

Let see what command you use to print the *error log*.

Which exercise? 1 Exercise help -- Choose from:

complete write/read multi pass/multi pattern.

Complete write/read one pass.

disk read or write multiple.

Disk read or write single chunk.

error display/reset

multiple sector write/read repeated.

Multiple sector write/read once.

quit.

random reads.

Random number reset.

seek tests canned.

Seek tests selective.

track test multi pass/multi pattern.

Track test one pass.

xtra disk read write multiple tracks.

The command is **e**.

Which exercise? **e** *Error display or Reset?*

It is asking you if you want to *display* or *reset* the error log. We want to *display* it.

Error display or Reset? **d**

If any new errors were detected, they would be listed here.

Lets quit the disk exercise routine.

Which exercise? **q**

sifex 2.4>

If you have new errors, then you must again enter the *bad block edit* routine and enter the *head* and *cylinder* information into the *bad block table*, goto **STEP 8**.

If no new errors were detected, then goto **STEP 9**.

STEP 8: Adding new Bad Blocks to Bad Block List.

NOTE:

The following steps just show the commands you enter and not the prompts or the screen output since you have already seen it.

1. Enter **b** (This enters Bad Block Edit routine)
2. Enter **a** (Add bad blocks -- Enter new bad spots as before)
3. Depress **<return>** (to terminate the edit process)
4. Enter **q** (quit edit routine)

5. Enter **e** (This enters exercise test routine)
6. Enter **e** (This enters error display or reset mode)
7. Enter **r** (This resets the error log)
8. Enter **q** (quit exercise test routine)

STEP 9: Re-formatting the disk drive.

You must now run the *format* program again, the reasons why:

- If you had to enter additional *bad block* information (step 8) into the *bad block table*, then, running *format* will assign alternate tracks for the newly added bad tracks, or,
- You entered this step from step 7, where you ran the complete set of tests against the drive, which destroyed all the format on the pack.

In either case, you must again format the pack.

The procedure would be the same as it was in step 6, except this time you would not run the exercise tests, you would jump to step 10 after the format was complete.

At this point you have a disk drive that is formatted and ready to receive the operating system. You will now restore the ROOT file system using the **Tape Copy to Disk** utility portion of sifex.

STEP 10: Copy Root File System From Tape to Disk.

You must now enter the *tape copy to disk utility* routine.

sifex 2.4 > **t** *Tape to Disk Copy*

Tape file (2) ?

Mdfex is asking you if you wanted the second file on tape. You say yes by entering <return> and accepting the default of 2. The ROOT file system is the second file on the Bootable tape.

Tape file (2) ?<return>

Unit (0) =

Its now asking you what drive to restore the filesystem to; disk drive 0 being the default. Depressing return accepts the default.

Unit (0) =<return>

File system (a) ?

Now it wants to know what partition to install the filesystem into; the default is **a**, and that is where we want ROOT to go. Depressing <return> will accept the default.

File system (a) ?<return>

Disk: Vertex V170 (987/7/17)

Copying 17850 blks in 714 chunks from tape file 2 to si0a

Type 'go <return>' to begin.....

go <return>

Started

Rewinding....

Copy started....

5 10 15 20 25 30 35

Rewinding.....

Tape to Disk copy complete

sifex 2.4 >

You are now ready to quit *sifex* and boot UNIX. Once UNIX comes up, you will load the user file system into partition "c".

sifex 2.4 > q Quit -- confirm quit with 'y':

If you really want to quit *sifex*, then enter **y**.

Quit -- confirm quit with 'y': **y**

At this point the workstation is rebooted by *sifex* and you are placed into the PROM monitor.

STEP 11: Copy User File System From Tape to Disk.

The next step is to boot the system using the ROOT file system that you just installed.

iris > **b**

SGI Extent Filesystem

Loading: md:0:defaultboot

Text: 038318 bytes

Data: 0113d8 bytes

Bss: 024d7c bytes

Jumping to load program ~ 20000400

SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]

(C) Copyright 1986 - Silicon Graphics Inc.

real = 4194304

kmem = 561152

user = 3633152

bufs = 819200 (max=16k)

dsd0 not installed

qic0 not installed

sii0 at mbio 0x07200 ipl 5

si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0

si1 not installed

sf0 floppy (80/2/8) slave 2

siq0 at mbio 0x73fc ipl 5

sq0 (qic02 cartridge tape) slave 0

iph0 not installed

tmt0 not installed

ik0 not installed

nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2

fpa installed

lpen not installed

*kernel debugger disabled.
root on si0a
swap on si0b. swplo=0 nswap=64000*

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

#

You are now in SINGLE-USER mode.

With the next step you will create the user partition that you will restore the *usr* filesystem into.

mkfs /dev/rsi0f

*isize = 1920
filsys = /dev/rsi0f, size = 79704: 7680, unused 26
heads = 7, Sectors = 17
cgsiz = 3984:384, firstc = 24, ncgs = 20*

Next you will mount the *usr* filesystem.

mount /dev/si0f /usr

WARNING!! -- mounting: < > as </usr>

The above warning occurs because the new *usr* filesystem (/dev/si0f) does not have a label, you will label it later.

Now change directories to the *usr* directory.

cd /usr

The next commands rewind the bootable tape and then position it forward two files. You are going to load the *usr* file system from the third file on the tape.

```
# mt rew
```

```
# mt fsf 2 (takes about 2 minutes)
```

The next command will restore the *usr* filesystem.

```
# cpio -ivhmd2
```

*if u want Backup
SKIP FSF 2
cpio -ivhmd2*

As each file is read, its pathname is displayed on the screen. The restore should take about 15 minutes (longer for larger systems).

NOTE: If you want to see that the file systems are now installed on the system, perform the following commands. It will show you the standard system is now present. It would also show you any optional products that are installed.

```
# cd /
```

```
# cat /Versions/*
```

<i>root</i>	004-0134-162	GL2-W3.4	Standard System (root)
<i>usr</i> <i>upd</i>	004-0134-172	GL2-W3.4	Standard System (usr)
<i>upd</i>	004-0144-043	GL2-W3.4	Update System
<i>man</i>	004-0154-033	GL2-W3.4	Update manual pages
<i>games</i>	004-0164-033	GL2-W3.4	Update games
<i>demos</i>	004-0174-033	GL2-W3.4	Update demos

*gitat
TROFF FAN NFS XNS*

You are almost done. Now you must restore the disk drive labels.

STEP 12: Restoring Disk Labels.

First, we will restore the disk label using the `sgilabel` command. The following shows the correct syntax:

```
Usage: sgilabel [- n name] [-s serial #] dv#
```

OK, lets label the drive.

```
# sgilabel -n "Release GL2-W3.5" -s 20638 si0
```

My drive serial number was 20638, the name I used indicates the release of software installed (your name can be what you want), and the device was *si0*.

Lets look and see if we really did label the drive. To do this, we again use the *sgilabel* command.

```
# sgilabel /dev/si0
```

```
/dev/si0: Name: Release GL2-W3.5, Serial: 20638
drive: Priam V170, controller: Qualogy 5217
cylinders/heads/sectors(512 byte): 987/7/17
alternate cylinder/# of alt cylinders: 970/17
badtracks=0, interleave=1, trkskw=0, cylskw=11
fs base size
sectors(cylinders)
a: 119( 1), 17850( 150) Root
b: 17969( 151), 17731( 149) Swap
c: 35700( 300), 79730( 670)
d: 119( 1), 115311( 969)
g: 119( 1), 115311( 969)
h: 0( 0), 115430( 970)
```

*look in
owner manual
chapter 6*

Disk config.

2nd Drive

*part in partition G
Y B - as 2nd drive
... 10*

Next, we will restore the filesystem labels using the *labelit* command.

Before you write the labels you must *sync* the disk drive.

```
# sync
```

The following shows the correct syntax for the *labelit* command.

```
Usage: labelit /dev/r???? [ fsname volume [-n]]
```

The question marks designate the file system you want to label. The "-n" option skips the label check before labeling. (I will not use the *-n* option)

Label the *usr* filesystem first.

```
# labelit /dev/rsi0f usr sgi
```

```
Current fsname: , Current volname: , Blocks: 79704, Inodes: 7680  
FS Units: 512b, Date last mounted: Sat June 21 18:00 1986  
New fsname = usr, New volname = sgi  
#
```

Next, label the *root* filesystem.

```
# labelit /dev/rsi0a root sgi
```

```
Current fsname: , Current volname: , Blocks: 17848, Inodes: 7660  
FS Units: 512b, Date last mounted: Sat June 21 18:00 1986  
New fsname = usr, New volname = sgi  
#
```

You must now reboot the system by **Depressing the Reset Button**. This will insure that the labels get updated.

When the prom monitor is entered, boot the system using the normal method.

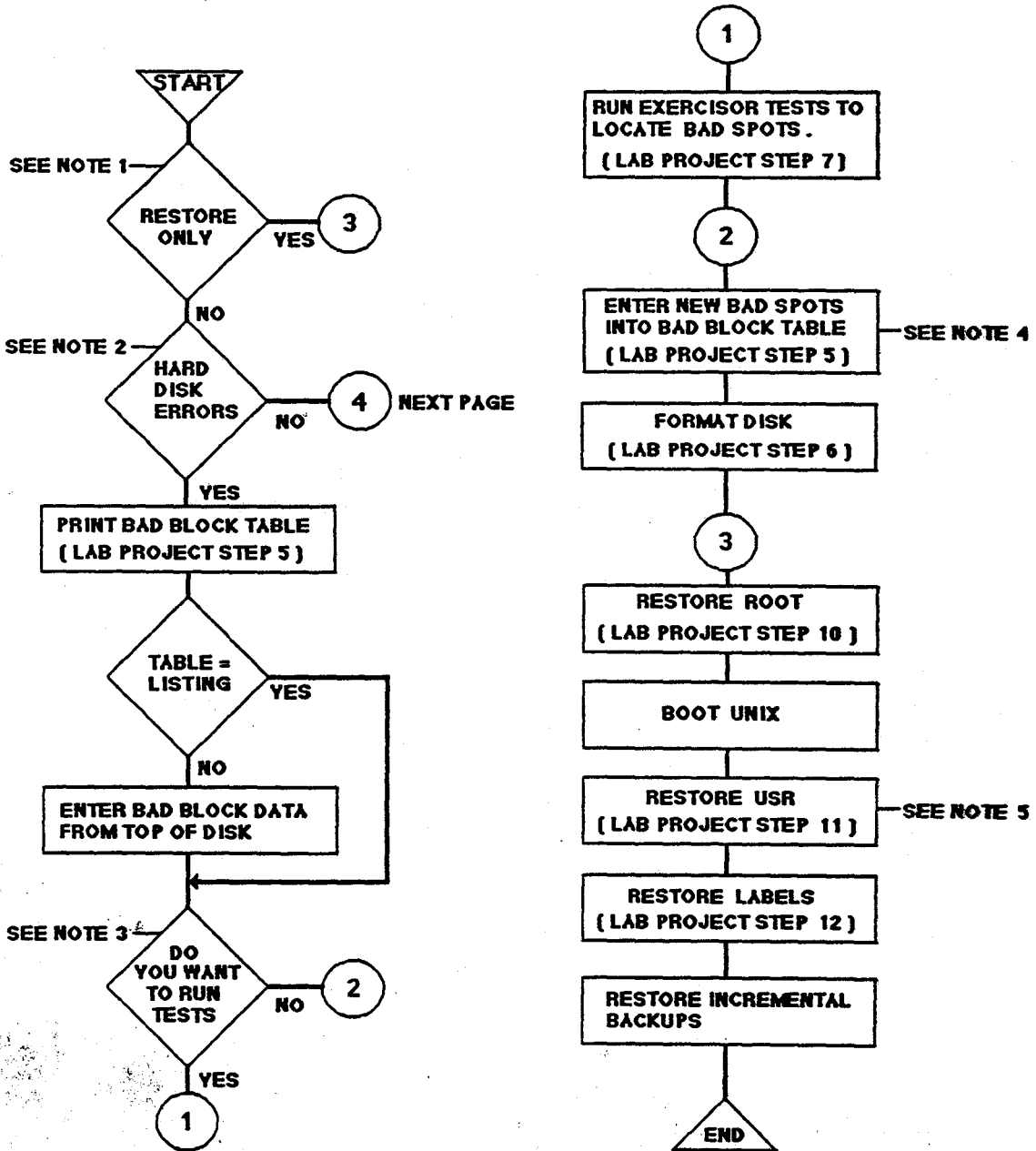
Your customer would now be up and running. The restoring of additional files would be a customer task. You got the system back to the basic system, the customer would then use the backup tapes to restore the full system.

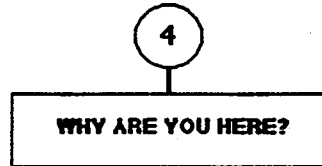
The next section lists only the generic procedure. It is preceded by a simple flow chart to help you determine where to enter the procedure depending on the type of disk problems you are experiencing.

Now, complete the questionnaire and turn it into your instructor.

SECTION 2: Generic Procedure

DISK RESTORATION FLOW CHART



DISK RESTORATION FLOW CHART - PAGE 2**NOTES:**

1. YOU ARE HERE TO ONLY RESTORE ROOT OR USR FILESYSTEMS AND NOT TO ADD NEW BAD SPOT INFORMATION OR FORMAT THE DISK.
2. THIS WOULD BE HARD DISK ERRORS THAT WERE REPORTED BY UNIX.
3. THIS WOULD BE TO VERIFY UNIX REPORTED DISK ERRORS OR EXERCISE THE DISK LOOKING FOR NEW BAD SPOTS.
4. THESE WOULD BE NEW BAD SPOTS REPORTED BY UNIX OR DETECTED WHILE RUNNING THE EXERCISOR TESTS.

YOU MAY WANT TO USE THE ALTERNATE METHOD OF ADDING NEW BAD SPOTS WRITTEN BY CHRIS DUNLAP, SEE DOCUMENT AT END OF THIS LAB PROJECT.

5. THIS COULD COME OFF THE BOOTABLE OR BACKUP TAPE.

TROUBLESHOOTING TIPS:

1. IF YOU ARE GETTING MULTIPLE RANDOM ERRORS, THE PROBLEM IS PROBABLY THE DISK CONTROLLER.
2. YOU CAN NOT MAP OUT HEAD 0, CYL 0 (REPLACE THE DISK IF YOU ARE EXPERIENCING ERRORS HERE)
3. IF RUNNING THE EXERCISOR TESTS AND YOU GET ERRORS STARTING AT HEAD 0, CYL 0, AND THE ERRORS INCREMENT UPWARDS (I.E. 0/0, 0/1, 0/2, 0/3, ETC...), CYCLE THE POWER OF YOUR WORKSTATION AND TRY THE TESTS AGAIN.

This section shows only the actual steps of the lab project. **Remember, use the flow chart to guide you through the correct sequence of steps.**

The IRIS OWNER'S GUIDE, Series 3000, gives an outline of the Tape Restore portion of this procedure.

STEP 1: Reboot the workstation

STEP 2: BOOTING SIFEX FROM TAPE:

1. Insert the bootable tape.
2. Enter " h " for prom monitor command listing.
3. Enter the proper command to boot the FEX program from tape.

STEP 3: Entering Field Engineer portion of SIFEX

4. Enter an upper case Z.
5. Enter the correct password.

STEP 4: SELECTING THE DISK DRIVE

6. Type " s " to set miscellaneous variables.
7. Type " u " to select the drive.
8. Enter " 0 " or " 1 " to select drive to be restored.
9. Type "q" to quit set misc. variables.

STEP 5: ENTER BAD BLOCK DATA:

10. Type " b " to enter bad block edit mode.
11. Type " p " to print the current bad block table.

NOTE:

If the hard copy listing MATCHES the bad block table, execute "q" and goto STEP 6, else execute the following and then goto STEP 6.

- A. Type " c " to clear bad block table.
- B. Type " y " to confirm clearing the table.
- C. Type " a " to add bad block entries.
- D. Now enter each entry from the hard copy listing.
Use the following format: Cylinder/head.
i.e. 31/0, 921/6.

E. Depress <return> without making an entry to terminate the entry process.

12. Type " q " to quit bad block edit mode.

STEP 6: FORMAT DISK:

13. Type " f " to format the disk.

14. Enter " go " to begin the process. (Takes about 8 minutes)

STEP 7: EXERCISE DISK:

15. Type " s " to set misc variables.

16. Type " w " to select write lock toggle function.

17. Enter " off <return> " to turn write lock off.

18. Type " q " to quit set misc variables.

19. Type " e " to enter exercise mode.

20. Type " c " to run the complete read/write test.

21. Depress <return> to loop the test.

22. Depress <return> to test only the selected drive.

NOTE:

One pass takes about 8 minutes. This test should be run at least 12 hours and preferably 24 hours for a thorough check of all the disk, but do what time allows you.

23. Depress " a " to terminate the test.

24. Type " e " to enter error display.

25. Type " d " to display any accumulated errors.

Record any errors on a sheet of paper.

26. Type " q " to quit exercise mode.

NOTE:

If the exercise test DID NOT detect any new errors, then skip to STEP 9.

STEP 8: ADD NEW BAD BLOCKS TO BAD BLOCK TABLE:

A. Type " b " to enter bad block edit mode.

B. Type " a " to add bad blocks.

C. Enter each new bad block: format= cyl/head.

D. Depress <return> without an entry to terminate.

- E. Type " q " to quit bad block edit mode.
- F. Type " e " to enter exerciser routine.
- G. Type " e " to enter error display or reset mode.
- H. Type " r " to reset the error log.
- I. Type " q " to quit exerciser routine.

STEP 9: Re-formatting the Disk Drive.

- 27. Type " f " to format the disk.
- 28. Enter " go " to start the format.

STEP 10: COPY ROOT FILE SYSTEM FROM TAPE TO DISK:

- 29. Type " t " for tape utility program.
- 30. Depress <return> to copy from tape file #2.
- 31. Depress <return> to copy to the selected drive.
- 32. Enter " a " to copy to partition " a ".
- 33. Enter " go " to begin copying.
- 34. Type " q " to quit FEX.
- 35. Type " y " to confirm quit.

STEP 11: COPY /USR FILE SYSTEM FROM TAPE TO DISK:

- 36. Boot UNIX from disk: Enter " b ".
- 37. Enter " mkfs /dev/rsi0f " to create user file partition.
- 38. Enter " mount /dev/si0f /usr " to mount user file system.
- 39. Enter " cd /usr " to change directory to /usr.
- 40. Enter " mt rew " to rewind tape.
- 41. Enter " mt fsf 2 " to skip tape forward two files.
- 42. Enter " cpio -ivhmud2 " to copy in the user file system.

STEP 12: RESTORE DISK DRIVE LABELS:

- 43. Enter the " sgilabel " to get usage of disk label command:
Usage: sgilabel [-n name] [-s serial#] dv#
i.e. sgilabel -n "Release GL2-W3.5" -s 14596 si0
- 44. Enter " sync " to sync the disk drive.
- 45. Label the USER file system first:
i.e. labelit /dev/si0f usr sgi
- 46. Label the ROOT file system:
i.e. labelit /dev/si0a root sgi
- 47. Depress the RESET button to reboot the system.
- 48. Boot up UNIX and go to MULTI-USER to ensure proper operation.

SECTION 3: Review

Student Name:

Please complete the review and hand it into your instructor.

b n c t 0 : s i f e x

b n s i f e x : / s t a n d / s i f e x

1. Where do the FEX programs reside?

i n s t a n d b e c k t a p e

2. Name the three FEX programs and list when you would use each one:

m d f e x m i d i a s - D S D

i p f e x - 4 7 4 ~~s m d~~ s m d - i n t e r f a c e ^{2 1 9 0}

s i f e x - - 3 f o r m a t 1 1

3. List the prom monitor command that you would use to list the headers on the bootable tape:

*L S c t 0 : **

4. What FEX command would you use to enter "Bad Block Edit Mode"?

b

5. If you entered SIFEX to add bad block data that you received from UNIX during normal operation, after entering the cyl/hd information, would you need to format the disk?

y e s

6. If you answered yes to question #5, then answer #6. Why do you need to format the disk if all you are doing is adding a new entry to the already existing table?

To map ~~it~~ ^{The} table on to disk

7. Once UNIX is up, what command do you use to create a new disk partition?

^{RAW}
mkfs /dev/rs10f

Make File System

8. What command is used to label a disk drive?

sgilabel -N "Name" -S SN 510

9. What command is used to label a file system?

Labelit /dev/rs10f usr sgi

10. If you are some where in the FEX program and want to return to the main program (top level fex prompt), what key must you depress?

Del

Mapping Out Bad Tracks on Disk Drives

*D. Christopher Dunlap
Product Support Engineering*

Silicon Graphics Computer Systems, Inc.

October 24, 1986

1. INTRODUCTION

This bulletin describes the procedure for mapping out a bad track. Known media defects are mapped out when the drive is formatted at SGI, but occasionally new errors appear. Sometimes these new errors don't represent actual media defects. This can happen, for example, when the system is interrupted unnaturally while writing to the disk. Although there are other ways of dealing with these types of errors, it's usually easiest to map them out as you would any true media defect as it's often difficult to tell these data errors from defects. The FEX programs (ipfex, mdxfex or sifex) are used to map out bad tracks.

This document assumes you have a general understanding of the Unix operating system, particularly interaction with a shell and the booting and shutdown sequences. Anyone who is not comfortable with these things should not attempt any of the procedures described here. In some of these procedures, mistakes can be fatal to the system.

2. GENERAL PROCEDURE

2.1 The FEX Programs

All versions of FEX are stand-alone programs. They are booted directly from the PROM monitor. Use the "reboot" command to get to the PROM monitor from Unix. You must be the Super-user (root) to use the "reboot" command.

The version of FEX that you will use depends on which disk controller your system has. If the drive you are having problems with has an "md" prefix, such as "/dev/md0c" you use "mdxfex"; if it's got an "si" prefix, such as "si0a", use "sifex"; and if it's got an "ip" prefix, use "ipfex". All the examples here will show "mdxfex". If you are using "ipfex" or "sifex", just substitute the correct name wherever applicable.

Boot FEX by typing "b stand/mdxfex" at the PROM monitor prompt. FEX will return with a prompt, such as "mdxfex 2.4>".

If you want to abort any operation in FEX, hit the key.

If the drive you are having problems with is drive 1, you will have to set the drive unit number, as FEX uses drive 0 by default. Type "s" in response to the "mdxfex 2.4>" prompt, and then type "u" for drive unit number, enter 0 or 1 and hit <return>.

Some versions of FEX have some password security built into them. This was put in to protect users from some of the more dangerous operations in FEX. To check for password security, type a "?" at the "mdxfex 2.4>" prompt. If "Map out a bad track" is listed in the menu, you can skip to the next paragraph. Otherwise, type "Z". FEX should prompt you for a password. The password is "don!" (all lower case). If FEX doesn't accept the password go back to the PROM monitor by typing "q" at the "mdxfex 2.4>" prompt, boot FEX and try it again.

2.2 Information About Your Drive

Later in this procedure, you will need to know how many heads, cylinders, and sectors your drive has. To find this out, boot FEX and enter the Security password if needed. Set the drive unit number

if needed. Type "s" for "Set" in response to the "mdfex 2.4>" prompt, then type "?" to check the possible options. Type the letter indicated to select "Display Settings". Several lines of information will appear. Look for one that starts with "Drive:". This line will list the drive model, the unit number of the drive, and a group of numbers in parenthesis representing the size of the drive.

For example, in the line:

Drive: Hitachi 511-8, Unit=1, (814+9/10/17(512))

"Hitachi 511-8" is the model of the drive, "1" is the drive unit number (always 0 or 1), "814" is the number of user accessible cylinders "9" is the number of cylinders reserved for alternates for bad track mapping, "10" is the number of heads, "17" is the number of sectors per track, and "512" is the number of data bytes per sector. You would need to write down "cylinders=814, heads=10, and sectors=17". Type "q" to get back to the "mdfex 2.4>" prompt.

2.3 Identifying The Bad Track

When you get a hard error on the disk, the error message will list the location of the error. Because of the way the Extent File System (EFS) manages disk space, the location indicated by these errors is not always accurate and complete. See the section titled "Related Issues" below for more details on error messages and the EFS.

In order to get an accurate accounting of the errors on the disk, it is necessary to run a test with the FEX program. This test simply reads the drive a track at a time until it finds an error. It will report the location of the error accurately.

Boot the appropriate FEX program as described above. Type "?" in response to the "mdfex 2.4>" prompt. If selection "e" does not appear, you need to enter the security password as described above.

Enter "e" to get to the "exercises:" prompt. Type "d" for "Disk Read or Write Test". FEX will prompt you with "Repeat How Many Times:" for the number of tracks to read. This is the number of user accessible cylinders times the number of heads. Next, type "r" for "Read", and then enter the number of sectors per track. Type "s" for "sequential reads" and then just hit return for a step size of one track (the number in parenthesis will match the number of sectors per track). Some versions of FEX will start the Read-test now, but others will prompt "Starting at cyl/hd/sec?". If you get this prompt, just type "0/0/0" and hit <return>.

The Read-test will run through until it encounters an error or gets to the end of the test. If it finds an error it will either return to the "exercise:" prompt or it will ask you to "Retry/Skip/Quit". If it returns to the "exercise:" prompt, type "q" to quit and map out the bad track as shown below. Note that you will still need to return to the Read-test to check for additional bad tracks. If FEX prompts to "Retry/Skip/Quit", type "s" for skip and write down each bad track. You may need to "Skip" several times before you reach the end of the test. A disk that passes the Read-test will have a long series of dots followed by the word "Passed".

2.4 Mapping Out a Bad Track

Once you have Determined the locations of the bad tracks, you must map them out so they won't cause problems again in the future.

At the "mdfex 2.4>" prompt, select "m" for "map out a bad track". The FEX program will prompt you for the Cylinder, and then for the Head. It will then try to read the bad track. If it gets errors, it will try several times before either asking you if it should "Retry/Skip/Quit" or giving up on it's own if the error is drastic enough. If given the option, you should type "r" for "Retry" at least a few times as there's a chance that FEX will be able to read the track if you tell it to retry enough.

Eventually, when either FEX succeeds in reading the track, gives up, or you tell it to "skip", it will reformat both the bad track and it's alternate and will return to the "mdfex 2.4>" prompt. If you have any other bad tracks to map out, you should map them out now.

Run the Read-test again to make sure you got all the bad tracks before quitting FEX. Once you are done, type "q" to quit.

2.5 Cleanup

After mapping out bad tracks, "fsck", the file system consistency checker, will usually find lots of inconsistencies. Just answer "y" to all it's questions and it will repair any damage on the file system. This will only restore the integrity of the file system; any files that were lost will have to be restored from tape.

3. RELATED ISSUES

3.1 EFS and Disk Error Reporting

The Extent File System is much more efficient in managing disk space than the Bell file system that was previously shipped on Silicon Graphics systems. Part of this stems from the efficiency of handling file system data as larger chunks called "Extents". An effect of this is that the EFS will often request the driver to read a chunk of data from the disk that extends across the boundaries of two or more physical tracks on the disk. Because of the way the hard disk controllers operate, it's impossible for the drivers to tell for sure where an error occurs when the read has extended across track boundaries in this fashion. The drivers only report the starting point of the error, and how much was read. FEX can read a track at a time, so it avoids this confusion, and always reports errors accurately.

3.2 Bad Track Already Mapped Out

Don't use the "Edit the Bad Track List" facility of FEX for mapping out bad tracks. It doesn't do the same thing as "Map Out a Bad Track". A Typical symptom when "Edit The Bad Track List" has been used is you will get an error while running the FEX Read-test, and then find that "Map Out A Bad Track" claims that the track has already been mapped out. To correct this, Edit the bad track list and delete the problem track from the list. Then proceed to "map out the bad track". Note that the FEX Read-test is the only sure way to correctly verify the error location.

3.3 Hardware Issues

See the document "Disk Drive Compatibility Issues" for more information on disk drive issues.

Contents

LAB 4.4: Creating New User Accounts	1
SECTION 1: Instructional Procedure	2
I. Creating A New User Account	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT.	2
V. PROCEDURE	3
SECTION 2: Generic Procedure	22
SECTION 3: Review	24

LAB 4.4: Creating New User Accounts

This lab project contains three sections:

Section 1: This section is used to teach you how to create a new user account.

Section 2: This section lists just the procedure without all the instructional text.

Section 3: This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. Creating A New User Account

On the typical IRIS workstation, there is a good chance of having at least two other users besides the console if the workstation is on a network or if it is in an environment where more than one person has a need to use it. Whoever is designated the "Administrator" has the responsibility to make sure each user has a properly working environment in which to work. This module covers the operations required to provide that environment.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly list the names of all related system administration files that require modification when adding new accounts, changing the workstation name, changing the *Message-of-the-Day*, and various other administration functions.
- B. Demonstrate that you can change the name of your workstation.
- C. Demonstrate that you can create a new user account and set the correct user protections and permissions for that account.

III. PURPOSE

Teach the student how to add new accounts to the workstation. The project seeks to develop a basic skill in each student that will help the student to understand some of the basic functions that the system administrator must perform, and it allows each student sufficient time to practice some of these skills.

IV. EQUIPMENT.

A functional IRIS 68020 based workstation.

V. PROCEDURE

Two students will be assigned one IRIS workstation. The two students will work as a team with one student making the actual command entries at the primary console.

You will create a new account; the account name created will be the same as one of the student's last name.

It is **very important** that you follow the lab project precisely and **do not** wander off in your own direction. You will have ample time for *free-time* later in the course.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

- | | |
|------------------------|---|
| 1. iris > | The prom monitor. |
| 2. JULIE n > | Student account prompt when in Multi-user mode, where n = command number. |
| 3. JULIE n # | Student account prompt when in Super-user mode, where n = command number. |
| 4. # | Single user mode. |
| 5. JULIE login: | This prompt appears when you go into Multi-user mode from Single-user. |

NOTE: Some of these prompts will change as a result of some of the functions you perform during this project.

The following text will lead you through the execution of several commands. When you are required to enter a command, the text recreates the **STANDARD OUTPUT** for that command. Just keep reading and following the flow of the lab project.

Also throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **# mkdir /usr/people/smith**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in **normal print**.

BEGIN ACTUAL PROCEDURE HERE:**STEP 1: Power on the workstation.**

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985  
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)  
Configuration Switch: 0x0000  
Multibus Window (2mb) at Megabytes 0 and 1.  
Multibus accessible memory (1mb) begins  
at Physical memory page 300,  
at Virtual address 2000000.  
iris>
```

STEP 2: Boot the UNIX System.

```
iris > b
```

```
SGI Extent Filesystem  
Loading: md:0:defaultboot  
Text: 038318 bytes  
Data: 0113d8 bytes  
Bss: 024d7c bytes  
Jumping to load program ~ 20000400
```

```
SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]  
(C) Copyright 1986 - Silicon Graphics Inc.  
real = 4194304
```

LAB 4.4: Creating New User Accounts

```
kmem = 561152
user = 3633152
bufs = 819200 (max=16k)
dsd0 not installed
qic0 not installed
sii0 at mbio 0x07200 ipl 5
si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0
si1 not installed
sf0 floppy (80/2/8) slave 2
siq0 at mbio 0x73fc ipl 5
sq0 (qic02 cartridge tape) slave 0
iph0 not installed
tmt0 not installed
ik0 not installed
nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2
fpa installed
lpen not installed
kernel debugger disabled.
root on si0a
swap on si0b. swplo=0 nswap=64000
```

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

#

STEP 3: Go to multi-user mode.

multi

#

INIT: New run level: 2

Is the date [day month date time time-zone year] correct? (y or n) y

Do you want to check filesystem consistency? (y or n) y

Checking file systems for consistency:

/dev/si0a

File system: root Volume: SGI

*** Phase 1 - Check Blocks and Sizes*

*** Phase 2 - Check Pathnames*

*** Phase 3 - Check Connectivity*

*** Phase 4 - Check Reference Counts*

*** Phase 5 - Check Free List*

nnn files nnnnn K used nnnn K free

/dev/si0f

File system: usr Volume: SGI

*** Phase 1 - Check Blocks and Sizes*

*** Phase 2 - Check Pathnames*

*** Phase 3 - Check Connectivity*

*** Phase 4 - Check Reference Counts*

*** Phase 5 - Check Free List*

nnn files nnnnn K used nnnn K free

Mounting: /usr

Preserved editor files

Cleared /tmp

Resetting locks and logs

Hostname: JULIE

Daemons:

update

cron

xnsd

lpd

lpsched

Daemons started

JULIE login:

STEP 4: Login as Root.

When performing system administration functions you should be a *Super-User* or logged in as **ROOT**.

JULIE login: root

Step 5: Naming the IRIS workstation.

The default name of an IRIS workstation is *IRIS*. The name of all the workstations in the classroom are *JULIE*. If you have more than one workstation on a network, you must assign each workstation a unique name. The name can be up to *eight* characters long and must contain no blanks.

Before you create a new user account you will perform the following three functions:

1. Change the name of your workstation.
2. Insure that the new name is displayed at login time.
3. Change your workstation login welcome message.

Step 6: Changing your workstation name.

The name of the workstation is established by the character string that is held in the file */etc/sys_id*. Lets go look at the contents of this file.

```
# cd etc
# more sys_id
```

JULIE

As you can see, the name of your workstation is *JULIE*. Next look at a file called *sys_idDEFAULT*. This file will not exist on a customer system. It is a copy of the *sys_id* file as it comes from the factory, and it is here for illustration only.

```
# more sys_idDEFAULT
```

IRIS

You will now change the name of the workstation by editing the *sys_id* file using the **vi** editor. Change the name to some name that you can both agree upon.

To feel the results of the change you must then *re-boot* the system, but before you re-boot, I will also have you edit a file called */etc/gettydefs* to insure that the workstation name is displayed when you login.

Step 7: Changing the *gettydefs* file.

This is the file that is accessed when you login. The */etc/inittab* file defines what line to access first for each of the terminal types: Console (*co*), ASCII on one of the ports (*dx*), or dial in (*du*).

The information on each line defines the baud rate, the message to be displayed, and a pointer to the next baud rate to attempt if this rate fails. (remember, depressing the *break* key forces the *getty* process to fetch the next entry in this file if you receive garbage at your login terminal.)

```

# more gettydefs
co_9600# B9600 # B9600 SANE TAB3 #\r\n\r\nJULIE login: #co_4800
co_4800# B4800 # B4800 SANE TAB3 #\r\n\r\nJULIE login: #co_2400
co_2400# B2400 # B2400 SANE TAB3 #\r\n\r\nJULIE login: #co_1200
co_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #co_300
co_300# B300 # B300 SANE TAB3 #\r\n\r\nJULIE login: #co_9600
dx_9600# B9600 # B9600 SANE TAB3 #\r\n\r\nJULIE login: #dx_4800
dx_4800# B4800 # B4800 SANE TAB3 #\r\n\r\nJULIE login: #dx_2400
dx_2400# B2400 # B2400 SANE TAB3 #\r\n\r\nJULIE login: #dx_1200
dx_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #dx_9600
du_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #du_300
du_300# B300 # B300 SANE TAB3 #\r\n\r\nJULIE login: #du_1200
    
```

Handwritten annotations:

- console* (next to co_9600)
- Initial Flags* (next to B9600)
- Band Rate* (next to # B9600)
- Final Flag* (next to SANE)
- Port Parameters* (next to TAB3)
- Action* (next to #\r\n\r\n)
- NEWLINE* (next to \r\n\r\n)
- IF term fixed* (next to JULIE)
- Next Line* (next to login:)
- Point to next line when break is pressed.* (next to #co_4800)
- ASCII Terminal* (bracketed next to dx_9600 to dx_1200)
- Dial Terminal* (bracketed next to du_1200 to du_300)

Using the *vi* editor, change the login in prompt, *JULIE login:*, to the new name of the workstation established in *step 6*. Change it on every line in the file. This will insure that the login prompt will display the name of the workstation.

Step 8: Changing the *Message-of-the-Day*.

Lets first look at the current message.

```
# more motd
```

```
Welcome to the world of the  
IRIS personal workstation.
```

```
The SGI education department  
hopes that you have a very  
pleasant and informative stay  
with us.
```

Now, using vi, change your message to what ever you want it to say.

Step 9: Re-booting the system.

Execute the following commands in the order shown. The output produced by each command is not shown since you have seen it many times before.

```
# reboot
```

```
iris > b
```

```
# multi
```

```
NEWNAME login: root
```

As you booted up the system and logged into ROOT you should have noticed the changes that you made.

You are now going to create a new account, using your last name as the new account name.

Step 10: Adding a new account name to the */etc/passwd* file.

```
# cd etc
```

account

more passwd

```

root:Quu1RjNscs85s:0:0:Superuser:/:bin/csh
rootcsh::0:0:Superuser:/:bin/csh
rootsh::0:0:Superuser:/:bin/sh
daemon:*:1:1:/:
bin:*:2:2:Binary Files:/:
uucp:*:3:5:UUCP Login Account:/usr/spool/uucppublic:/usr/lib/uucp/uucico
adm:*:5:3:Administration:/usr/adm:
uucpadmin:*:8:8:UUCP Administration:/usr/lib/uucp:
lp:*:9:9:Line Printer:/:
tutor::993:997:/:usr/people/tutorial/c.graphics:/bin/csh
mexdemos::996:997:/:usr/people/mexdemos:/bin/csh
demos::997:997:/:usr/people/demos:/bin/csh
guest::998:998:/:usr/people/guest:/bin/csh
games:*:999:999:Games:/usr/games:/bin/sh
rich:g1tF9GTITysy2:10:20:Richard Houston:/usr/people/rich:/bin/csh
student:iR5.B3iwOfdXl:11:20:class students:/usr/people/student:/bin/csh
student1:w2gmv/cAnjkwI:12:20:class students:/usr/people/student1:/bin/csh

```

The first step required to create a new account will be adding the new account name to the *passwd* file, but before we do this lets talk about the *passwd* file.

The */etc/passwd* file:

- Determines who can log into the system and who can't.
- Contains line entries for each user "account"
- Contains special accounts reserved for exclusive UNIX System purposes.
- Is readable by everyone. Passwords are encrypted to maintain security.

The IRIS workstation is shipped with six user accounts:

root, *rootcsh*, *rootsh* - Super-user accounts with a *csh* or *sh* environments.

guest - Both a sample user account and a real guest account using a *csh* environment.

demos, *mexdemos* - Regular and Window Manager demo accounts

LAB 4.4: Creating New User Accounts

Special function accounts are used by background UNIX processes while the system is running. You can not log into these accounts.

- Daemon** This entry is used to indicate to daemons a starting path of root.
- bin** Bin also points to root, and is also used to give a reference to a path in which the directory bin is found.
- uucp** This account is used whenever the system handles uucp requests. It points to the general uucp directories.
- uucpadmin** Points to the administration files for the uucp handler.
- lp** Generically, lp points to /dev, relative to root as shown.
- adm** Administrative tasks use this account to find the administration directories

In addition to the pre-defined accounts, each user on an IRIS workstation is given his or her own unique account (rich, student, student1, tutor, etc). This is what you are going to add to the file; a new user account.

The file */etc/passwd* contains a line for each account on a UNIX system. Each line has seven fields separated by colons (:).

on previous page.

	FIELD	CONTENTS
	1	Account name - login names
	2	Encrypted user password - * MEANS NO PASSWORD
	3	User numerical ID - NO LOGIN
<i>SYSTEM</i>	4	Group numerical ID
<i>admin</i>	5	Users real name
<i>1001 #</i>	6	Home directory (default /)
<i>/etc/group</i>	7	Login shell (default is /bin/sh) - Choose csh - Berkeley or SH Bourne
<i>File Fa Group</i>		
<i>Pass Wd.</i>		

The following gives a description of each field:

● **Account Name**

The *account name* is the login name of the new account. This field is searched for by the login program when the user logs in. Any combination of letters, numbers, and special characters (like *_*) are valid. Generally, account names are under 14 characters in length and are determined by the user for ease of memorization.

- **Password**

The second field is the *encrypted password* for that user. For new accounts and newly shipped workstations, this field will be empty, ie, nothing between the colon delimiters (::). Users should be made aware that security depends on the complexity of the password they choose. Normally users should be required to have a password of at least 6 characters (`passwd(1)` will balk over passwords under 6 characters) and have a mix of letters, numbers, and special characters. They should NEVER use their names, initials, or commonly known acronyms and nick-names.

The (*) seen in some of the system names (sometimes "dummy") guarantees that no password attempt can log in except a process owned by root. There will be no match because any entered attempt will be encrypted into a 13 character password before comparison to whatever string is in the password field. There cannot be a match.

Password aging is a method to force users to change their password every so often for security purposes. It is generally considered an aggravation, but is important in some environments. Aging is in effect if the encrypted password is followed immediately by a comma and a non-null string of characters. See the `passwd(4)` section of the UNIX Programmer's Reference Manual.

- **User Numerical ID**

The *User number* represents a number that is assigned to each file the user creates. This establishes the ownership of a given file. If two users have the same user number, the system has no way to distinguish them for purposes of directory and file protection. In general, the ownership numbers are sequential, but do not have to be. The key is having a unique number for each user. Numbers can go to 999. Owner 0 is always root or super-user and should never be assigned to a normal account.

- **Group Numerical ID**

The group number is the group identification given to the user at login. When the user logs in, he or she has access to a pool of directories and files shared by all users in that group. When setting up a new account, provide the group number corresponding to whatever generic group this account should belong to. On the IRIS, a default group of "users" is designated as group 20. More on this later.

- **Users Name**

Field 5 is used as an information field for any human reader. In this field can be placed the Users name, or a comment or nothing.

- **Initial Working Directory**

This field contains the account's default directory. When logged in, the user will be placed in this directory and it will generally be assigned as the users "home" directory. Note that in the above example, the guest account is passed to `/usr/people/guest`. That directory will contain the `.login` or `.profile` to set up the environment a guest would use. The default is `/` (root).

- **Program to use as Shell**

Normally a user account will have a shell (`/bin/sh` or `/bin/csh`) in this field. For special accounts, any program can be located here that will execute when the account name is referenced. A pseudo-shell could be created to severely limit access to specific functions for a security account. The default is `/bin/sh`.

Using `vi`, add a new line to the `passwd` file. The account name will be the last name of one of the students. Following is an example of what you will add.

```
your.last.name::13:20:your.full.name:/usr/people/your.last.name:/bin/csh
```

i.e.

```
martin::13:20:Dennis Martin:/usr/people/martin:/bin/csh
```

<i>martin</i>	This is the account name.
<i>13</i>	This is the next sequential user number.
<i>20</i>	This account has been made a part of the <i>user group</i> which has the group ID number of 20.
<i>Dennis Martin</i>	This is the real name of the owner of the account. Note: The account name did not have to be his last name, it could have been anything.
<i>/usr/people/martin</i>	This is the pathname to the account. It lives under the <i>people</i> directory.
<i>/bin/csh</i>	This selects the <i>shell</i> that the user wants to use.

NOTE:

The name *martin* will be used for illustration throughout this project. You will substitute your name in place of *martin* at the appropriate spots.

Step 11: Setting up the new user directory.

Once the `/etc/passwd` file has been edited, the Administrator should set up the new users account directory.

Create the new user directory using the following command:

```
# mkdir /usr/people/martin
```

This is the place to which the `passwd` file points when *martin* logs in.

Step 12: Copying `.login` and `.profile` or `.cshrc` files.

The new directory only contains (`.`) and (`..`), providing no environment for the new user. The IRIS workstation has default `.login`, `.profile`, and `.cshrc` files available in root that you can copy over, or you can use a copy of your own. Enter the new account directory and copy in the appropriate file(s).

NOTE:

We will use `.login` and `.cshrc` since we are going to be using the `csh` shell in the new account.

```
# cd /usr/people/martin
```

```
# cp ../login .login
```

```
# cp ../cshrc .cshrc
```

```
# ls -a
```

```
.  ..  .cshrc  .login
```

If you wanted to use the `sh` shell, then you would only copy over the `.profile` file.

We will look at the contents of these files later in the project.

Step 13: Changing Protections and Ownerships of the new account.

Once all the appropriate files have been copied into the directory, you must set the proper ownerships and protections. Remember, because you are *su* or root, everything you have created so far is owned by root, therefore *martin* would not be able to use his directory!

Lets first look at the contents of our new account directory using the *long list* format of the *ls* command.

```
# ls -al
```

```
total 4
drwxrwxrwx 2 root sys 80 Jul 31 10:23 .
drwxr-xr-x 11 bin bin 176 Jul 31 10:02 ..
-rw-r--r-- 1 root sys 41 Jul 31 10:03 .cshrc
-rw-r--r-- 1 root sys 144 Jul 31 10:03 .login
```

As you can see, *martin* owns nothing.

The following sequence of commands will change the ownership of the two new files you just copied into the directory.

```
# chown martin .login
```

```
# chown martin .cshrc
```

```
# ls -al
```

```
total 4
drwxrwxrwx 2 root sys 80 Jul 31 10:23 .
drwxr-xr-x 11 bin bin 176 Jul 31 10:02 ..
-rw-r--r-- 1 martin sys 41 Jul 31 10:03 .cshrc
-rw-r--r-- 1 martin sys 144 Jul 31 10:03 .login
```

Make these user

Next, you must change the ownership of the directory to *martin*.

```
# cd ..
```

```
# chown martin martin
```


Lets look at the change...

```
# ls -al
```

```
total 10
drwxr-xr-x 11 bin bin 192 Jul 31 15:38 .
drwxrwxrwx 19 root sys 304 Jul 28 09:36 ..
drwxr-xr-x 3 demos demos 336 Jul 28 09:32 demos
drwxr-xr-x 8 guest guest 128 Jul 28 09:33 gifts
drwxr-xr-x 3 guest guest 112 Jul 28 09:35 guest
drwxrwxrwx 2 martin sys 64 Jul 31 15:21 martin
drwxr-xr-x 6 mexdemos demos 512 Jul 28 09:35 mexdemos
drwxrwxrwx 4 root sys 192 Jul 29 08:38 rich
drwxr-xr-x 3 student user 128 Jul 28 09:36 student
drwxr-xr-x 2 student1 user 144 Jul 28 09:36 student1
drwxrwxrwx 3 tutor demos 48 Nov 12 14:00 tutorial
```

Initial protection can be set up by the administrator for the user. Notice that *martin* has a permission code of 777. The following command will change the permissions.

```
# chmod 755 martin
```

```
# ls -al
```

```
total 10
drwxr-xr-x 11 bin bin 192 Jul 31 15:38 .
drwxrwxrwx 19 root sys 304 Jul 28 09:36 ..
drwxr-xr-x 3 demos demos 336 Jul 28 09:32 demos
drwxr-xr-x 8 guest guest 128 Jul 28 09:33 gifts
drwxr-xr-x 3 guest guest 112 Jul 28 09:35 guest
drwxr-xr-x 2 martin sys 64 Jul 31 15:21 martin
drwxr-xr-x 6 mexdemos demos 512 Jul 28 09:35 mexdemos
drwxrwxrwx 4 root sys 192 Jul 29 08:38 rich
drwxr-xr-x 3 student user 128 Jul 28 09:36 student
drwxr-xr-x 2 student1 user 144 Jul 28 09:36 student1
drwxrwxrwx 3 tutor demos 48 Nov 12 14:00 tutorial
```

You set read, write, and execute for the owner (*martin*), read and execute only permission to everyone else. The new user (*martin*) should configure his *.login* or *.profile* file to set up an *umask* command (see *cs*h or *sh*) for individual file and directory protection.

Step 14: Setting up Group relationships.

The group number assigned in the passwd entry for each user corresponds to groups defined in the file `/etc/group` (see `group(4)`).

`/etc/group` format:

There are three fields:

- The name of the group
- An encrypted password (just like the `/etc/passwd` file)
- Numerical group ID.

*not exactly
* means no password*

Lets go look at the `/etc/group` file as it is when shipped.

```
# cd /etc
# more group
```

```
sys:*:0:
system:*:0:
staff:*:0:
daemon:*:1:
bin:*:2:
adm:*:3:
sgi_use:*:4:
uucp:*:5:uucp
sgi_use:*:6:
sgi_use:*:7:
uucpadm:*:8:uucp
lp:*:9:
sgi_use:*:10:
sgi_use:*:11:
sgi_use:*:12:
sgi_use:*:13:
sgi_use:*:14:
sgi_use:*:15:
sgi_use:*:16:
sgi_use:*:17:
sgi_use:*:18:
sgi_use:*:19:
```

```

user:*:20:
demos:*:997:
guest:*:998:
games:*:999:

```

Notice that the *user* group ID is set to 20. This is the default and the value you entered into field 4 of the line you added to the passwd file.

Martin is a member of the user group.

NOTE:

The following information is for general knowledge. It assumes a hypothetical system with the */etc/group* file set as shown by the following listing. It shows you how an account can be assigned to more than one group.

```

GROUP sys:*:0:
system:*:0:
staff:*:0:
daemon:*:1:
bin:*:2:
adm:*:3:
sgi_use:*:4:
uucp:*:5:uucp
sgi_use:*:6:
sgi_use:*:7:
uucpadm:*:8:uucp
lp:*:9:
sgi_use:*:10:
sgi_use:*:11:
sgi_use:*:12:
sgi_use:*:13:
sgi_use:*:14:
sgi_use:*:15:
sgi_use:*:16:
sgi_use:*:17:
sgi_use:*:18:
sgi_use:*:19:
user:*:20:
wimps:*:21:
bigguys:*:22:
football:*:22:wimps,bigguys
nerds:*:30:martin,rich,mzl
demos:*:997:
guest:*:998:

```

Handwritten annotations:

- GROUP* (written vertically next to the first line)
- passwd* (written above the first line)
- Group Name* (written above the first line)
- ID* (written next to the line *lp:*:9:*)
- Sub group user accounts* (written above the lines *wimps:*:21:* and *bigguys:*:22:*)

```
games*:999:
```

There are two ways to add users to a group. The first is by the `/etc/passwd` group ID field. If *martin* was assigned to group 22, he would be assigned to the group "bigguys" (see the above file listing). If you also wanted to assign him to the group "football", you can add his group to the football group (ostensibly to do in the "wimps" group) or simply add his account name to the group as exemplified by the group "nerds."

To use his group affiliation, the user executes the command `newgroup(1)`. Su can setup the group password by entering into the group and executing the `passwd(1)` command.

The file (`/etc/group`) is readable to all so it can be used as a resource to anyone and because passwords are encrypted. If a password is placed in the second field, anyone attempting to access a file shared by that group would have to enter a password first. If nothing is in this field, no password is requested. `*` permits any user of that group to `newgroup` into it without a password. If the user is not part of that group, he is prompted for a password, but none will match the `*`.

What you want to do now, is change the group that your new directory belongs to.

Lets again change directories to the new directory.

```
# cd /usr/people
```

```
# ls -al
```

```
total 10
drwxr-xr-x 11 bin      bin      192 Jul 31 15:38 .
drwxrwxrwx 19 root      sys      304 Jul 28 09:36 ..
drwxr-xr-x  3 demos    demos    336 Jul 28 09:32 demos
drwxr-xr-x  8 guest     guest    128 Jul 28 09:33 gifts
drwxr-xr-x  3 guest     guest    112 Jul 28 09:35 guest
drwxr-xr-x  2 martin   sys      64   Jul 31 15:21 martin
drwxr-xr-x  6 mexdemos demos    512 Jul 28 09:35 mexdemos
drwxrwxrwx  4 root      sys      192 Jul 29 08:38 rich
drwxr-xr-x  3 student  user     128 Jul 28 09:36 student
drwxr-xr-x  2 student1 user     144 Jul 28 09:36 student1
drwxrwxrwx  3 tutor    demos    48   Nov 12 14:00 tutorial
```

Notice that *martin* belongs to the `sys` group. This is the way it was created. Lets change it...

```
# chgrp user martin
```

ls -al

total 10

```

drwxr-xr-x  11 bin          bin          192 Jul 31 15:38 .
drwxrwxrwx  19 root        sys          304 Jul 28 09:36 ..
drwxr-xr-x   3 demos      demos       336 Jul 28 09:32 demos
drwxr-xr-x   8 guest      guest       128 Jul 28 09:33 gifts
drwxr-xr-x   3 guest      guest       112 Jul 28 09:35 guest
drwxr-xr-x   2 martin    user         64 Jul 31 15:21 martin
drwxr-xr-x   6 mexdemos  demos       512 Jul 28 09:35 mexdemos
drwxrwxrwx   4 root        sys          192 Jul 29 08:38 rich
drwxr-xr-x   3 student   user        128 Jul 28 09:36 student
drwxr-xr-x   2 student1  user        144 Jul 28 09:36 student1
drwxrwxrwx   3 tutor      demos        48 Nov 12 14:00 tutorial

```

Your account is now set up where you own it. When you create any new files using the editor, those files will also belong to you. Any new directories you create under your top directory will also belong to you.

Step 15: Setting up your account environment.

The `.login` and `.profile` files have the following default formats.

`.login:`

make these

```

setenv SHELL /bin/csh
stty erase ^H kill ^U intr ^C echoe
set ignoreeof noglob
set tmp = ('tset -S -Q')
setenv TERM $tmp[1]
unset noglob

```

access → /etc/passwd

`.profile:`

```

SHELL=/bin/sh
TZ='cat /etc/TZ'
export SHELL TZ

```

```

stty erase ^H kill ^U intr ^C echoe
eval 'tset -S -Q' ; TERMCAP=

```

It is generally left to the user to add detail to his/her `.login` or `.profile` files to setup their environment. Details can be gleaned from the description found in `cs(1)` and `sh(1)`, or by looking at other users

Remember, it is these files where *alias* commands are placed, your *search rule* is expanded, special programs you want run when you login, etc.

A Final Note:

Once the administrator has performed the above, the user will generally be able to do his/her thing without further difficulty. If the user is going to be using a remote terminal, printer, and/or dial in via modem, there are a few more things to do before the user is properly set up on the system. Each of these things will be covered by the next lab project.

You should now logout of ROOT and login to your new account. Try changing your account prompts by editing `.cshrc` and adding the lines of code that control the prompts (see `/usr/people/student/.cshrc`).

Create a new sub-directory in your account and using `vi`, create a new file. Look at the ownership of the new directory and file to verify that they belong to you.

When you are satisfied that you can create a new account, answer the questions in section 3 and hand them into your instructor.

SECTION 2: Generic Procedure

This section shows only the actual steps required to add a new user account to the workstation.

The IRIS OWNER'S GUIDE, Series 3000, gives an outline of this procedure. This lab project uses a slightly different approach, but the results are the same.

The numbered steps of this section are different than section 1, because, the steps showing the boot process are not listed.

Naming the IRIS workstation

STEP 1: Become Super-user or login as ROOT.

STEP 2: Using vi, edit */etc/sys_id* and change name.

STEP 3: Edit */etc/gettydefs*, changing each line prompt name to the new workstation name.

STEP 4: Reboot the system.

Adding a new account to the system

STEP 1: Edit */etc/passwd*, add line for the new account.

STEP 2: Create the new user directory

```
mkdir /usr/people/new.name
```

STEP 3: Copy *.login* and *.cshrc* or *.profile* into the new directory.

STEP 4: Change the ownership of the environmental files just copied.

STEP 5: Change the ownership of the new directory.

STEP 6: Change the permissions of the new directory.

STEP 7: Edit */etc/group*, adding user to other desired groups.

STEP 8: Change the group affiliation of the new account to desired group.

STEP 9: Modify environmental files to fit your needs.

SECTION 3: Review

Student Name:

Please complete the review and hand it into your instructor.

1. What file do you edit to change the workstation name?

sys_id

2. What file do you edit to insure that the workstation name is displayed at login time?

gottydefs

3. What file do you edit to add a new account name to your system?

/etc/passwd

4. How many fields are there in each entry of the */etc/passwd* file?

7

5. What does field 3 (of a *passwd* entry) define?

user id

6. Do user ID numbers have to be sequential?

No

7. Can two accounts have the same user ID number?

should not

8. You want a new account to use the *C-shell*. Show the correct statement that you must place into field 7 of the *passwd* entry.

1/bin/csh

9. What files define an accounts environmentals?

login . CShell

10. What is the command that you use to change file or directory ownerships?

chown user Files

11. What is the command that you use to change permissions?

chmod 755 File a Dir

12. What is the file that defines what group or groups a user will belong to?

/etc/group /etc/passwd ↳ Group ID

13. Can a user belong to more than one group? What file defines this?

/etc/group

14. What is the command that you use to change the group affiliation of a file or directory?

chgrp group name or SD# File

Contents

LAB 4.5: Adding ASCII Devices	1
SECTION 1: Instructional Procedure	2
I. Adding an ASCII terminal	2
II. OBJECTIVE	2
III. PURPOSE	2
IV. EQUIPMENT	3
V. PROCEDURE	3
SECTION 2: Adding an ASCII Printer	19
SECTION 3: Generic Procedures	27
SECTION 4: Review	30

LAB 4.5: Adding ASCII Devices

This lab project contains four sections:

Section 1: This section is used to teach you how to add an ASCII terminal to your workstation.

Section 2: This section is used to teach you how to add an ASCII printer to your workstation.

Section 3: This section lists just the procedures without all the instructional text.

Section 4: This is a questionnaire that you will complete and hand in after performing the lab project.

SECTION 1: Instructional Procedure

I. Adding an ASCII terminal

The IRIS provides RS232 and network access to itself through the three ports located on the rear of the machine, and through 32 XNS network or 8 TCP ports. Theoretically, a given IRIS can support 35 concurrent users.

Terminal support is provided through a chain of software events as summarized below:

- *init* generates *gettys* on each port allocated for terminals.
- *getty* generates the login prompt at a baudrate specified in */etc/gettydefs*. When the user gives his/her login id, *getty* passes it to and initiates *login*.
- *login* prompts for the password, initiates a number of system variables and starts a shell specified in */etc/passwd*.

Adding a new terminal requires the editing of several system files to properly handle the remote terminal.

II. OBJECTIVE

When you finish this lab project you should be able to:

- A. Correctly list the names of all the related system administration files that require modification when adding an ASCII terminal or printer.
- B. Demonstrate that you can add an ASCII terminal to your workstation and correctly modify all the related files.
- C. Demonstrate that you can add an ASCII printer to your workstation and correctly modify all the related files.

III. PURPOSE

Teach the student how to add ASCII terminals and printers to the workstation. The project seeks to develop a basic skill in each student that will help the student to understand the functions required when adding an ASCII terminal or printer and other options or devices that require

similar tasks. Each student is allowed sufficient time to practice some of these skills.

IV. EQUIPMENT.

A functional IRIS 68020 based workstation, an ASCII terminal and printer.

V. PROCEDURE

Two students will be assigned one IRIS workstation. The two students will work as a team with one student making the actual command entries at the primary console.

It is very important that you follow the lab project precisely and do not wander off in your own direction. You will have ample time for *free-time* later in the course.

During the project you will see five different prompts; the current prompt is determined by where you are in the workstation systems.

1. iris > The prom monitor.
2. JULIE n > Student account prompt when in Multi-user mode, where n = command number.
3. JULIE n # Student account prompt when in Super-user mode, where n = command number.
4. # Single user mode.
5. JULIE login: This prompt appears when you go into Multi-user mode from Single-user.

NOTE: Some of these prompts may be different as a result of some of the functions you performed during the last lab project.

The following text will lead you through the execution of several commands. When you are required to enter a command, the text recreates the STANDARD OUTPUT for that command. Just keep reading and following the flow of the lab project.

Also throughout the text are comments that explain what you are doing or what is about to happen as a result of executing a command.

- The commands that you are required to enter are indented and **HIGHLIGHTED USING BOLD** print next to the prompt that would be displayed.

i.e. **#telinit q**

- The **stdout** of each command or any terminal output that would result from something that you are required to do, will be beneath the command, left justified and **HIGHLIGHTED USING ITALICS**.
- The instructional comments that makeup the bulk of the lab project will be in **normal print**.

BEGIN ACTUAL PROCEDURE HERE:

STEP 1: Power on the workstation.

Power on your IRIS workstation using the front panel switch. You should see the initial PROM monitor message appear at your terminal:

```
IRIS (IP2 - Revision B) Monitor Version 3.0.7 December 20, 1985  
Memory Size 4mb (Physical Map (1mb/bit) 0x0000000f)  
Configuration Switch: 0x0000  
Multibus Window (2mb) at Megabytes 0 and 1.  
Multibus accessible memory (1mb) begins  
at Physical memory page 300,  
at Virtual address 2000000.  
iris>
```

STEP 2: Boot the UNIX System.

```
iris > b
```

SGI Extent Filesystem
Loading: md:0:defaultboot
Text: 038318 bytes
Data: 0113d8 bytes
Bss: 024d7c bytes
Jumping to load program ~ 20000400

SYSTEM 5 UNIX #0 [Wed May 7 04:49:59 PDT 1986]
(C) Copyright 1986 - Silicon Graphics Inc.
real = 4194304
kmem = 561152
user = 3633152
bufs = 819200 (max=16k)
dsd0 not installed
qic0 not installed
sii0 at mbio 0x07200 ipl 5
si0 (Hitachi 512-17 name: Hitachi 512-17) slave 0
si1 not installed
sf0 floppy (80/2/8) slave 2
siq0 at mbio 0x73fc ipl 5
sq0 (qic02 cartridge tape) slave 0
iph0 not installed
tmt0 not installed
ik0 not installed
nx0 (FW 2.5 HW 4.0) (0800.1400.3948) at mbio 0x7ffc ipl 2
fpa installed
lpen not installed
kernel debugger disabled.
root on si0a
swap on si0b. swplo=0 nswap=64000

INIT: SINGLE USER MODE

*Welcome to the world of the
IRIS personal workstation.*

*The SGI education department
hopes that you have a very
pleasant and informative stay
with us.*

#

STEP 3: Go to multi-user mode.

multi

INIT: New run level: 2

Is the date [day month date time time-zone year] correct? (y or n) y

Do you want to check filesystem consistency? (y or n) y

Checking file systems for consistency:

/dev/si0a

File system: root Volume: SGI

**** Phase 1 - Check Blocks and Sizes**

**** Phase 2 - Check Pathnames**

**** Phase 3 - Check Connectivity**

**** Phase 4 - Check Reference Counts**

**** Phase 5 - Check Free List**

nnn files nnnnn K used nnnn K free

/dev/si0f

File system: usr Volume: SGI

**** Phase 1 - Check Blocks and Sizes**

**** Phase 2 - Check Pathnames**

**** Phase 3 - Check Connectivity**

**** Phase 4 - Check Reference Counts**

**** Phase 5 - Check Free List**

nnn files nnnnn K used nnnn K free

Mounting: /usr

Preserved editor files

Cleared /tmp

Resetting locks and logs

Hostname: JULIE

Daemons:

update

cron

xnsd

lpd

lpsched

Daemons started

JULIE login:

STEP 4: Login as Root.

When performing system administration functions you should be a *Super-User* or logged in as *ROOT*.

JULIE login: root

Before you add the terminal, the following information is presented as a review.

Devices & Ports

Devices are special files located in */dev* that point to kernel resident device drivers. Those device drivers perform all of the I/O between a calling process and whatever physical device is being accessed. The advantage of such an architecture is that devices can be written to and read from just like any file.

All hardware devices supported or attached to the system must have a corresponding device if it is to be used. Normally, the device will be located in */dev* and will be given a name reflecting the device it supports. The following list represents the normal devices provided with the IRIS:

Console

console	
syscon	Linked together as same device
systty	

Disks

ip0a - h	474Mbyte fujitsu (Block)
rip0a - h	474Mbyte fujitsu (Character)
ip1a - h	Second Fujitsu (Block)
rip1a - h	Second Fujitsu (Character)
md0a - h	Vertex (or other) 72Mbyte (Block)
rmd0a - h	Vertex 72Mbyte (Character)
md1a - h	Second Vertex (Block)
rmd1a - h	Second Vertex (Character)
si0a - h	Fujitsu/Hitachi 170Mbyte (Block)
rsi0a - h	Fujitsu 170Mbyte (Character)
si1a - h	Second Fujitsu 170Mbyte (Block)
rsi1a - h	Second Fujitsu 170Mbyte (Character)

Floppy disks

mf0a	Standard Floppy (Block)
rmf0a	Standard Floppy (Character)

sf0a	3030 Floppy Controller (Block)
rsf0a	3030 Floppy Controller (Character)
Memory	
kmem	User memory (Kernel space real-time memory)
mem	User memory
Tapes	
mt1	Linked with rmt1
rmt1	Streamer rewind after read (Character)
rmt2	Streamer no rewind after read (Character)
rmt3	1/2" rewind after read (Character)
rmt4	1/2" no rewind after read (Character)
Streamer	
sq0	(Block)
nrsq0	(Character)
Terminals	
tty	Returns your terminal name
ttyT0 - T7	TCP/IP ports (8 is the max)
ttyd0 - d3	RS232 ports
ttyn0 - ttyn31	XNS Network ports
ttyw0 - ttyw9	Textports
Tektronics	
tek	Tektronics printer (Character)
IBM 3270	
pxd	Supports the IBM card (3270 interface controller)
GPIB	
ib0 - 9	GPIB (IEEE 488) controller

STEP 5: Terminal hardware connection.

The IRIS is perfectly happy with a 3-wire connection to a terminal, provided that pins 2 & 3 are crossed. In other words, use a null modem cable with at least pins 2, 3, and 7 connected. Terminals that require additional pins can be handled by either providing those connections to the IRIS or by jumpering. See the following tables on modems for a detailed pinout for cabling.

The RS232 hardware connections

The serial ports on the IRIS are designed to connect directly to *Data Communications Equipment* (DCE) such as modems, via a modem cable. A *modem cable* has pin 1 of the connector on one side connected to pin 1 of the other connector, pin 2 to pin 2, and so forth. The following table shows the pin definitions for the IRIS and for the modem cable.

Modem Cable		
IRIS	Modem	Signals
1	1	Chassis ground
2	2	Transmit data
3	3	Receive data
4	4	Request to send
5	5	Clear to send
8	8	Carrier detect
6	6	Data set ready*
20	20	Data terminal ready
7	7	Signal ground

*Pin 6 is used only for *ttym2*.

Pin Definitions for Modem Cable

Connecting the IRIS directly to **Data Terminal Equipment** (DTE), such as terminals and printers, requires a different cable arrangement, a **null modem cable**. The following table lists the pin definitions for a null modem. The pin numbers that are shown separated by commas (,) should be connected together and also to the other pin or pins listed in the same row.

Null Modem Cable		
IRIS	Terminal	Signals
1	1	Chassis ground
2	3	Transmit data
3	2	Receive data
4	8	Request to send/Clear to send
8	4,5*	Carrier detect
6	20	Data set ready*
20	6,22*	Data terminal ready
7	7	Signal ground

*These connections may be necessary on the terminal side.

Pin Definitions for a Null Modem Cable

These pin definitions work with any serial device that complies with the RS-232C specification.

Most printers work with simpler cabling. If the printer is to be used via a modem, the following table should be used. Refer to your printer manual for more details.

Simplified Null Modem Cable		
IRIS	Printer	Signals
1	1	Chassis ground
2	3	Transmit data
3	2	Receive data
	4,5	Request to send/Clear to send
6*	[6]*,8,20	Carrier detect/Data set ready*
7	7	Signal ground

*Connect 8,20 of printer to 6 of IRIS for modem control.

Simplified Null Modem Pin Definitions for Printers

Most terminals do not require the various handshaking lines such as *clear to send* or *data set ready*, and will work with a three-wire null modem cable. The following table lists the pin definitions for the three-wire null modem cable.

Three-wire Null Modem Cable		
IRIS	Terminal	Signals
2	3	Transmit data
3	2	Receive data
7	7	Signal ground

Three-wire Null Modem for Terminals

The following table summarizes the types of cables to use with different peripherals.

Cable Usage	
Peripheral	Cable
Modem	Modem cable
Terminal	Null modem
Printer	Simplified null modem
CRT	Three-wire null modem

Summary of Cable Types

NOTE:

As of software release 3.4 only the *Three-wire Null Modem Cable* is supported on the IRIS. Future releases will support the other cable configurations. You should check with the *Hot Line* if you are not sure what cable configurations are currently supported.

To connect an ASCII terminal to the cabinet attach an RS-232 cable (null modem) from the ASCII terminal to the IRIS RS232 port 2, 3, or 4 (located on I/O panel in back of the IRIS).

Note:

Since your training workstation already has two ASCII terminals attached, you will not make any physical connections. You may want to pull off the *three-wire* connector and examine it before continuing with the project.

The project will actually have you remove the terminal and then place it back on line.

STEP 6: Editing /etc/inittab.

/etc/inittab determines which ports are open for terminal support, and initiates a *getty* for that port. It is the first part in the chain of events for terminal support.

The Format of */etc/inittab* is fairly simple. Lets look at the file.

```
# cd /etc
# more inittab
is(s):initdefault:# ( # ) $Header: /ws/tel/src/etc/RCS/inittab,v 1.10 87/03/09 17:25:06 fong Exp $
s0::sysinit:/etc/rc.s0 1>/dev/console 2>&1
b0::bootwait:/etc/brc </dev/console >/dev/console 2>&1
b1::bootwait:/etc/bcheckrc </dev/console >/dev/console 2>&1
rc::wait:/etc/rc 1>/dev/console 2>&1
pf::powerfail:/etc/powerfail 1>/dev/console 2>&1
co::respawn:/etc/getty console co_9600 none LDISC0
d1::respawn:/etc/getty ttyd1 dx_9600 none LDISC0
d2::respawn:/etc/getty ttyd2 dx_9600 none LDISC0
d3:x:respawn:/etc/getty ttyd3 dx_9600 none LDISC0
t0:3:respawn:/etc/getty ttyT0 dx_9600 none LDISC0
t1:3:respawn:/etc/getty ttyT1 dx_9600 none LDISC0
t2:3:respawn:/etc/getty ttyT2 dx_9600 none LDISC0
t3:3:respawn:/etc/getty ttyT3 dx_9600 none LDISC0
```

Always put line
Single or 2/3-multi
Like password to getty def
Runs Fscck
Logical Notes 138
do only once wait till done.
console
RS-232 ports
8 Ter IP ports
tcp

Tcp Needs to be set up manually XNS does not

TOP IP
t4:3:respawn:/etc/getty ttyT4 dx_9600 none LDISC0
t5:3:respawn:/etc/getty ttyT5 dx_9600 none LDISC0
t6:3:respawn:/etc/getty ttyT6 dx_9600 none LDISC0
t7:3:respawn:/etc/getty ttyT7 dx_9600 none LDISC0

The line containing *co_9600* and those below it determine which ports *init* will start *getty* processes running against. Each line contains four fields separated by colons. For example:

d3:x:respawn:/etc/getty ttyd3 dx_9600 none LDISC0

- d3** The ID representing the port
- x** ^{Run state} *rstate*. This field is rather complex, but for now the *x* indicates that this line is ignored by *init*. An empty field turns this line "on", and a *getty* process will be started against the terminal.
- respawn** This is an instruction to tell *init* to reinitialize the *getty* on this port if the *getty* process should die. All terminal support ports should have *respawn* in this field.
- /etc/getty** This field specifies to *init* which process to spawn or respawn. *getty* is used to set the login prompt and receive the user's login id.
- ttyd3** This is the device *getty* will be running on. Be extra careful not to have two users running on the same port.
- dx_9600** *getty* uses this field to determine which line to look at in the *gettydefs* file (see below). That line will determine the baudrate and login prompt with which *getty* services the port.
- none** The *type* argument is a character string describing to *getty* what type of terminal is attached to the line in question. The default, in this case is *none*, i.e., any CRT or normal terminal unknown to the system.
- LDISC0** The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal.

Since the lines that begin with *co*, *d1*, and *d2* do not contain an " *x* " in the second field, *getty* processes should have been started against these terminals by the *init* process.

Lets look and see...

ps -aux

USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	TIME	COMMAND
root	445	47.7	3.5	172	116	d1	R	0:01	ps -aux
root	79	0.0	2.1	108	68	d1	I	0:04	/etc/getty ttyd1 dx_9600
root	57	0.0	0.6	12	12	w0	S	0:00	/etc/update
root	60	0.0	1.1	44	32	w0	I	0:00	/etc/cron
root	78	0.0	1.2	48	32	w0	I	0:00	-csh (csh)
root	2	0.0	0.2	640	0	?	D	0:04	pagedaemon
root	1	0.0	1.8	64	56	?	I	0:04	INIT 2
root	0	0.0	0.1	0	0	?	D	0:04	swapper
daemon	72	0.0	1.5	60	44	w0	S	0:00	/usr/lib/lpd
root	69	0.0	1.5	60	44	n31	I<	0:00	/etc/xnsd /usr/local/boot
root	80	0.0	1.2	48	32	d2	I	0:00	/etc/getty ttyd2 dx_9600
lp	75	0.0	2.1	76	68	?	I	0:00	/usr/lib/lpsched

As you can see, two *getty* processes are running against the ports of *ttyd1* & *ttyd2*. The console (which you are on) had a *getty* process started against it, but when you logged on, the *getty* process was terminated and the *shell* (csh) was started for that device.

You are now going to stop the *getty* process for port *d1*.

Edit */etc/inittab* and place an "x" in the second field of the line for port 2 (*d1*).

The next command will cause the *init* process to again access the file.

telinit q

Lets look at the result of *init* process.

ps -aux

USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	TIME	COMMAND
root	445	47.7	3.5	172	116	d1	R	0:01	ps -aux
root	57	0.0	0.6	12	12	w0	S	0:00	/etc/update
root	60	0.0	1.1	44	32	w0	I	0:00	/etc/cron
root	78	0.0	1.2	48	32	w0	I	0:00	-csh (csh)
root	2	0.0	0.2	640	0	?	D	0:04	pagedaemon
root	1	0.0	1.8	64	56	?	I	0:04	INIT 2
root	0	0.0	0.1	0	0	?	D	0:04	swapper
daemon	72	0.0	1.5	60	44	w0	S	0:00	/usr/lib/lpd
root	69	0.0	1.5	60	44	n31	I<	0:00	/etc/xnsd /usr/local/boot
root	80	0.0	1.2	48	32	d2	I	0:00	/etc/getty ttyd2 dx_9600
lp	75	0.0	2.1	76	68	?	I	0:00	/usr/lib/lpsched

As you can now see, only one *getty* process is now running.

O.K, edit the file again and bring port 2 back on line. Then execute the *telinit q* command again.

Once you get the terminal you took off line back on line, move to one of the ASCII terminals and login as student. After you login perform the *ps -aux* command and verify that a *shell* is now running in place of the *getty* process.

Logout of the ASCII terminal after you execute the *ps* command.

STEP 7: Editing the */etc/gettydefs* file.

The file */etc/gettydefs* provides the *getty* process the information it needs to fully support a user on a terminal line. Lets look at the file...

```
# cd /etc
```

```
# more gettydefs
```

```
co_9600# B9600 # B9600 SANE TAB3 #\r\n\r\nJULIE login: #co_4800
```

```
co_4800# B4800 # B4800 SANE TAB3 #\r\n\r\nJULIE login: #co_2400
```

```
co_2400# B2400 # B2400 SANE TAB3 #\r\n\r\nJULIE login: #co_1200
```

```
co_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #co_300
```

```
co_300# B300 # B300 SANE TAB3 #\r\n\r\nJULIE login: #co_9600
```

```
dx_9600# B9600 # B9600 SANE TAB3 #\r\n\r\nJULIE login: #dx_4800
```

```
dx_4800# B4800 # B4800 SANE TAB3 #\r\n\r\nJULIE login: #dx_2400
```

```
dx_2400# B2400 # B2400 SANE TAB3 #\r\n\r\nJULIE login: #dx_1200
```

```
dx_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #dx_9600
```

```
du_1200# B1200 # B1200 SANE TAB3 #\r\n\r\nJULIE login: #du_300
```

```
du_300# B300 # B300 SANE TAB3 #\r\n\r\nJULIE login: #du_1200
```

Each line has five fields summarized below:

co_9600	This is the name pointed to by the last field of an <i>inittab</i> entry. For example, if the <i>getty</i> on port <i>ttyd2</i> was given <i>co_9600</i> in <i>inittab</i> , then the <i>getty</i> would start using the information on this line in <i>gettydefs</i> .
B9600	This is a <i>termio(7)</i> baudrate used if no terminal type has been passed to <i>getty</i> . In that case, <i>getty</i> will service the port at the speed specified using "raw" (dumb teletype) mode.
B9600 SANE TAB3	This is the "final flags" field. This field is used to condition the line using <i>termio(7)</i> parameters before <i>getty</i> executes <i>login</i> . This field can also be empty. The default settings include: B9600 The initial baudrate SANE A flag ensuring a rational handling of the port between the processor and the terminal. One of a limited number of terminal types could also be put here (see <i>getty(1)</i>), but generally is not done. <i>Getty's</i> function is served with a minimum functionality, permitting the default dumb tty interface until <i>login</i> is started. TAB3 This converts tabs to spaces on the terminal
rn IRIS login	This field contains the prompt printed to the terminal by <i>getty</i> .
co_4800	This field points to another line to go to if the terminal sends a BREAK signal. This is used by <i>getty</i> to adjust baudrates.

Commonly, the *gettydefs* baudrate sequence will be:

9600 will point to 4800
4800 will point to 2400
2400 will point to 1200
1200 will point to 300
300 will point to 9600

The administrator can restrict or expand the sequence to include any expected baud rate.

You should now edit the */etc/inittab* file and change the pointer into the */etc/gettydefs* file for one of the ASCII terminals. Select *dx_4800* baud rate for the terminal.

Reboot the system, go to *multi-user* and verify that garbage appears at the ASCII terminal when

the login prompt is displayed.

When you depress *break*, garbage will still be printed. This is because each line in *gettydefs* for the ASCII terminal (the lines beginning with *dx*) point to themselves. Edit *gettydefs*, changing each line to point to the next baud rate, with *dx_1200* pointing to *dx_9600*. (This is the way the console [*co_XXXX*] is set up)

Reboot the system again and verify that now when *break* is depressed, the correct baud rate will be found by continual depressions of *break*.

STEP 8 : Editing the */etc/ttytype* file.

The *ttytype* file is read by the *login* process when the user attempts to login to a given port. *Login* checks the port, takes the terminal type and uses it to get appropriate terminal control information from *termcap(5)*. If your terminal does not have a definition in *termcap*, you must pick an equivalent entry or create one.

/etc/ttytype looks like the following:

```
wsiris    systty
wsiris    console
wsiris    syscon
?v50am    ttyd1
?du       ttyd2
?du       ttyd3
?v50am    ttyn0
?v50am    ttyn1
?v50am    ttyn2
.
.
?v50am    ttyn31
```

The first column should be set to the terminal type. Defaults have been built in expecting a VT50 terminal. If the workstation is to be attached to a net with other IRIS workstations, you should change the v50am entry for each permitted net port (ttyn*) to *wsiris* as the initial terminal type.

The leading ? before most of the entries is a prompt indicator. When a person logs in, the ? causes the system to ask if the default is acceptable ((v50am?)). The user can hit 'return' to accept the default or enter another terminal type.

/etc/termcap

The */etc/termcap* file contains protocol descriptions for a large number of common terminals. If your terminal type is not listed, you can create an entry of your own.

Display */etc/termcap* and see how many different terminals are supported.

tset and stty

Each user's shell startup file (*.login* or *.profile*) should contain *tset* and *stty* commands that use information from */etc/ttytype* and */etc/termcap* to initialize the terminal interface. This way, each user can configure his/her own terminal environment. The IRIS workstation provides examples in the root (/) *.login* and *.profile* files:

.login:

```
stty erase '^H' kill '^U' intr '^C' echoe
set          ignoreeof noglob
set          tmp = ('tset -S -Q')
setenv      TERM$tmp[1]
unset      noglob
```

.profile

```
stty erase '^H' kill '^U' intr '^C' echoe
eval 'tset -S -Q' ; TERMCAP=
```

When you created the new account (the last lab project) you copied over these basic files. These lines of code are in these files and the user then adds to these files all the instructions and commands he/she wants for a custom account environment.

STEP 9: Test the Terminal

You have already performed what would be the final steps of installing a new ASCII terminal, they are:

- Inform *init* of the change to */etc/inittab* by typing the command:

```
telinit q
```

Entering this command causes *init* to read */etc/inittab* and start a *getty* process running on the terminal port.

- Hit the return key on the terminal, making sure the prompt appears, ungarbled. If trash is printed, hit BREAK until the right stuff shows up. Then change either the terminal configuration or the inittab file to begin with the correct baud rate.

The user should now be able to use the remote terminal without difficulty.

Remember, section three of this lab project gives only the procedure without all the instructional comments.

SECTION 2: Adding an ASCII Printer

In the following procedure, steps 1-8 serve only to test the printer. Once you are sure you have the correct parameters for the printer you are using, and it is printing correctly, step 9 illustrates how to edit */etc/rc* and make the required changes that insure the printer will be configured every time the system is booted.

STEP 1: Connecting the Printer to the IRIS.

To connect a printer to the cabinet, attach an RS-232 cable (null modem type) from the printer to "Port 2", "Port 3", or "Port 4" on the standard I/O panel. The manual for the printer should have a specification for its RS-232 interface and may require some jumpering of RS232 pins. Most modern printers are 3-wire null-modem compatible.

STEP 2: Configuring the Software for a Printer Connection.

To enable the software for connecting a printer to the IRIS:

- Log in as *root* or become the *super-user* by entering the *su* command.
- Edit the file */etc/inittab* to disable *getty* on the port to which the printer is connected by entering an 'x' in the second field corresponding to that port.
- Inform *init* of the change to the */etc/inittab* file by typing the command:

```
# telinit q
```

Entering this command causes *init* to read */etc/inittab* and kill the *getty* for the port.

- Inform the shell of the port you have chosen by giving the port's device name.

In the following command examples, the asterisk (*) represents *d1*, *d2*, or *d3*. (see table on next page for related */dev* devices when adding terminals or printers)

SERIAL DEVICE NAMES

Serial Device Names	
Device Name	Description
/dev/console	Console terminal
/dev/syscon	System console (linked to /dev/console)
/dev/systty	System tty (linked to /dev/console)
/dev/ttyd1	Serial on standard I/O panel
/dev/ttyd2	Serial on standard I/O panel
/dev/ttyd3	Serial on standard I/O panel
/dev/tty2	Serial used with modem control used in lieu of ttyd2

If using csh:

A

set port=/dev/tty*

change

If using sh:

port=/dev/tty*

/TTY02

The above commands set a variable (*port*) equal to the value */dev/tty**. Throughout the procedure, this value (which is the device name of the interface for the printer) is used every time the variable *port* is used in a command.

i.e. set port=/dev/ttyd3

STEP 3: Give all users permission to access the port.

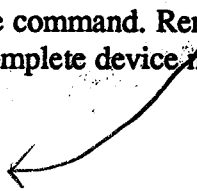
chmod 222 \$port

Because of A we can do this otherwise you would have to do

NOTE:

The variable *port* is being used in the above command. Remember, it was created in STEP 2. If you had not created the variable, then the complete device name would have to be entered at this point.

i.e. chmod 222 /dev/ttyd2



STEP 4: Change the owner of the device to *root* and its group to *sys*:

```
# chown root $port  
# chgrp sys $port
```

STEP 5: Link the device file for the printer port to */dev/lp*.

```
# rm -f /dev/lp  
# ln $port /dev/lp
```

STEP 6: Open the device file.

UNIX automatically resets the baud rate and terminal modes for all serial devices after programs open and close a device file. Therefore, before setting up the printer parameters, you must first run a program to open the device file for the printer and keep it open. Type these lines to the shell:

If using *cs*h:

```
# sleep 10000000 < $port >& /dev/null &
```

If using *sh*:

```
# sleep 10000000 < $port 2> /dev/null &
```

These lines open the device file for the port you have chosen and keep it open for four months.

Lets look at the result of executing the above *sleep* command.

```
# ps -aux
```

USER	PID	%CPU	%MEM	SZ	RSS	TT	STAT	TIME	COMMAND
root	445	47.7	3.5	172	116	d1	R	0:01	ps -aux
root	79	0.0	2.1	108	68	d1	I	0:04	/etc/getty ttyd1 dx_9600
root	57	0.0	0.6	12	12	w0	S	0:00	/etc/update
root	80	0.0	1.0	32	28	w0	I	0:00	sleep 10000000
root	60	0.0	1.1	44	32	w0	I	0:00	/etc/cron
root	78	0.0	1.2	48	32	w0	I	0:00	-csh (csh)
root	2	0.0	0.2	640	0	?	D	0:04	pagedaemon
root	1	0.0	1.8	64	56	?	I	0:04	INIT 2
root	0	0.0	0.1	0	0	?	D	0:04	swapper
root	69	0.0	1.5	60	44	n31	I<	0:00	/etc/xnsd /usr/local/boot
lp	75	0.0	2.1	76	68	?	I	0:00	/usr/lib/lpsched

As you can see, the *sleep* process is running. This keeps the printer device file open. Also, notice that only one *getty* is running (you killed the other one in step 2).

STEP 7: Set printer parameters.

The *stty* command sets terminal or printer parameters. For example, to use a printer at 1200 baud with *XON/XOFF* protocol, that does not supply a carriage return before each line feed, the *stty* command would look like the following:

```
stty 1200 ixon -onlcr < $port
```

See *stty(1)* for the options that can be set. In *stty* commands, a dash (-) in front of an option means to turn it off. No dash means to turn it on.

The printer that we will use in class is a Data Products SPG-8020. Enter the following *stty* command to set its parameters.

```
# stty 9600 ixon -onlcr < $port
```

STEP 8: Test the line printer by entering these commands:

You are now going to redirect the output of a file (we are using */etc/passwd*) into the *special device file* for the printer. The file that the output is redirected into is */dev/lp*, but this file is *linked* to */dev/tty** (see step 5), therefore, the data will end up in the special file. Once in the special file, the printer driver will output the data to the printer and the *passwd* file will be printed.

```
# cat /etc/passwd > /dev/lp
```

IRIS workstation only:

```
# lpr /etc/passwd
```

The first command, using *cat*, tests whether characters can be sent to the device, */dev/lp*, by sending the file */etc/passwd* directly to the printer. If the printer does not print the file, check the serial line connections and the arguments to the *stty* command.

The second command tests the line printer spooling system. The spooling software for the printer is not included with terminals. Only writes directly to the printer device */dev/lp* are possible. When this command is used, a header is printed before the data is printed.

The next steps should be completed only after you're sure of the arguments to the *stty* command (step 7), in other words, your printer works the way you want it to work.

STEP 9: Edit */etc/rc*.

This file is accessed every time the system is booted. By placing the following three commands into */etc/rc*, the printer will be configured every time you re-boot the system.

Edit the file */etc/rc*. Locate the following line:

```
echo "Daemons started."
```

NOTE:

You will notice that the above line exists in two places:

- *If your machine is a terminal, place the three commands after the first occurrence of the above line.*
- *If your machine is a workstation, place the three commands after the second occurrence of the above line.*

Also, the three commands you add to */etc/rc* can go anywhere on the line, but placing them at the first tab stop uses correct programming format.

The three commands are:

```
nohup sleep 10000000 < devicename 2> /dev/null &  
sleep 3  
stty options < devicename
```

i.e.

```
nohup sleep 10000000 < /dev/ttyd2 2> /dev/null &  
sleep 3  
stty 9600 ixon -onlcr < /dev/ttyd2
```

Lets look at the file */etc/rc* after the above changes.

```
# more rc  
#!/bin/sh  
# ~(#)rc.sh          1.7  
# $Source: /ws/rel/src/etc/RCS/rc,v $  
# ~(#)$Revision: 1.13 $  
# $Date: 85/07/31 03:19:43 $  
TZ='cat /etc/TZ'  
export TZ  
if [ ! -f /etc/mnttab ]  
then  
    > /etc/mnttab  
    /etc/devnm / | /etc/setmnt
```

fi

```
case '/bin/uname -t' in  
2300|2300T|3010)
```

```
    PATH=':/usr/bin:/bin:/etc'  
    export PATH  
    rm -rf /tmp/* /tmp/.*? /usr/spool/colord/?data*  
    echo "Cleared /tmp /usr/spool/colord"  
    if [ -f /.mexrc ]  
    then  
    su iris -c 'SHELL=/bin/tesh HOME=/ TERM=wsiris mex'  
    fi  
    if [ -r /etc/sys_id ]  
    then  
    hostname 'cat /etc/sys_id'  
    echo Hostname: 'hostname'  
    else  
    echo No hostname  
    fi  
    echo "Daemons:"  
    /etc/update;echo " update"  
    if [ -r /etc/rc.xns ] ; then sh /etc/rc.xns ; fi  
    if [ -r /etc/rc.tcp ] ; then sh /etc/rc.tcp ; fi  
    if [ -r /etc/rc.488 ] ; then sh /etc/rc.488 ; fi  
    echo "Daemons started."  
    sleep 3  
    clear  
    exit  
    ;;
```

```
esac
```

```
PATH=':/usr/local/bin:/usr/bin:/bin:/etc'
```

```
export PATH
```

```
set 'who -r'
```

```
if [ $7 = 2 -o $7 = 3 ]
```

```
then
```

```
    echo "Mounting:                sh /etc/rc.fs  
    echo "."  
    /usr/lib/ex3.7preserve -;echo "Preserved editor files"  
    rm -rf /tmp/* /tmp/.*?;echo "Cleared /tmp"  
    echo "Resetting locks and logs"  
    rm -f /usr/spool/lpd/lock /usr/spool/lp/SCHEDLOCK  
    rm -f /usr/adm/acct/nite/lock*  
    cd /usr/adm ; chmod go-rwx sulog cronlog OLDsulog OLDcronlog  
    cd /usr/adm ; cp sulog OLDsulog; cp /dev/null sulog
```

```
cd /usr/adm ; cp cronlog OLDcronlog; cp /dev/null cronlog
rm -f /usr/spool/uucp/LCK* /usr/spool/uucp/ST* /usr/spool/uucp/TM*
hostname 'cat /etc/sys_id'
echo Hostname: 'hostname'
echo "Daemons:"
/etc/update;echo " update"
/etc/cron;echo " cron"
if test -f /etc/rc.xns ; then sh /etc/rc.xns ; fi
if test -f /etc/rc.tcp ; then sh /etc/rc.tcp ; fi
if test -f /etc/rc.488 ; then sh /etc/rc.488 ; fi
if [ -x /usr/lib/lpd -a -x /usr/lib/lpsched ]
then
/usr/lib/lpd;echo " lpd"
/usr/lib/lpsched;echo " lpsched"
fi
echo "Daemons started."
nohup sleep 10000000 < /dev/ttyd2 2> /dev/null &
sleep 3
stty 9600 ixon -onlcr < /dev/ttyd2 2> /dev/null &
fi
```

The three added commands are highlighted. Notice they are at the first tab stop.

After you edit the `/dev/rc` file, your printer is now ready to be used any time you log into the system.

Appendix 4.2 (of the student workbook) gives additional information concerning printer spooling and setting up printer classes, but this should always be the system administrators task and not yours.

SECTION 3: Generic Procedures

This section shows only the actual steps required to add an ASCII terminal and/or printer to the workstation.

The IRIS OWNER'S GUIDE, Series 3000, gives outlines of these procedures. This lab project uses a slightly different approach, but the results are the same. (Also, Technical Aid I/O-013 R1 gives additional information concerning connecting an ASCII printer)

The numbered steps of this section are different than section 1, because, the steps showing the boot process are not listed.

Adding an ASCII Terminal

STEP 1: Connect the terminal to RS232 port 2, 3, or 4.

STEP 2: Become Super-user or login as ROOT.

STEP 3: Change directory to /etc.

STEP 4: Edit the file *ttytype* to specify the type of terminal being attached.

STEP 5: Edit the file *inittab* to turn on the *getty* for that port.

STEP 6: Edit the file *gettydefs* to control the baud rates, login prompt, and port configuration.

STEP 7: Issue a *telinit q* to make *init* aware of the change.

STEP 8: Login on the new terminal.

Adding an ASCII Printer

STEP 1: Connect the printer to RS232 port 2, 3, or 4.

STEP 2: Become Super-user or login as ROOT.

STEP 3: Change directory to /etc.

STEP 4: Edit *inittab* to disable the *getty* on the port that you connected the printer (place an "x" in the second field).

STEP 5: Issue a *telinit q* to make *init* aware of change.

STEP 6: Inform the *shell* what port you have selected:

If csh set port=/dev/tty*

where * = d1, d2, or d3

If sh port =/dev/tty*

where * = d1, d2, or d3

STEP 7: Give all users permission to access the port.

chmod 222 \$port

STEP 8: Change the owner and group to: *root* and *sys*.

chown root \$port

chgrp sys \$port

STEP 9: Link the device file for the printer port to */dev/lp*.

rm -f /dev/lp

ln \$port /dev/lp

STEP 10: Open the device file.

If csh sleep 10000000 < \$port >& /dev/null &

If sh sleep 10000000 < \$port 2> /dev/null &

STEP 11: Set printer parameters.

```
stty options < $port
```

STEP 12: Test printer.

```
cat /etc/passwd > /dev/lp
```

STEP 13: If above works, edit */etc/rc* and add the following lines after the line *echo "Daemons Started"*.

Remember, there are two occurrences of this line, select the correct one. The determining factor is the type of system you are attaching the printer to: a Terminal or a Workstation.

First occurrence = Terminal

Second occurrence = Workstation

```
nohup sleep 10000000 < devicename 2> /dev/null &  
  
sleep 3  
  
stty options < devicename
```


SECTION 4: Review

Student Name:

Please complete the review and hand it into your instructor.

1. What cable do you use when connecting an ASCII terminal to the IRIS?

RS232 215 crossed + 7
 Simplified null modem

2. What IRIS ports can you connect an ASCII terminal to?

D1-3 102 RS23C 2-3

3. What pins of the *three-wire* cable are used and what signals does each line carry?

2 Transmit Data
 3 Receive Data
 7 Signal Ground

4. What file do you edit to start a *getty* process for a port that you are going to connect an ASCII terminal to?

/etc/inittab

5. When you make a change to */etc/inittab*, what command do you use to inform *init* of the change?

Telinit g

6. What files determine the baud rate for a terminal?

/etc/gettydefs + inittab
 also inittab

7. What file would you edit to inform the system that the attached terminal is a *Tektronix* terminal?

etc / TTYType

8. What file defines the parameters for all IRIS supported terminals?

ETC / TERMcap

9. What ports can you attach an ASCII printer to?

PS M2 2-4

10. What file must you edit to disable the *getty* process for the port that you select to attach your printer to?

*Because getty is for login + 2 way communication
So you to disable getty process*
ETC / INITTAB 2 Field (X)

11. Show the command used to allow all users access to the printer port.

Chmod a 222 /Dev/TTYD1 a \$Port

12. What file do you link to the special file for the printer?

LN \$Port + /Dev/LP

13. What is the *stty* command used for?

Set printer parameters set port parameters

14. Show the command used to cause printing to occur from either an IRIS Terminal or IRIS Workstation.

> /Dev/LP

15. What file do you edit and add three commands to, that insures that the printer is configured each time the system is booted?

ETC/RC

16. What Technical Aid gives additional information concerning printer installations?

TA 110-013-R1
appendix 4-2

SOFTWARE CARTRIDGE TAPE FORMATS

NOTE: The first entry in each box is the contents of that file on tape.
 The second entry is the format in which it is written.
 The entries in *italics* may or may not be on the tape.

L2-W3.5r2 Update Tape (800-1019-001)

dist cpio 004-0024-028	root cpio 004-0144-044	usr cpio 004-0144-544	man cpio 4-0154-034	games cpio 4-0164-034	demos cpio 4-0174-034	gifts cpio 4-0184-034	tutorial cpio 4-0204-031	mail cpio 4-0194-031
------------------------------	------------------------------	-----------------------------	---------------------------	-----------------------------	-----------------------------	-----------------------------	--------------------------------	----------------------------

NOTE: Used by customer's to update software from 3.4 to 3.5.

GL2-W2.5 Update Tape (800-1001-001)

dist cpio 004-0024-036	root cpio 004-0134-183	usr cpio 004-0134-193	<i>options</i> <i>cpio</i>
------------------------------	------------------------------	-----------------------------	-------------------------------

NOTE: Used by customer's to update software from 2.4 to 2.5.

GL2-W3.5r2 Bootstrap Tape (800-1004-001) [FE TAPE]

dist cpio 004-0024-028	bootstrap dd <i>mini-root</i> 004-0124-045	root cpio 004-0134-163	root% cpio 004-0134-663	usr cpio 004-0134-173	usr% cpio 004-0134-673
------------------------------	---	------------------------------	-------------------------------	-----------------------------	------------------------------

NOTE: Used by customers to restore their system disk from tape.

GL2-W2.5 Bootstrap Tape (800-1002-001)

dist cpio 004-0024-032	bootstrap dd 004-0124-061	root cpio 004-0134-181	usr cpio 004-0134-191
------------------------------	---------------------------------	------------------------------	-----------------------------

NOTE: Used by customer's to update software from 2.3/Bell to 2.4/EFS.

GL1-W2.3 Bootstrap Tape (800-)

dist cpio	bootstrap dd	root&usr cpio	<i>options</i> <i>cpio</i>
--------------	-----------------	------------------	-------------------------------

NOTE: Used by customer's to update software from 2.2 to 2.3.

*make 3k diagnostic tape
 already on delphin
 in /d/diagnostics_3k
 cpio -chv1
 cpio -ihv1 to verify
 {9250k read}*

GL2-W3.5 Bootable Backup Tape (mkboot)

stand cpio	root dd	usr cpio
---------------	------------	-------------

NOTE: Made by customers to restore their system disk from tape.

GL2-W2.5 Bootable Backup Tape (mkboot)

stand cpio	md0g dd	root dd	usr cpio
---------------	------------	------------	-------------

NOTE: Made by customers to restore their system disk from tape.

L1-W2.3 Bootable Backup Tape (mkboot)

stand cpio	root dd	usr cpio
---------------	------------	-------------

NOTE: Made by customers to restore their system disk from tape.

<i>model</i>	<i>Release</i>
1000	
1200	
1400	2.3.1
1500	
2000	
2200	2.5 r1
2400	
2500	
2400T	
2500T	3.5 r2
3000	

CONTENTS

Appendix 4.1: Installing Software Updates	1
Installing Software Updates	2
I. Introduction	2
II. Module Objectives	2
III. Tape Organization	2
IV. Overview of Software Installation Procedure	3
V. Software Installation Procedure	4
V.1. Using A Remote Tape Drive	8
VI. Software Installation Tools	8
VI.1. Files	8
VI.2 Commands	9

Appendix 4.1: Installing Software Updates

This appendix gives information for the procedure use when installing software updates.

You will use this procedure when installing options that the customer purchased after the initial system purchase and install, or when SGI releases updates to the existing system.

When update tapes are shipped, instructions for installing the files are shipped with the tape, but, these instructions are for experienced UNIX people and like so many of the instructional documents, the instructions are "unfriendly" to the novice user or Field Engineer.

This document was taken from the SGI System Administration Course and placed into the Hardware Course so that the Field Engineer would have a more detailed procedure of the software installation instructions.

Installing Software Updates

I. Introduction

With the advent of a wider product line and more frequent software updates, Silicon Graphics began a system of "update" tapes. These tapes:

- Automate and simplify the software update process
- Provide self-updating installation tools
- Permit specialized tools required only for unique options without having to provide those tools as release software.

This module describes how to install software updates and software options on the IRIS workstation.

II. Module Objectives

On completion of this module, the student will:

- Describe the format of a typical Update tape
- Prepare the IRIS for update installation
- Install software updates
- Recover the system should the update fail

III. Tape Organization

Silicon Graphics IRIS workstation software is distributed on quarter-inch cartridge or half-inch nine-track tape. The software is stored on the tape in *cpio (1)* format.

The tape consists of software to be installed on the user's system and a set of distribution tools that allows users to access the software. The software contains one or more components or *entries*.

A tape *entry* is a collection of related software that is installed as a unit. Entries are identified by short names recognized by the distribution tools. For example, the files making up the UNIX system update are collected in an entry called *upd*, and the FORTRAN language option is in an entry called *ftn*. On each tape, the first entry is a *cpio (1)* image containing the distribution tools. The programs, scripts, and text files that make up the distribution tools reside in a directory called */dist*. (For a full description of the software installation tools, see Section 5 of the 3000 Owner's Guide)

As part of the distribution tools, each tape includes two files that allow the user to determine which entries are on the tape. The two files are:

- *toc* A table of contents of the entries on the tape, in the order that they appear on the tape, and the sizes of the entries.
- *desc* A list of all possible entries with a brief description of each.

Software update tapes contain an entry for the UNIX update software, entries for optional update software such as on-line manual pages and games, and an entry for each software option that you have ordered. A software option tape contains only entries for each option that you ordered.

IV. Overview of Software Installation Procedure

The first step in the installation process is to read the *dist* directory into the root directory with a *cpio* command. The distribution tools can then be used for all further tape positioning and access.

The next step in the installation process is to determine which entries are on the tape, and which of these entries to load onto the disk. Use the table of contents (*toc*) and the entry descriptions (*desc*) to help determine which entries to install on your system.

If there are user files on the workstation, or if you are installing many software options, there may not be sufficient disk space to install all of the new software. To find out if there is sufficient space on the disk, you need to compare the amount of space required for installation with the amount of space available on the disk. The distribution tool called *Spchk* computes the projected change in disk usage for the root and */usr* file systems as compared to the current usage.

If *Spchk* projects an increase in disk space usage, compare the value reported by *Spchk* with the number of available disk blocks reported by *df (1)*. If the number of blocks available is less than the number required, the disk will overflow when the software is installed. To prevent this, back up some user files to tape and then delete them from the disk.

After determining whether there is sufficient space on the disk, the next step is to read in the files

from the disk with another distribution tool called *Read*. This tool reads in the selected entries (or all entries, by default) and puts the individual files in their proper places on the disk.

Once they are read in from tape, some files require special attention. A distribution tool called *Install* attends to these details. For example, when new libraries are read in, it is necessary to run the *ranlib (1)* program to make sure the library is in the condition expected by the loader. The *Install* command runs *ranlib (1)* for each library in the entry.

The *Install* command also updates configuration files as required by the new software. (Configuration files are files that control the site- and machine-specific details of system operation.) Silicon Graphics attempts to avoid updating configuration files, but occasionally it is necessary to do so because of changed functionality in a release.

When the *Install* command updates the configuration files, the old configuration files are saved under a different name, and the new configuration files replace them. The names of configuration files that are updated are reported as the *Install* command runs. As the next step in the installation procedure, you need to compare the old configuration files with the new, and make any site-specific alterations to the new version.

If a new kernel has been installed, the last step in the installation process is to reboot the system. This ensures that the new system uses the new kernel.

Once installation is completed, delete the distribution tools from the disk.

V. Software Installation Procedure

To install software updates and options, follow the steps below. Updates and options can be installed separately or together. The basic procedure assumes that only one machine is involved. If the tape drive is on a remote machine to be accessed using XNS, see "Using a Remote Tape Drive" at the end of this section.

- Make sure the system is in multi-user mode, but that there are no other users logged on during the installation process. Log in as *root*.
- Change your working directory to the *root* directory:

```
cd /
```

- If there is an old copy of the distribution directory present, remove it:

```
rm -rf dist
```

Appendix 4.1: Installing Software Updates

- Put the distribution tape in the tape drive.
- Read in the distribution tools with *cpio* :

```
cpio -ivhmud1
```

If you are using a half-inch tape, use device 3 rather than 1:

```
cpio -ivhmud3
```

- To make it easier to access the distribution tools, put the */dist* directory in your search path. Enter this command carefully; if the search path is wrong, the shell will not be able to find the programs you try to run.

```
set path = ( /dist /usr/bin /bin /etc )
```

- Determine which entries are on the tape, and which of them you want to install. To see the names of the entries on tape, examine the table of contents file *toc* :

```
cat /dist/toc
```

To find out what each entry name means, examine the description file *desc* :

```
cat /dist/desc
```

Most of the distribution tools accept a *names* argument that determines which of the entries listed in *toc* will be processed.

To process all of the entries listed in *toc* , omit the *names* argument. To process a subset of the available entries, list the entry names as the *names* argument, in the same order as they are given in the *toc* file.

- Unless the system is new, or has no user files on it, check the disk space to ensure that the new entries will fit on the disk. To do this, use the *Spchk* and *df(1)* commands:

```
Spchk names
df
```

The *Spchk* command reports the projected change in disk usage for each file system (*/* and */usr*) as plus or minus some number of disk blocks. If the number of disk blocks reported by *Spchk* is an increase in disk usage, compare the increase with the number of available blocks reported by the *df* command. Remember that the increase projected by *Spchk* is approximate, so be generous — allow a few hundred extra blocks if possible. If it looks

close, or if there are not enough free blocks, back up some user files to tape and delete them from the disk. When you have installed the software, you can reorganize the disk to make room for restoring the files that you deleted.

Remember that the disk is divided into *root* and *usr* file systems. If the space problem is on */usr*, for example, only deleting files from */usr* will help.

- Read in the entries from the tape:

Read *names*

If you would like feedback while the command progresses, give the *-v* (verbose) option:

Read *-v names*

The *v* option of the *Read* command shows the names of the files as they are extracted from the entries on the tape. The disadvantage of the verbose option is that error messages can scroll off the screen unnoticed.

- Perform the installation details for each entry:

Install *names*

This command indicates its progress and the names of the affected files.

As the *Install* command runs, it reports the names of the altered configuration files. If the *Install* command updates any configuration files, the old configuration files are saved under a different name, and the new configuration files replace them. After the *Install* command is finished, compare the old configuration files with the new configuration files. Make any necessary site-specific alterations to the new version.

While you are running the *Install* command, you may receive a message indicating that the you need to run the *Install* command again after the system is rebooted. (See Step 14, below.)

- If you install a new kernel, verify that certain critical files are present before you reboot the system. This is by necessity an incomplete check, but does serve to catch some of the more significant problems. If any of these problems exist, they must be corrected before the system is rebooted. Failure to do so may make it impossible to bring the system back up again. To verify that the critical files are present, type:

Verify

If *Verify* reports any errors, correct them. If you need assistance, call the Geometry Hotline (see Chapter 1 of the *IRIS Series 3000 Owner's Guide*).

- If an update or any communications options were installed, reboot the system:

```
reboot
b
multi
```

- Log in as *root*.
- Once the system is back up, and if you were instructed to do so perform the installation cleanup details:

```
/dist/Install -cleanup
```

- Delete the distribution directory:

```
rm -rf /dist
```

- If you deleted user files to make room on the disk, restore them.

V.1. Using A Remote Tape Drive

If you are installing the distribution on a workstation that does not have a tape drive, use a remote workstation with a tape drive and the XNS network protocol.

Follow the procedure above, but with these changes:

- Log in as *root* on the remote machine. Follow steps 2-4 as above. Using the commands in step 5 above, read the distribution tools onto the remote machine using *cpio*.

Log in as *root* on the machine on which you are installing the software. Copy the distribution tools to the machine on which you are installing the software using these commands (*remote* is the name of the machine with the tape drive):

```
cd /
xcp -r -v remote:/dist.
```

- Instead of reading with the `-v` option, issue the `Read` command with the `-x` option. This option tells the `Read` command the name of the remote machine.

```
Read -x remote names
```

The `names` argument is optional.

VI. Software Installation Tools

This section describes the distribution tools in more detail. The distribution tools are collected in a directory called `/dist`. The set of tools includes commands used to manage software installation, and files that describe the distributed software. The `/dist` directory is written on each distribution tape in `cpio (1)` format.

VI.1. Files

A file called `toc` in the distribution directory indicates which entries are on the tape. Each line in this file contains the name and size of an entry. Entries in this file are in the same order as they appear on the tape. Most of the distribution commands use this file to obtain the default list of entry names to be processed.

The `desc` file is a text file that gives a brief explanation of the contents of each possible entry. This file is useful for correlating entry names with their contents.

There are several files called `list.name` in the distribution directory, where `name` is an entry name. These lists contain the names of the files in each entry. The path names for each entry are given in sorted order, with no leading slash. These lists are used by the distribution commands while entries are being processed. If you have a specific question about files included in an entry, examine these files.

The `fluff.name` files in the distribution directory contain the names of files shipped in previous releases that are now obsoleted by the new software. There is a "fluff" list for each entry. These lists are used by the `Install` command, so that obsolete files will be removed automatically when the new software is installed. The fluff lists are often very short, containing only the old part number file in the `/Versions` directory.

The `nsizes` file contains the size of all files for all entries that are part of the software release from which the distribution was made. The `Spchk` command uses this file to determine the size of the new files on the tape *before* the tape is actually read in.

VI.2 Commands

If the disk is full or nearly full, attempting to install new software can exhaust all available space. In some cases, a partially installed UNIX update can be quite difficult to work with. It is always best to be sure that there is enough space to install the new software *before* reading the entries from the tape.

The *Spchk* command approximates the change in disk space usage that will occur when the selected entries are read in. It does this by comparing the old and new sizes of each file in the entry. The "old" size is taken from the file currently residing on disk. The "new" size is taken from the *nsizes* file. The total change is reported as an increase or decrease in the amount of disk space that will be used. An increase can be compared with the amount of available space on the disk to determine whether the new software will fit.

The two most significant distribution tools are the *Read* and *Install* commands. The *Read* command is used to read entries from the distribution tape and put them on the disk. Normally, the *Read* command does its work relatively quietly. The *-v* (verbose) option causes the names of the individual files to be reported as they are extracted from the entries on tape and written to the disk. If no arguments are given to the *Read* command, all entries listed in the *toc* file in the distribution directory are read from the tape (i.e. the entire tape). To select specific entries instead of all entries, specify them as arguments.

The *Install* command is used after the desired entries are read. Like the *Read* command, it processes all entries listed in *toc* by default. To cause the *Install* command to process only specific entries rather than all entries, name them as arguments. This command takes care of a wide range of details that must be attended to when new software is installed. For example, it deletes obsolete files from previous releases of the entry, runs the *ranlib (1)* command on each library, installs new kernels as */defaultboot* and */vmunix*, saves old configuration files while installing new ones, installs shared-text files that cannot be directly overwritten, makes new devices, and makes sure that certain links are done properly. In some cases, certain tasks cannot be done until the system is rebooted. If so, *Install* will ask you to run *Install* again after rebooting the system.

The *Have* command indicates whether the specified entries are installed on the disk, and whether they are complete or not. By default, all entries in the *toc* file are checked; specific entries can be checked by naming them as arguments. The *-v* (verbose) option causes the names of the missing files to be reported.

The *Verify* command makes sure that certain critical files are in place before rebooting the system. When installing new software, especially complete system updates, it is best to be on the safe side and do everything possible to ensure the integrity of the system before attempting to take the old system down and bring the new one up. If there are problems, it may not be possible to bring the new system up. The *Verify* command reports specific problems, and should be

used whenever an update is installed, but necessarily performs an incomplete set of checks.

CONTENTS

Appendix 4.2: Printer Spooling and Communications	1
Section 1: Printers	2
I. Printer Spooling	2
II. Commands Related to <i>LP</i>	3
III. Initializing the system	4
IV. Adding new destinations	5
V. Setting up classes	6
VI. Setting up and using an interface program	7
VII. Using <i>lpstat</i>	10
SECTION 2: Modems and Direct Connection to Other Systems	11
I. The RS232 hardware connection	11
II. Which Device?	14
II.1. Edit <i>/etc/inittab</i>	14
III. Edit <i>/usr/lib/uucp/L-devices</i>	14
IV. Inform <i>init</i>	15
V. Inform the shell	15
VI. Test the serial line with <i>cu</i> :	16
VII. CU	17
VII.1. Summary of Components	17
VII.2. Configuring the IRIS for <i>cu</i>	17
VII.3. Notes	17
VIII. UUCP	19
VIII.1. Summary of Components	19
IX. Configuring the IRIS system for <i>uucp</i>	22
X. Edit <i>/etc/passwd</i>	22
XI. Edit <i>/usr/lib/uucp/L.sys</i>	23
XII. Edit the file <i>/usr/lib/uucp/L-dialcodes</i>	25
XIII. Editing the <i>/usr/lib/uucp/USERFILE</i>	26
XIV. Introduction to <i>/usr/lib/crontab</i>	27
XV. Editing <i>/usr/lib/crontab</i>	28
XVI. Uudemmon.hr, day, and wk	28
XVII. Common Problems	29

Appendix 4.2: Printer Spooling and Communications

This appendix is divided into two sections:

Section 1: This section gives additional information concerning printers.

Section 2: This section gives procedures and information concerning modems and communications.

This document was taken from the SGI System Administration Course and placed into the Hardware Course so that the Field Engineer would have more detailed information concerning printers, modems, and communication.

Section 1: Printers

I. Printer Spooling

- The *lp* spooling system is a general purpose system to spool output to any shared output resource.
- *lp* provides a queue that receives requests from users. *lp* then schedules the servicing of those requests by the output device(s).
- Queued requests (data) can be spooled to processes, for example, *nroff*.
- You may group devices into classes and assign a single queue to each class.

II. Commands Related to *LP*

- lpadmin** Used by the administrator to describe and assign printers, classes, and devices.
- Used to build (configures) or reconfigure the *lp* system.
 - Can be used to control the handling of requests through the system.
- lpsched** Controls the flow of data through the spooling system by scheduling *lp* requests to the appropriate devices.
- enable** Enables printers for use by the *lpsched* program.
- disable** Disables printers for use by the *lpsched* program.
- lpmove** Permits the administrator to move requests from one queue to another.
- lpshut** Begins a controlled shutdown of the *lpsched* program:
- Current jobs will be finished.
 - Outstanding requests will be saved.
- reject** Shuts 'off' a particular queue so it cannot accept new input.
- accept** Restarts a queue that has been rejected.
- lp** Sends a request to a selected printer or class of printers. Flags present a number of options to the user. A unique id is given each request.
- cancel** Permits a user to cancel a particular request based on the request id.
- lpr** A less sophisticated version of *lp*, *lpr* places requests on the cueue.
- lpstat** Prints *lp* status information. Options permit the status to return lists of destinations, class names & members, status lists of devices and the scheduler itself.
- mail** *lp* returns completion and some status information to the user via mail.

III. Initializing the system

- Set up a printer interface
- Modify */etc/rc*:
 - Uncomment the following four lines to permit starting the *lpd* daemon and scheduler:

```
if [ -x /usr/lib/lpd -a -x /usr/lib/lpsched ]
then
    /usr/lib/lpd;           echo " lpd"
    /usr/lib/lpsched;      echo " lpsched"
fi
```

- Be sure the *lp* account is located in */etc/passwd*.
- If running, stop the *lpsched* program:

```
/usr/lib/lpshut
```

- Execute the following, in fact, put this in */etc/rc* also (* = 1, 2, or 3):

```
/usr/lib/lpadmin -pptr0 -v/dev/ttyd* -mdumb (creates ptr0 as a cueue)
```

```
/usr/lib/lpadmin -dptr0 (assigns the default destination)
```

```
/usr/lib/accept ptr0 (informs lp to accept requests for ptr0)
```

```
enable ptr0 (informs lp ptr0 can receive data)
```

- Turn back on the *lpsched* program

```
/usr/lib/lpsched
```

IV. Adding new destinations

- As many destinations can be set up as there are devices (not just printers!).
- Use *lpadmin* to assign cueues and devices. For the most part, you must turn *lpsched* off before using *lpadmin*:

```
/usr/lib/lpshut
```

```
/usr/lib/lpadmin -pcueue -vdevice -mdumb
```

```
/usr/lib/lpsched
```

- Only the administrator (*su* or *root*) can use *lpadmin*.

V. Setting up classes

Classes are useful if you have several printers and you wish to collectively address them as one device. *lp* will direct output to whichever device within the class is free. Output can be directed to a class by using the *-d* option of *lp*.

- Classes are created with the *lpadmin* program:

```
/usr/lib/lpshut  
  
/usr/lib/lpadmin -pcueue -cclass  
  
/usr/lib/lpsched
```

- A device cueue (*ptr0*, for example) can be assigned to multiple classes.

VI. Setting up and using an interface program

There are times when output queues need special data control. "Interface programs" or scripts are used by *lp* to massage data in whatever ways are needed.

- Interface programs can be actual binary code or shell scripts, and is usually referenced during the assignment of a new queue.
- User defined interface programs are generally collected in the directory */usr/spool/lp/interface*.
- *lpadmin* is used to create the relationship between a queue and an interface program:

```
/usr/lib/lpshut
```

```
/usr/lib/lpsched -pname -vdevice -iIprogram
```

```
/usr/lib/lpsched
```

- For lots of nroff (or troff) output, use an interface file to actually do the *roffing so the system does not get bogged down with several concurrent *roff processes.
- Existing interface programs are available for use in the */usr/spool/lp/model* directory. the following page shows the listing for the script "dumb" - a simple interface script.

exit 0

VII. Using lpstat

lpstat provides any user with the information he/she may need in dealing with an output device. Options include:

- u Prints the status of output requests for given users
- v List the printers and their associated devices
- a Returns the acceptance status of any named destination
- c Returns the members of given class names
- d Print the system default destination for *lp*
- o Prints the status of output requests (provided job ids, class or printer names)
- p Returns the status of given printers
- r Returns the status of the *lp* request scheduler
- s Prints the status of most of the above
- t Print all possible status

SECTION 2: Modems and Direct Connection to Other Systems

UNIX provides two generic tools for interhost communications: *cu(1)* and *uucp(1)*. These tools assume you will be using a modem, so part of setting up for modem support is setting up a generic *cu* and *uucp* environment. This section addresses all the issues related to modem connections, including a hardware overview.

I. The RS232 hardware connection

The serial ports on the IRIS are designed to connect directly to *Data Communications Equipment* (DCE) such as modems, via a modem cable. A *modem cable* has pin 1 of the connector on one side connected to pin 1 of the other connector, pin 2 to pin 2, and so forth. The following table shows the pin definitions for the IRIS and for the modem cable.

```

box,center,tab(!); cB s s cI cI cI c c l. Modem Cable _ IRIS!Modem!Signal = 1!1!Chassis
ground 2!2!Transmit data 3!3!Receive data 4!4!Request to send 5!5!Clear to send 8!8!Carrier
detect 6!6!Dataset ready* 20!20!Data terminal ready 7!7!Signal ground

```

*Pin 6 is used only for *ttym2*.

Pin Definitions for Modem Cable

Connecting the IRIS directly to *Data Terminal Equipment* (DTE), such as terminals and printers, requires a different cable arrangement, a *null modem cable*. The following table lists the pin definitions for a null modem. The pin numbers that are shown separated by commas (,) should be connected together and also to the other pin or pins listed in the same row.

```

box,center,tab(!); cB s s cI cI cI c c l. Null Modem Cable _ IRIS!Terminal!Signal = 1!1!Chassis
ground 2!3!Transmit data 3!2!Receive data 4!8!Request to send/Clear to send 8!4,5*!Carrier
detect 6!20!Dataset ready*

```

*These connections may be necessary on the terminal side.

Pin Definitions for a Null Modem Cable

These pin definitions work with any serial device that complies with the RS-232C specification.

Most printers work with simpler cabling. If the printer is to be used via a modem, the following table should be used. Refer to your printer manual for more details.

box,center,tab(!); cB s s cI cI cI c c l. Simplified Null Modem Cable _ IRIS!Printer!Signal =
 1!1!Chassis ground 2!3!Transmit data 3!2!Receive data !4,5!Request to send/Clear to send
 6*!6]*,8,20!Carrier detect/Dataset ready 7!7!Signal ground

* Connect 8,20 of printer to 6 of IRIS for modem control.

Simplified Null Modem Pin Definitions for Printers

Most terminals do not require the various handshaking lines such as *clear to send* or *data set ready*, and will work with a three-wire null modem cable. The following table lists the pin definitions for the three-wire null modem cable.

box,center,tab(!); cB s s cI cI cI c c l. 3-wire Null Modem Cable _ IRIS!Terminal!Signal =
 2!3!Transmit data 3!2!Receive data 7!7!Signal ground

Three-wire Null Modem for Terminals

The following table summarizes the types of cables to use with different peripherals.

box,center,tab(!); cB s cI cI l l. Cable Usage _ Peripheral Cable = Modem!Modem cable
 Terminal!Null modem Printer!Simplified null modem

Summary of Cable Types

II. Which Device?

Any of the three ports (*/dev/ttyd1 - 3*) may be used with modems. For the rest of this section, *ttyd** will stand for any of the three ports.

II.1. Edit */etc/inittab*

Login as root or execute the *su* command.

/etc/inittab should be edited to permit or not permit *getty* to run on the port. For dial-in lines, you must have *getty* running. For dial-out lines, you have to turn it off. Because of this, you must either have two modems dedicated to dial-in and dial-out or have a single modem and modify */etc/inittab* to meet the current need.

To use a dial out modem, find the entry for *ttyd2*. Edit this line to read:

```
d1:x:respawn:/etc/getty ttyd* dx_9600
```

Since this line contains an *x* in the second field, logins are disabled by preventing a *getty* process from being run for the port.

To use a dial in modem, modify the *ttyd** line to read:

```
d1::respawn:/etc/getty ttyd* dx_9600
```

Because no *x* is in the second field, *getty* will be spawned or respawned to permit incoming logins via the modem.

III. Edit */usr/lib/uucp/L-devices*

This file contains line speed entries for each port and is used by *uucp* and *cu* to determine which device is associated with which type of connection and with what baudrate to service the port. The file is generally poorly documented so some detail will be covered here.

The following shows the default */usr/lib/uucp/L-devices* file.

<i>DIR</i>	<i>ttyd1</i>	<i>ttyd1</i>	4800
<i>DIR</i>	<i>ttyd2</i>	<i>ttyd2</i>	4800

<i>DIR</i>	<i>ttyd3</i>	<i>ttyd3</i>	<i>1200</i>
<i>DIR</i>	<i>ttyd3</i>	<i>ttyd3</i>	<i>300</i>
<i>DIR</i>	<i>xns</i>	<i>xns</i>	<i>xns</i>

Entries in this file contain four fields:

Type	The first field contains the type of connection. Use DIR for a hardwired connection or ACU for an automatic calling unit. An Ethernet XNS line is always direct (DIR).
line	This field is the device name of the port (line) to which the modem is connected. Use ttynn for a direct line, cu10 for a line connected to an ACU, and xns for Ethernet XNS.
call-device	If the ACU keyword is specified, this field contains the device name of the ACU (ttyd* in our example). Use xns for Ethernet XNS. Otherwise the contents of this field is used only as a place holder.
4800	This is the baud rate of the connection. Each baudrate requires it's own <i>L.devices</i> entry. You may use a given port for more than one baudrate. Use xns for Ethernet XNS.
Protocol	This field is not used.

To configure the */usr/lib/L-devices* file for a null-modem or modem, Add one of these two lines for device *ttyd**:

```
DIR ttyd* ttyd* baudrate (for null-modem)
ACU ttyd* ttyd* baudrate (for modem)
```

IV. Inform *init*

init must be notified to re-evaluate the change to */etc/inittab*. Type the command:

```
telinit q
```

This command causes *init* to read */etc/inittab* and kill (or spawn) the *getty* for that port.

V. Inform the shell

*cs*h users:

```
set port=/dev/ttyd*
```

sh users:

```
port=/dev/ttyd*
```

Give all users permission to access the port:

```
chmod 666 $port
```

Change the owner of the device to *root* and its group to *sys*:

```
chown root $port  
chgrp sys $port
```

VI. Test the serial line with *cu*:

```
cu -sbaudrate -ldevice phonenumber
```

Configured thus far, a modem can be used in the normal fashion for typical uses, such as *cu*, a remote terminal, or even to transfer files using packages like *kermit*. More sophisticated applications using *uucp* require additional system configuration.

VII. CU

CU provides a simple on-line interface permitting general interactive access and file transfer communications with other UNIX (and non-UNIX) machines.

VII.1. Summary of Components

Directories and files related to *cu*:

<i>/bin/cu</i>	Call Unix - the interactive calling program
<i>/dev/ttynn</i>	Any tty device for hardwire (direct) communications or modem support
<i>/usr/lib/uucp/L-devices</i>	Contains port options for <i>cu</i> and <i>uucp</i>
<i>/etc/inittab</i>	In this context, <i>getty</i> prevents an unwanted <i>getty</i>

VII.2. Configuring the IRIS for *cu*

Configuring the IRIS for the support of *cu* is relatively simple:

- Follow the procedure for setting up a modem or Null-modem
- Update */usr/lib/uucp/L.devices* to include appropriate port information
- Update */etc/inittab* to turn off the *getty*
- Notify the system you've changed it:

```
telinit q
```

- Run *cu*

VII.3. Notes

cu is fairly well documented in terms of actual use. Here are a few additional notes that might be helpful:

- Be sure your port (*/dev/ttyd**) has the mode set to 666. This is a very common oversight.
- If possible, enter *cu* from a window manager port. If you should hang *cu* for any reason, another port can be used to kill the process.
- Check */usr/spool/uucp* for a lock file and remove it (normally, you would set up an entry in the */etc/rc* file to automate this). Lock files (LCK-something) will lock out any attempts to use a locked port.
- When *cu* is invoked, it scans through */usr/lib/uucp/L.devices* for a port matching the provided description. It is good practice to place the most used ports highest in the list.
- *cu* does not attempt any sophisticated error checking. Using *cu* to transfer file over the phone lines is always risky business.

VIII. UUCP

uucp is a series of related programs that permit UNIX to UNIX communication for automatic and direct file transfer. Mail, networking, and indirect file transfers exemplify common applications of *uucp*.

VIII.1. Summary of Components

The following is a summary of directories and files related to *uucp* broken down into reasonable groups.

<i>uucp</i>	Unix to Unix copy utility
	Dirs/files
	What
	<i>/usr/lib/uucp</i> Control and data files for <i>uucp</i> .
	<i>/usr/spool/uucp</i> Contains the logfiles and data files for <i>uucp</i> .
	<i>/usr/spool/uucppublic</i> File pool for incoming and outgoing public files
	<i>/usr/lib/uucp/L.sys</i> List of system names and call times
	<i>/usr/lib/L-cmd</i> List of commands for <i>uuxqt</i> to execute
	<i>/usr/lib/uucp/L-devices</i> Device codes and speeds
	<i>/usr/lib/uucp/L-dialcodes</i> List of phone numbers in L.sys
	<i>/usr/lib/uucp/SYSTEMNAME</i> Name of this system
	<i>/usr/lib/uucp/USERFILE</i> List of users and pathnames
<i>uucico</i>	Located in <i>usr/lib/uucp</i> , this program is used by <i>uucp</i> to copy files in and out and between systems.
<i>uuclean</i>	Located in <i>usr/lib/uucp</i> , this program is called by <i>uucp</i> to clean up the spool directory.
<i>uulog</i>	Returns logged <i>uucp</i> user and/or system activity
	<i>/usr/spool/uucp/LOGFILE</i> Human readable file indicating the time, status and <i>action</i> performed for every <i>uucp</i> request.

Appendix 4.2: Printer Spooling and Communications

uname	Returns systems names contained in <i>L.sys</i> <i>/usr/lib/uucp/L.sys</i>
uustat	Returns status or cancels uucp commands <i>/usr/lib/L_stat</i> System Status File <i>/usr/lib/R_stat</i> Request status file
uusub	Monitor uucp sub-network traffic & connections <i>/usr/spool/uucp/SYSLOG</i> Contains a log of all uucp traffic <i>/usr/spool/uucp/L_sub</i> Connection statistics <i>/usr/spool/uucp/R_sub</i> Traffic statistics
uuto	Sends a public file to another system. Calls <i>uucp</i> .
uupick	Interactively accepts or rejects files transmitted to #user. <i>/usr/spool/uucppublic</i> The location where publically received files are kept.
uux	Gathers files (if requested) from remote machines, performs an assigned task and redirects the output locally or to another remote system. uuxqt Located in <i>/usr/lib/uucp</i> , this program is used by <i>uux</i> to perform remote execution.
uusub	Defines and monitors uucp sub-networks
demons	<i>uudemon.day</i> , <i>uudemon.hr</i> , <i>uudemon.wk</i> are daemons responsible for sending and receiving <i>uucp</i> files on a time basis.

Additional Files Additional files likely to be found in */usr/lib/uucp* include:

ORIGFILE Contains a list of originating systems or users for which the local IRIS is willing to forward files.

FWDFILE Contains a list of systems to which the local IRIS is willing to forward files.

SYSLOG This is a compacted file (by uudaemon.day) and contains a list of activity.

ERRLOG Receives error information. Use for debugging.

IX. Configuring the IRIS system for *uucp*

uucp will run on a direct (null-modem) or modem connection. Bell 102 and 212 compatible modems are typically used at up to 1200 baud. The new 2400 and 4800 baud modems can also be used. *uucp* may also be used over the Ethernet connection.

Summary of Procedure

Configuring an IRIS for *uucp* requires the following:

- Edit the file */etc/passwd* to ensure the *uucp* and *uucpadm* accounts are in place (Watch your security!)
- Edit the file */usr/lib/uucp/L-devices* to create the appropriate ports.
- Edit the file */usr/lib/uucp/L.sys* file to include all the systems to which you wish to communicate.
- Ensure your */etc/sys_id* file is up to date.
- Edit the */etc/inittab* file to turn off unwanted gettys (outbound).
- Edit the */etc/gettydefs* file to set up a *uucp* login prompt.
- Edit */usr/lib/uucp/USERFILE* to control user access within your system (security is important!)
- The directory */usr/spool/uucppublic* should already be set up. If not, create it.
- You may want to edit */usr/lib/crontab* to time trigger a *uucp* event, like to send mail at two a.m. and to trigger some house cleaning scripts like *uudemon.hr*, *uudemon.day*, and *uudemon.wk*
- Optionally Edit the */usr/lib/uucp/ORIGFILE* and *FWDFILE* to control forwarding if you wish to be network link.

X. Edit */etc/passwd*

On delivery, the IRIS */etc/passwd* contains the following two lines:

```
uucp:*:3:5:UUCP Login Account:/usr/spool/uucppublic:/usr/lib/uucp/uucico
uucpadm:*:8:8:UUCP Administration:/usr/lib/uucp:
```

These are all you need to get started. However, a few notes:

- The password (*) is preset to prevent logging in as a real account. If you actually use a password, any system dialing in knowing the password is to be considered "in the net." It is better to leave this the way it is.
- */usr/lib/uucp/uucico* is the program that is invoked by one of several sources:
 - cron* or another system daemon
 - by one of *uux*, *uucp*, or *uuxqt*
 - a remote system

It is for the last reason that *uucico* needs to be invoked as the "shell" for the uucp account. *uucico* will put incoming files into the *uucppublic* directory.

XI. Edit */usr/lib/uucp/L.sys*

This file contains information about sites to which *uucp* can communicate. The following shows */usr/lib/uucp/L.sys* as it is shipped.

/usr/lib/uucp/L.sys

```
# gendir1 direct RS-232 connection master (don't supply getty for ttyd1)
# uucpco is an account for "the Company" that is granted more liberal access
# permissions in USERFILE than uucp (should have different UID in /etc/passwd
# for security)
# speeds above 4800 are not recommended
gendir1 Any ttyd1 4800 ttyd1 ""
# gendir2 direct RS-232 connection slave (supply getty for ttyd2)
# uucpco is an account for "the Company" that is granted more liberal access
# permissions in USERFILE than uucp (should have different UID in /etc/passwd
# for security)
# speeds above 4800 are not recommended
gendir2 Never ttyd2 4800 ttyd2 ""
# ACU dialing with Ventel modem by hand (don't supply getty for ttyd3)
# (ttyd3 should be DIR in L-devices)
genphone Any ttyd3 1200 ttyd3 ""
# generic xns ethernet connection
# uucpco is an account for "the Company" that is granted more liberal access
# permissions in USERFILE than uucp (should have different UID in /etc/passwd
# for security)
works2 Any xns xns xns ""
# example entry for tcp/ip as referenced in the ud manual page.
# excelan Any ttyT8 9600 ttyT8 "" excelan assword "^M^M" in:-EOT-in: uucp assword: s
# xln Any ttyT8 9600 ttyT8 "" xln assword "^M^M" in:-EOT-in: uucp nassword: sesame
```

The `/usr/lib/uucp/L.sys` file contains entries representing remote systems that can be dialed when using `uucp`.

Entries in this file contain the following fields:

```
system_name  time  device  class  phone  login
```

<i>system_name</i>	The name of the remote system. Example: <i>shasta</i> - a system that kindly volunteered to pass on <code>uucp</code> data to other systems (commonly referred to as a "node").
<i>time</i>	When the system should be called. Options for this field are detailed in the <i>UNIX programmer's reference manual</i> - example: <i>Any, never, MoTuTh0800-1730</i> .
<i>device</i>	The device name to be used for the call. Generally this will be <code>ACU</code> for a modem or <code>ttyd*</code> for a hardwired port.
<i>class</i>	Typically the baud rate, or <i>xns</i> .
<i>phone</i>	A two part phone number: <i>prefix number</i> where <i>prefix</i> is a number (usually a city and area code) listed in the <code>/usr/lib/uucp/L-dialcodes</code> file. <i>number</i> will be the actual phone number of the remote machine. If hardwired, use the port device (<code>ttyd*</code>).
<i>login</i>	Login information prompt. This is the most confusing of all the fields but if you remember that <code>uucp</code> (<code>uucico</code>) logs into the remote machine as a user, it helps. This field is where you tell <code>uucp</code> how to log into the remote system.

The information is given as a series of fields and subfields:

```
expect send
```

`expect` is the field expected from the other system (like `login`). `send` is what we send in response (like `uucico`). Fields are separated by spaces. Colons are actually part of the strings!

Subfields are defined by the use of the (-):

```
expect-send-expect-send.....etc
```

In this example, the first `expect` waits about 20 second for a response. If no response, then send the `send`. The next `expect` will wait for a response and, if none, will send the next `send`. Usually you give up about this time. For Example:

```
BONZO Any xns xns xns "" \r ogin:-BREAK- ogin: uucp password:
password
```

This will try uucp over the net to BONZO. The "" represents a null response (nothing) is expected. \r (carriage return) is sent. Login is expected, and if not received in 20 seconds, a break is sent. If login is received, uucp is sent as an account request. 'Password:' is expected and then *password* is sent. If this fails, *uucico* will be triggered by *cron* at another time and will try again.

See subsection entitled "2.3.4 System File" in *uucp Administration* in the *UNIX Programmer's Manual, Volume II* for more information.

XII. Edit the file */usr/lib/uucp/L-dialcodes*

L-dialcodes contains the telephone prefixes referenced in the *L.sys* file. It's format is very simple:

city prefix

city can be anything really, but is usually a city name like "sf" or "bos." *prefix* is usually the area code for the city. The city field must reflect *exactly* the name used in the *L.sys* file. If you use more than one names for a given location, simply add a line for each name:

```
sj      408
sf      415
dc      202
oc      64-
pit     12=
```

The last two entries indicate special handling:

- Dial any number preceding the -, and then wait for a dial tone before dialing the remainder of the number in the *L.sys* file. This is used for special access numbers.
- = Same as -, except it waits a fixed 5 seconds before dialing the remaining *L.sys* number. This is good for delayed switch responses.

XIII. Editing the */usr/lib/uucp/USERFILE*

This file is used to restrict logins by other systems to specific directories. If a remote system dials in, *USERFILE* must have an entry for the login or a null field. *USERFILE* has four basic fields:

```
login,nodename c pathname
```

- login* The login account underwhich the remote system logged in. If no entry is found with this name, *any null field line will be used*. Normally, this will be *uucp*. A comma immediately follows this field.
- nodename* This is the name of the remote system. It immediately follows the comma. If no name is given, *any system will be accepted*.
- c* A literal letter 'c' following a space after *nodename* informs uucico that it must call back the remote system to verify it's identity. This is an optional field. Security note - any number of systems given the same name could dial in and lie about who they are.
- pathname* any pathname to which you wish to restrict the write permissions of the login account. Normally, you restrict the remote to */usr/spool/uucppublic* to prevent clever "rummaging" programs. More than one pathnames may be specified by separating them with spaces.

Example:

```
uucp,BONZO c /usr/people/stuff
uucp,TWIT /
, /usr/spool /usr/people/stuff
u,m /usr/xyz /usr/spool
u, /usr/spool
```

The login name, nodename, and c are optional. The third example implies any uucp login and any system id can write files in the */usr/spool* and */usr/people/stuff* directories. The fourth and fifth examples indicate that a system dialing in as 'u' is limited to */usr/spool* unless it's name is m, then has additional access to */usr/xyz*. */usr/people/stuff* would also be available because of the third example.

XIV. Introduction to */usr/lib/crontab*

/usr/lib/crontab is used by the *cron* daemon to schedule and execute events based on specified time periods.

The following listing shows a typical *crontab* file:

```

#Minute      Hour      Day      Month      Weekday Command
0,10,20,30,40,50 * * * * /usr/lib/atrun
0      4      *      *      *      calendar -
0      5      *      *      6      find / -name core -o -name\
      dead.letter ) -a -atime +7 -exec rm '{}' ';'
35     7-22   *      *      *      /usr/lib/uucp/uucico -r1
0      1      *      *      0,4    cd /usr/spool/uucp;umask 7033;\
cp LOGFILE oLOGFILE;cp /dev/null LOGFILE
1      1      *      *      5      cd usr/spool/uucp;umask 7033;cp\
SYSLOG oSYSLOG;cp /dev/null SYSLOG
2      1      *      *      5      cd /usr/spool/uucp;umask 7033;cp\
AUDIT oAUDIT;cp /dev/null AUDIT
3      *      *      *      5      cd /usr/spool/uucp;umask 7033;cp\
ERRLOG oERRLOG;cp /dev/null ERRLOG
4      1      *      *      *      cd /usr/adm;umask 7033;cp cronlog\
OLDcronlog;cp /dev/null cronlog
5      *      *      *      5      cd /usr/adm;umask 7033;cp sulog\
OLDSulog;cp /dev/null sulog;chmod go-rwx OLDSulog
6      1      *      *      5      cd /etc;umask 7033;cp wtmp OLDwtmp;\
cp /dev/null wtmp
0      *      *      *      0,6    su adm -c "/usr/lib/sa/sal"
0      8-17   *      *      1-5    su adm -c "/usr/lib/sa/sal 1200 3"
0      18-7   *      *      1-5    su adm -c "/usr/lib/sa/sal"

```

This file is described in the *cron(1M)* man page.

cron reads this file every minute. If something has changed, *cron* will schedule in that change.

Fields are separated by tabs or spaces. The first five are:

minute(0-59) - schedules events every n minutes.

hour(0-23) - schedules events every n hours.

day of month (1-31) - schedules events on a given day or range of days.

Month of year (1-12) - schedules events on a monthly basis

Day of week (0-6) - schedules events daily (0 = Sunday)

The digits indicating the time range can take the forms:

A number representing the appropriate hour, minute, day, etc.

Two numbers separated by a minus (3-6) indicating a range of time
 A list of numbers delimited by commas representing a list of times
 An * meaning all possible times

The Sixth field is a string executed by the shell at the appropriate time.
 % becomes newline, \ becomes .

XV. Editing */usr/lib/crontab*

uucico is used in conjunction with *crontab* to trigger the uucp events on a timely basis. This is not documented! An example might be:

```
00    02    *    *    *    /usr/lib/uucp/uucico -r1
30    02    *    *    *    /usr/lib/uucp/uucico -r1 -sBONZO
30    05    *    *    *    /usr/lib/uucp/uucico -r1 -sTWIT
```

This example would cause *uucico* to call any system for which files are cued at 2 a.m. and send the files. The next two entries reach out to two systems at 0230 and 0530 to cause those systems to send us anything they have. The timing should be ordered to permit the other systems to collect replies before returning whatever to our system. Carefully timed sequencing gives 1-day or less turn-around!

The *uudemon.hr*, *uudemon.day*, and *uudemond.wk* files (see next page) are designed to properly time uucp activities. In lieu of the above, use:

```
56    *    *    *    *    /usr/lib/uucp/uudemon.hr
00    04    *    *    *    /usr/lib/uucp/uudemon.day
30    05    0    *    0    /usr/lib/uucp/uudemon.day
```

XVI. Uudemon.hr, day, and wk

uudemon.hr:

```
#!/bin/sh
# ~(#)uudemon.hr 1.2
# 'perform every hour on the 56-minute mark'
PATH=/usr/lib/uucp:/bin:/usr/bin
cd /usr/lib/uucp
uucico -r1
uulog
```

uudemon.day:

```

#! /bin/sh
#      ~(#)uudemon.day 1.3
# 'perform once per day at 0400 hours'
PATH=/usr/lib/uucp:/bin:/usr/bin
cd /usr/lib/uucp
uuclean -p -m -n168 >/dev/null 2>/dev/null
uuclean -d.XQTDIR -p -n72 >/dev/null 2>/dev/null
uustat -c168 >/dev/null 2>/dev/null
cd /usr/spool/uucp
mv LOGFILE temp
uniq -c temp >> Log-WEEK
rm temp
uusub -call -u24
cd /usr/spool/uucppublic
find . -type f -mtime +30 -exec rm -f {} ;

```

uudemon.wk:

```

#! /bin/sh
#      ~(#)uudemon.wk 1.3
# 'perform once per week, Sunday at 0530 hours'
PATH=/usr/lib/uucp:/bin:/usr/bin
cd /usr/spool/uucp
rm -f o.Log-WEEK* o.SYSLOG*
mv Log-WEEK o.Log-WEEK
(date; echo =====) >> o.Log-WEEK
mv SYSLOG o.SYSLOG
>> SYSLOG
pack o.Log-WEEK o.SYSLOG

```

XVII. Common Problems

uucp is somewhat prone to difficulty because of the number of supportive files and directories. It is also sensitive to the goings-on of other systems!

Common problems include:

- Remote program execution may fail because the remote system does not have the command in CMDLIST or uuxqcmds or is not compiled in uuxqt.
- A file may not be transferable because:
 - The source file is not readable (permissions)

- The path is protected or wrong
- A destination file exists and is not writable
- The destination directory is not writable
- Pathname of destination is not in USERFILE
- Communication may not be possible because:
 - Baud rates are incorrect
 - Bad cabling or modem connection
 - L.sys and L-devices do not agree
 - ACU (auto-dial modem) may not be compatible with your uucico. Vanilla UNIX supports only Bell modems! The IRIS supports VEN-TEL modems. HAYES modems are prone to give you difficulty.
- Rmail is missing! You can usually recover this by linking mail with rmail. The IRIS is shipped with rmail.

Read *mail(1)* for an explanation of the routing of mail to another system via *uucp*.

A new *uucp* section in the Programmer's Reference Manual Vol IIB is useful and is good reading.

Appendix 4.3: Running a Demonstration Program

This appendix gives information and procedures for running the IRIS demo programs.

The document was meant for training new sales persons and is given to the hardware student as a simple reference guide for running demos.

GIVING A DEMONSTRATION ON AN IRIS SUPERWORKSTATION

Demo Software Release 3.4

This manual explains how to operate the demonstration programs for the Silicon Graphics IRIS Superworkstation. It includes simple operating instructions, descriptions of the IRIS features, and sample questions and answers.

The manual is for Silicon Graphics sales, marketing and executive personnel. It is not for distribution outside the company.

The software described works with both standard IRIS workstations (with 32 bitplanes) and portable "bubble" systems. In most cases, programs that do not use Z-buffering run identically on both types of IRIS.

Before the demo--Booting the IRIS

If the machine is already turned on, skip this section.

This section tells how to start (boot) operations on an IRIS workstation. Bear in mind that the IRIS is a complicated system and cannot be powered up and down like a toaster.

Verify that all cables are connected correctly before plugging in the IRIS. Verify that the power source is a 115V, 3-wire circuit with at least 10A of available current. Set the power switch on the front panel up (OFF). Plug in the IRIS. Set the power switch on the front panel down (ON). Set the power switch on the monitor ON.

The keyboard will beep, the pilot lamp will light, and, in a few moments a configuration message and prompt will appear. Wait about a minute for the disks on the machine to warm up. Then type *b* to boot the system. ("Boot" is short for "bootstrap", where we think of the computer as pulling a program in by its own bootstraps.)

The system will bring up UNIX. Note the messages that go by. They explain what the system finds as it explores its peripheral environment -- as it tests to see which disks, tape, ethernet, and memory options are installed. The IRIS will present a prompt, *#*, which tells you to continue.

Type *multi* to put the system in multiuser mode. In a moment the system will present you with a time and date and ask if it's correct. Either type *y* (saying "It's close enough") or *n* (saying "Let's do it right"). If you say "No," the IRIS will prompt for a new time. Type it as *mmddhhmmyy*: month, day, hour (24 hour clock), minute, last two digits of the year. The year is optional. You must acknowledge the new time as correct.

Then the system will ask you if you wish to test the file system. You should do so. The system will then examine the sections of the disk to see if there are any damaged files. On the screen, you will see this message:

Checking file systems for consistency:

and proceed to do just that.

The program will sometimes find an error and print a message of the form:

such and such wrong, fix it?

Type **y**. The program will proceed. Occasionally, a problem is found which affects UNIX so much that it must fix force a reboot. You'll have to start over again.

Finally, after all this, it will give you a login prompt. Let's continue from here.

The Preliminaries

Dim the lights in the room. It reduces screen glare and makes the screen appear brighter.

Move yourself to the side of the monitor. Let the client have an unobstructed view of the screen.

The machine should have login prompt which says:

madonna login:

where *madonna* or some other name is the name of the machine. If this prompt is not showing, there are two possibilities. One possibility is that you have not completed the entire boot (power up) sequence. The other possibility is that someone else is already logged into the system. Type **logout** or **exit** to see if you can finish the old session.

Log into the IRIS with the name, 'sales.' To do this, type **sales** where you see the login prompt. You should see:

madonna login: sales

Always terminate a command by pressing the carriage return. Wait a few moments while the system logs you in. When the IRIS returns a prompt message, typically a few words and numbers followed by a percent sign, %, it is ready to accept your commands. Your screen may look like this:

madonna1%

Then type **startup** and wait a few moments more.

Q: What's happening here?

A: A simple program (a *shell script*) is setting up the demo environment. It is starting the window manager, building a common color map for all the demos, and calling up a background.

Q: What are those funny things in the background?

- A: Just background decorations. We call them "Haerberli blobs" after the programmer who put them in.

The Mouse--A Brief Interlude

The mouse is the primary input device for a product demonstration. The better you handle the mouse, the better the demo.

You should first understand how you use the mouse to control the window manager. You can start reading either the guide entitled *Mouse Aerobics* or the chapter entitled 'The Default User Interface to the Window Manager', in the *Multiple Exposure: The IRIS Window Manager* section of the *IRIS User's Guide*. Before continuing, be sure to read one of these two documents and experiment with the window manager.

Points to Make Throughout the Demo

- 1) It is a workstation--both processing unit and graphics display.
- 2) Motorola 68020 32-bit CPU.
- 3) Runs UNIX System V with some 4.2 extensions.
- 4) All demonstration programs written in C and Fortran. No microcode nor assembler required.
- 5) Geometry Engine--custom VLSI for graphics.
- 6) 3-D with perspective and clipping.
- 7) Color--up to 4096 colors with look up table, up to 16.7 without look up table.
- 8) Multiple Exposure window manager.
- 9) Graphics features: depthcuing, Gouraud shading, z-buffering hidden surface removal.

Typical Order of Commands in a Demonstration

- 1) **startup** -- startup the window manager
- 2) **menu** -- the demo pop-up menu
- 3) **pauls_tools** -- many 2-D windows: talk about UNIX, colors, workstations
- 4) **jet** -- wire frame 3-D model: 3-D, perspective, depthcuing, Geometry Engines
- 5) **light** -- spinning cube in space: filled polygons
- 6) **insect** -- crawling spider: robotics, simulation, Graphics Library, not microcode
- 7) **surfcar** -- surface editor: curves, matrix operations have VLSI support
- 8) **zshadecar** -- Gouraud shade surface with hidden surface removal: Gouraud shading, z-buffer hidden surface removal, CAD/CAM, productivity
- 9) **exit** window manager and type **standalone**

- 10) `dog -i airshow` or `shadow -i airshow -- show flight simulation`

The Care and Feeding of Demos

Demo Menu

The principal window manager demos may be picked through an optional menu. To start the menu, simply type `menu`. First select the menu window. Press and hold down the *right* mouse and a pop-up menu will appear. Some menu entries specify categories of demo programs and have rollover menus with lists of individual demos. When the program you wish to run is highlighted, release the button. You may start several programs from here.

The demo programs fall into three main groups:

pauls_tools are a set of 2-D demos. Most are quite simple. They help show how windows work and help the demos get off to a quick start. They provide valuable lessons in color map manipulation.

3-D object manipulators move 3-D objects around. They include *arch*, *flow*, *heme*, *jet*, *shuttle*, *cube*, *curve*, *light*, *insect* and others. The IRIS performs simultaneous manipulation of different 3-D objects in independent windows.

Shaded object and curved surface demos show off other functions of the IRIS. They are discussed together because they work well with each other and their functions are intimately related in real applications.

Pauls_tools, 2-D Window Manager Demos

The demo package includes a collection of demos called *pauls_tools* that help show how the window manager works, and provide instruction on the color handling capabilities of the IRIS. Some of the programs provide user interaction; others just sit there.

Source to the programs are delivered with each IRIS as unsupported gifts. (See */usr/people/gifts*.) They provide simple programming examples for new users of the system.

Type *pauls_tools* into the textport to start this collection of demos. The underline in the middle of the command is important. The command is a shell script that calls several different window manager programs. At first, try the package as it is presented to you. If it doesn't fit your presentation style, edit the *pauls_tools* file and alter it to bring up your favorite demos. If you wish to run only particular programs, you may call them one by one, either through the demo menu or by typing in their names.

Use the mouse to place and size the windows produced. None of these tools is by itself worth a major effort, but the whole combination gives a good idea of the capabilities of the window manager. Only take a few minutes to show these programs.

Here is a brief description of the tools. The description is divided into four categories: backgrounds, color map tools, pixel access programs, and all other programs.

Backgrounds

bckgrnd is a non-interactive textured "wallpaper" with Art Deco designs. The background fills all available space which is not covered by a window.

texback creates several other non-interactive textured "wallpapers." Call *texback* with a number as an argument. For example, type *texback 2*.

Color Map Tools

showmap is a non-interactive window filled with up to 1024 little squares. It displays the current colors in the lookup table, counting from color 0 at the lower left corner up to the largest color index at the top right. I like to refer to it as the IRIS color "palette."

showramp is an interactive program with three Gouraud shaded rectangles. The rectangles are smoothly colored using adjacent (physically and chromatically) color indices, which form a color ramp in the color lookup table. The color indices used for the corners of the rectangles in *showramp* can be changed by the *left* mouse.

The *gamma* program uses a technique called gamma-correction to change the "smoothness" of a color map. The physical characteristics of the CRT phosphor, the ambient lighting, and the vision and preference of the viewer affect the perception of the color map. A strictly linear color ramp often appears too dark, especially towards the middle of the ramp. Gamma allows the increments from color to color to be exponential, and thus, the middle of the color ramp can be brighter or darker than would appear in a linear color ramp.

The *gamma* command accepts a single floating-point argument. A value of 1.0 creates a strictly linear color ramp. Values above 1.0 increase the brightness in the middle of a color ramp; values below 1.0 decrease the brightness. Values between 1.0 and 3.0 are typical.

The program does not immediately affect the color map. It simply stores its argument in a file called *.gamma* in your home directory. This file is referenced whenever color ramps are made in the future with other demonstration programs, such as *makemap* (See below). The *gamma* program itself does not open a window.

makemap remakes the color map for the demos. If you run a nonstandard demo that makes its own color map, use *makemap* to restore things later.

randmap makes a random color map. The results are usually not too attractive.

cedit is an interactive color editor in a graphics window. After the *cedit* window is brought up, attach input focus to it. To select a color index for editing, the cursor is moved outside the *cedit* window itself. The *left* mouse is pressed and released, and the color index of the pixel under the cursor is chosen. The chosen color index is painted on the "sample chip" on the right of the *cedit* window, and the "color sliders" are set to the red, green, and blue values of that color. To change the intensity of the color value of the sample chip, press the *left* mouse down while the cursor is located on one of the color sliders and move the sliders up and down. Try picking a color from *showmap* and editing it with *cedit*.

interp displays and alters a region of the color map. *Interp* is used to make color ramps for depthcued and smooth-shaded objects. *Interp* makes a color ramp of any size between any two color indices, interpolating the red, green and blue intensities of all indices between the two prescribed extremes. Two squares show the extreme colors of a color ramp and a long rectangle shows the intermediate color range.

You can use *interp*, in conjunction with *showmap* and *cedit* to create your own gamma-

corrected color ramps. First attach to the *cedit*. Use *cedit* to edit any two color indices displayed on *showmap*. Now, detach from *cedit* and attach to *interp*. Point to one of the extreme colors which you edited on *showmap* and press the *left* mouse button. Point to the other extreme color and again press the *left* mouse button. The colors will appear as two large squares in the *interp* window. (If you should make a mistake while choosing extreme colors, for example, the cursor is over an incorrect color index, then continuing picking colors with the *left* mouse until both extreme colors are visible in the *interp* window.) Now when you press *middle* mouse, the system will interpolate between the colors you have chosen and place the new values into the color map.

Pixel Access Programs

mag is an interactive program that magnifies or enlarges a pixel region on the screen. Attach to the *mag* window. Place the cursor on some image on the screen, then press and release *left* mouse button. A magnified image of the area around the cursor will fill the *mag* window.

Exercise: Magnify the shadow beneath a Haeberli blob.

paste draws any of several pixel images on the screen. Its result depends upon the state of the color map. For example, type *paste imageslib/cat.bw* to draw a picture of a cute, little kitten in shades of a color. Using *interp* can correct the color map and make a good black-and-white likeness. Practice with this one to avoid giving your prospects psychedelic experiences.

Other tools

keyboard is a 2-D model of the IRIS keyboard. If the window is selected for input, it will read the keyboard output and darken all the keys held down. Do this while explaining how the IRIS keyboard can be used as an array of buttons that can be either polled or queued.

mouse is a simple image of the mouse. When selected, the program follows the mouse with its image.

clock is a non-interactive analog clock face.

dbstars is a double buffered, perspective view of a star field. When the window is selected, the field can be rotated.

worms is a standard UNIX game that normally runs on a cursor-controlled terminal. Three brightly-colored planaria chew away on a field, eating all before them. The program is not interactive, but it (as well as the clock) shows the system maintaining multiple graphics processes.

Before going on to bigger demos, kill off most of the windows. Keep *cedit*, *showmap* and *interp* around so you can play with the color map. Some like the clock placed in a corner somewhere. These consume very little CPU time. Keep them if you choose, but bear in mind whenever the screen is redrawn, they use up time. Since *paste* and *mag* are drawn one pixel at a time, they can slow the redraw considerably. *worms* consumes a lot of computer power, so kill it before continuing.

2-D Hype

A few of the programs supplied provide 2-D advertising. These aren't demos per se, although they do show off raster fonts nicely. Many of these programs were shown in the

advertising videotape. See the man pages for detailed instructions on how to run these programs.

features and *partners* are text windows listing IRIS features and Geometry Partners. *quote* shows a quote about productivity from an IBM survey. *graph* and *graph2* are graphs derived from the IBM response time study. The first shows improved individual productivity, the second shortened project schedule. *mini* is a window with a blow-up from *graph*.

Exercise: Run *features*. Verify that you recognize each feature, understand its application, and can answer simple questions about it.

3-D Object Manipulation Demos

These are the fast 3-D demonstration programs. They illustrate fast 3-D transforms, polygon fill, depth-cuing, hidden surface removal, and overlapping windows. Many of the demos are double buffered for smooth motion.

Double buffer and z-buffer cannot work simultaneously. Thus, when showing smoothly shaded objects with z-buffer hidden surface removal, the programs use single buffer mode. Double buffering is used for most dynamic movement.

Arch, Flow, Heme, Jet and Shuttle

arch is a simulated architectural model. The eye is placed in a three-dimensional environment. One building (the white one) is always displayed. Several others can be displayed in wireframe or as solid objects. Once the window is selected for input, the user interacts with the program through both the keyboard and mouse. Each mouse button or combination of mouse buttons causes a different motion of the eye within the environment. The *f* key toggles the other buildings; *s* toggles the polygon fill on the buildings. Use the *m* key option to start a constant motion that continues even after the window is no longer selected for input (detached).

flow is an array of nearly 20,000 data points in a frame. The data was calculated at NASA-AMES on the Illiac-IV computer. It represents some solutions of the Navier-Stokes equations for flow in a turbulent fluid. *left* mouse controls viewer altitude and azimuth. *middle* mouse controls distance and twist (rotation on z-axis). *right* mouse brings up a pop-up menu with other controlling options.

heme is a depthcued molecular model comprised of 1022 points and 549 vectors. It is controlled by a special iconic pop-up menu. Hold down *middle* mouse to see the menu. To toggle an entry, position the cursor on the entry and press and release *left* mouse. Two menus appear. The left column of menu entries independently turns off and on each of the five parts of the molecule. The right column of menu entries control the transformations of the molecule. The menu entries in the right column are graphical icons. The bungalow icon (shaped like a house) resets the program back to the center (especially helpful if the molecule has been dragged out of sight). The stop sign icon exits the program and cancels the window.

The right-hand menu selects various motions and states of the object. A left mouse press changes the color of a menu entry and start that motion. Another *left* mouse press restores the initial state. Note that simultaneously specified opposite motions will cancel each other out, leaving no motion in the specified direction. The ENTER or *m* key causes any current motion to continue even after the program's window is detached. *heme* can also be operated through the dial and button box. Strike the *k* key (knobs) to turn operation over to the dials

and buttons.

jet is a wireframe model of a fighter plane. The image includes 660 vectors and can be displayed with constant intensity or depth-cuing. The jet is controlled with the *right* mouse which displays a pop-up menu. Some entries in the pop-up menu create sliders which rotate, translate and scale the object.

For *shuttle*, pressing one of the three mouse buttons enables rotation about one of the three axes. The amount of rotation is determined by the left to right motion of the mouse. Other combinations of mouse buttons allow X-Y translation controlled by the mouse and scaling at fixed positive and negative rates. The ENTER key enables and disables depth-cuing. Any alphanumeric key starts or stops the doors flopping open and closed. After the program is detached, the motion does not continue.

Light, Insect and Wave

light displays a floating cube and the shadow cast onto a grid beneath the cube. The shadow is simulated by an orthographic projection of the cube onto the plane. The sides of the cube are shaded to reflect their position with respect to the light source. The shaded cube is controlled by the mouse. When the *left* mouse is depressed, mouse movement causes the entire scene to move. When the *middle* mouse is down, mouse movement controls the orientation of the cube. When both the *left* and *middle* mouse buttons are depressed, moving the mouse causes the viewer to zoom towards and away from the shaded cube.

In the light demo, pressing the *right* mouse brings up a pop-up menu. The selections of the menu allow for automatic motion of the cube or the entire scene. Automatic motion allows you to focus attention on your audience, rather than the demo itself. A choice on the menu puts the window into information on 'MTV' mode is in the man page for light.

insect is a robotics and animation demonstration. It is a flat-facet shaded, six legged creature which walks on a plane in real-time. When attached to the insect window, the insect follows the location of the mouse (cursor) relative to the exact center of the monitor. Typing the f key alternately puts the insect in and out of follow mode. In follow mode, the view pans to follow the insect as it wanders. The Ctrl key and the Left Shift key zoom the viewer towards and away from the insect.

The insect is of particular interest to someone interested in robotics or visual simulation. As the insect moves, no two adjacent legs are simultaneously in the air, which guarantees stability.

wave shows a grid where the vertices are masses and the line segments connecting the vertices are springs. A mass can be perturbed, stretching the nearby springs on the grid. The entire grid can be set into motion. The wave demo does a lot of math.

In the wave demo, pressing the *right* mouse brings up a pop-up menu. Selecting the edit entry of the menu changes the color of the grid to red and puts the grid in edit mode. When in edit mode, you may move the mouse to pick points on the grid and press the *left* mouse button to displace a location of a point. The go entry of the pop-up menu changes the color of the grid to blue and starts the vibration of the grid. The reverse choice on the menu reverses the direction of the wave motion of the grid. The harmonics rollover menu provides a set of initial grid conditions.

light and *insect* use the IRIS graphics library call *backface* to remove hidden polygons

Curved and Smooth Images

The IRIS 2400 uses special microcode and hardware to draw smooth-shaded (Gouraud shaded), z-buffered images rapidly. The Geometry Engines enable real-time manipulation of splines. The surf (surface editors) and zshade (surface renderers) demonstration programs show off these features. The surf programs allow you to manipulate and modify the basis points (the data base) which determine a curved, wireframe surface. The zshade programs take the modified basis points and allows the surface to be filled using Gouraud shading and viewed with hidden surfaces removed. The surf and zshade programs are designed to be used in tandem. The surface design and rendering package allows you to design a bicubic patch in one window, pass it on to another, and draw a smooth-shaded, z-buffered image of the surface in the new window.

Surface Editor (surf)

See the extended document entitled "The Complete Guide to Surfcar."

Z-Buffered Gouraud Shaded Rendering (zshade)

NOTE: Hardware z-buffering is not available on machines which do not have 32 bits of image memory per pixel. Z-buffer hidden surface removal cannot be demonstrated on a bubble machine.

If a double buffered program not set up to work with the rendering demonstration programs is currently open, the system will not shade the object. It will print a warning message warning you about the situation. Kill all outstanding double buffer windows and try again to shade the object.

The surface defined by the modified points can now be rendered in another window as a solid surface. To spawn off the rendering window from the surface editor demo, see the surface editor document. The Gouraud shaded, z-buffered surface rendering program can also be executed directly without going through the surface editor stage to modify the data base. To bring up a surface rendering window with a pre-existing data base of basis points, type: zshadecar, zshadeabstract, zshadejet, or zshadeegg.

A wireframe outlining the basis points will appear. The data base of basis points is sent to the new window. The surface is subdivided into small quadrilaterals, and an illumination model is calculated. The surface is displayed as an array of dots. Detach from the surface editor window and select the rendering window for input. The surface is controlled with the right mouse button which displays a pop-up menu. Some entries in the menu create sliders which rotate, translate, and scale the object. Other entries in the menu allow the object to be viewed as points on the surface (vertices), splines, depthcued wireframe, Gouraud shaded without hidden surface removal, and Gouraud shaded with z-buffering hidden surface removal. There is a rollover menu entry which can be used to change the color of the surface.

revolution

The *revolution* program allows you to create a series of connected vectors and revolve them about an axis to create a three-dimensional surface. The surface can be Gouraud shaded and viewed as a solid with hidden surfaces removed. Hidden surface removal can be accomplished with either of two methods: the *binary space partition (bsp) tree* and z-buffering.

In one method, the binary space partition tree is calculated for the surface. The tree is traversed to determine which sides of the surface should be displayed. The bsp hidden

surface algorithm is done in software, so there is no minimum hardware (bitplane) requirement. The bsp tree is *not* an IRIS graphics library feature.

The other method is the general purpose z-buffer algorithm. The z-buffer used IRIS hardware, but it is directly callable from the IRIS graphics library. For less complicated models, the z-buffer algorithm is a slower form of hidden surface removal than the bsp tree, but the z-buffer algorithm requires less initialization time.

The operation of the revolution program consists of two sections: creating a form and shading that form.

When the window is started, there are a set of two axes. The surface is formed by revolving an outline around the vertical axis. To enter the vertices of that outline, move the cursor to the desired location and press the *right* mouse. Repeat this until you have a desirable outline. Then press the *middle* mouse to start the form by revolution and calculations of the binary space partition tree. The amount of time spent making the tree depends on the complexity (number of vertices) in your form. When the tree is finished, the cursor returns, and the shape is displayed in a depthcued wireframe.

To see the pop-up menu, press and hold the *right* mouse. The Color menu entry has an arrow to the side to indicate a rollover menu. The Color rollover menu has a selection of colors for the surface. The Rotate entry brings up sliders and puts the manipulation of the surface under slider control. Zbuffer draws the surface with Gouraud shading and the z-buffer hidden surface scheme. If there are any windows currently running in double buffer mode, the program puts up an error message stating there is a double buffer window outstanding, since double buffering cannot be done at the same time z-buffering is. Zbuffer is also not offered on the bubble machine.

The Gouraud menu command draws the surface with just the Gouraud shading and no hidden surface removal. BSP Smooth draws the surface with Gouraud shading and the binary space partition tree hidden surface removal scheme. This does not require any special states or hardware and can be run with double buffered windows or on the bubble machine. BSP Flat draws the surface with the BSP tree hidden surface scheme but with flat shaded polygons (one color per polygon).

When the sliders are displayed, they can be used to control the view of the object. Move the cursor into one of the three slider regions, which correspond to x, y and z axes. Press and hold the *right* mouse. Without releasing the button, move the mouse from left to right. The slider moves with the mouse, and the object is transformed (rotated, scaled, or translated) in the direction of the chosen slider (x, y or z).

To stop displaying the sliders, move the cursor out of the region of the sliders. Now press the *right* mouse. The pop-up menu appears. Move the cursor out of the pop-up menu and release the *right* mouse so that no menu option is chosen or choose a new operation from the menu. The sliders will have disappeared.

Outside the Window Manager

You need to leave the window manager to run some programs, such as the gem editor and the flight simulator. To leave the window manager, choose the *exit* selection from the background menu. Note that it gives you a second menu to confirm or deny confirmation of your intention to exit the window manager. Whether or not you exit the window manager

depends upon your response to the second menu displayed. All outstanding windows will be killed.

Then, to find the correct directory with the gem editor, flight simulator and other non-window manager demos, type **standalone**. This will restore the size of the console window and jump you into the correct directory in the file system. Now, if you want to determine what demos are available to you, list the directory by typing **ls**.

Shutting Down the IRIS

This section tells how to terminate (shut down) operations on an IRIS workstation. Please follow these instructions carefully.

If you are giving a demo in Mountain View, all you need to do is log out of your session. Simply type **logout**. You should *not* turn the power off the system, because other people may soon use the machine.

If you are at a customer site, you may need to turn off the system and transport it. Some parts of the IRIS, particularly the disks, are sensitive, so the machine cannot be powered down like a toaster.

To shut down the system, you need to have the authority of the system administrator. If you are not logged in, then log in as *root*, the system administrator's account. If you are already logged in, but not as the system administrator, you can type **su**, which temporarily substitutes the system administrator's privileges. In either case, you may need to know the system administrator's password. Once logged in, type **reboot**. The screen will clear and diagnostics will appear. Then simply shut off the power switch on the front panel.

MOUSE AEROBICS

Demo Software Release 3.4

This manual explains how to operate the mouse with the Multiple Exposure (mex) window manager.

The manual is for Silicon Graphics sales, marketing and executive personnel. It is not for distribution outside the company.

This section tells how to use the mouse to control the window manager. Almost all the window manager functions can be controlled by the mouse X-Y valuator and its rightmost button of the three on the mouse. Optional configuration files allow different button functions. Here the button functions of the sales demonstration window manager configuration are explained.

Hold the mouse with the wire away from you. The three buttons are named LEFTMOUSE, MIDDLEMOUSE, and RIGHTMOUSE. RIGHTMOUSE is the most important button to the window manager.

Menus and Windows

Move the mouse to place the cursor anywhere within the *console* window. Press and hold down the RIGHTMOUSE button. Do not lift your finger off the button. A menu with seven choices will appear. Place the cursor on the menu entry labeled *move*. The entry will change to a brighter color.

The cursor will change shape (to four arrows) and a red outline will appear. Move the mouse to drag the red outline to some position of your choosing. Press and release RIGHTMOUSE again.

The *console* window will disappear. It will be redrawn at the new location you have selected.

This is the basic interaction with the window manager:

1. Move the mouse to select a window.
2. Press RIGHTMOUSE, select a menu entry, and release RIGHTMOUSE. To leave the menu without selecting anything, move the cursor clear of the menu and release RIGHTMOUSE.
3. If the entry selected is *move* or *reshape*, move the mouse to select the location or shape you choose and press and release RIGHTMOUSE again.

Now try again, selecting the *reshape* menu entry. Note that the cursor is shaped like an upper left or lower right corner with an arrow. Press and hold the RIGHTMOUSE to fix one corner of the new window. A cursor shaped like the remaining corner will appear. Select its location with the mouse. Now release RIGHTMOUSE to draw the moved and reshaped window.

This is the basic sequence to open or resize a window:

1. Move the mouse to a position where you like to put one corner of your window.
2. Press and hold the RIGHTMOUSE to *anchor* that corner.
3. While you are still pressing the button, move the mouse to the position where you would like to put the other corner of the window. This *stretches* the size of the window.
4. Now release the RIGHTMOUSE and *anchor* the other corner of the window.

Remember: to open or resize a window, the steps are: *anchor, stretch, anchor*.

Exercise: Make the console viewport about two inches tall and as wide as the system will let you make it. Place the window close to the bottom of the screen.

As you use the window manager, you will notice that some windows fixed or constrained in size, position, or aspect ratio. Some others are unrestricted. The characteristics of each window are determined by its program before the window is opened.

Exercise: The program will limit the size of the console port. What are the limits and why are they imposed?

With only one window on the screen, we can't do much besides *move* and *reshape*. Next, we'll see how to make another window.

Background Menus

If the cursor is outside any window when you press RIGHTMOUSE, a special three-line menu will appear. Choose *new shell* to bring up another instance of the C shell. The cursor will change shape. Place and size the new shell as if you were using the *reshape* menu option; that is, *anchor, stretch* and *anchor*.

Going Back to the Main Menu

We now have two text windows. Since you know how *move* and *reshape* work already, use them on the windows in the display. Move them so that they overlap one another.

Now try *push* and *pop*. The former pushes the selected window *under* any windows that overlap it. The latter pops the selected window *over* any windows that obstruct it.

Exercise: Manipulate a window so it can't be seen at all. Then find it again.

Attaching, Detaching, and Killing Windows

At any time, one window is "attached". This means that mouse motions, keyboard strokes, and button pushes will (usually) be accepted by that window. This is sometimes termed attaching your *input focus* to a window. The attached window is always outlined in red.

Up till now, we have always been attached to the *console* window. Now attach to the new shell in a different window. Move the cursor onto the other shell window, Press RIGHTMOUSE, choose *attach* from the menu, and release the button. The window will be outlined in red.

To verify that it all works, type a simple UNIX command, such as `ls`. Note where the input and output go.

There are two ways to detach input focus from a window. One way is to simply strike `PF4`, the key at the extreme upper right of the keyboard. The red outline will disappear. With our sales demonstration window manager configuration, `PF4` always detaches input focus from a window.

Now type something on the keyboard. Nothing happens! Now move the cursor to the *console* window, choose *attach*, and verify that the outline appears and the keyboard works again. Note that once another program has been attached, each subsequent program must be reattached explicitly.

The other way to detach input focus from a window is even simpler. Just move the cursor outside of the currently attached window and into another window. Pressing the `RIGHTMOUSE` will bring up the window manager menu, and you can attach to another window. Attaching input focus to another window automatically detaches the input focus from the other window.

The window manager menu entry labeled *select* does the operation of both a *pop* and an *attach*.

The last entry in the window manager menu is *kill*. Use it to kill a window on the screen. Note that it gives you a second menu to confirm or deny confirmation of your intention to exit the window manager. Whether or not your process is killed depends upon your response to the second menu displayed.

Q: What happens when you try to kill the *console* window?

A: Nothing. The window manager does not allow the console to be killed. The console is always available for typing UNIX commands.

Back to the Background Menu

If the cursor is not over any window and you press `RIGHTMOUSE`, a three entry menu will appear. We have already covered that the *new shell* entry will create a UNIX text window. The *attach* directs input focus to the background process. A side effect of this *attach* is that input focus is detached from any other process. The effect of using this *attach* is equivalent to striking the `PF4` key.

The third menu entry, *exit*, we will cover on its own in the next section.

Leaving the Window Manager

To leave the window manager, choose the *exit* selection from the background menu. Note that it gives you a second menu to confirm or deny confirmation of your intention to exit the window manager. Whether or not you exit the window manager depends upon your response to the second menu displayed. All outstanding windows will be killed.

As soon as you leave the window manager, control will revert to the console window. You may run any program that does not use the window manager.

The Complete Guide to Surfcar

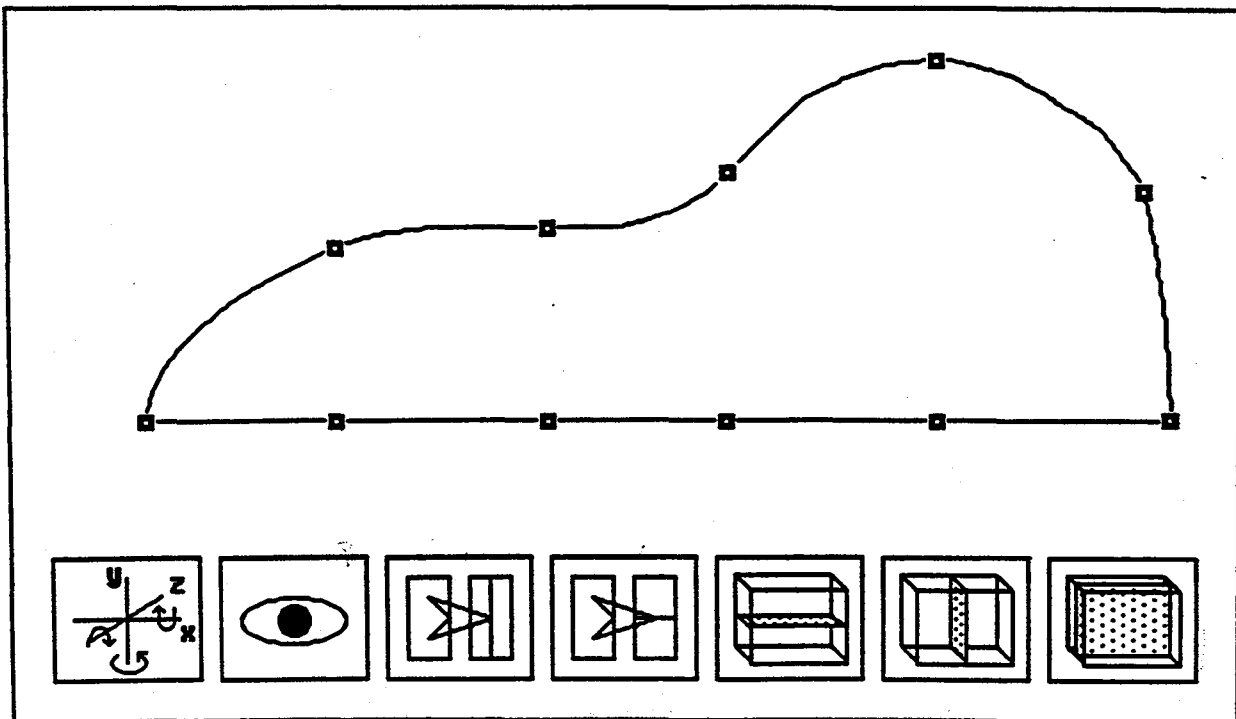
Silicon Graphics Inc.
revised: September, 1986
Release 3.4
Mason Woo

This guide provides a complete reference to the surface editor demo programs. The reference is divided into two sections. The first section explains how to give an abbreviated, yet impressive surface editor demo. The advanced section details everything else that the surface editor can do.

1. The simplest surface editor demo

The surface editor demo is an essential piece of an impressive product demonstration. The demo is a smaller version of a hypothetical CAD package which shows real-time manipulation of curves and patches. The Geometry Engine provides hardware support for curve and patch functions, which is a significant advantage Silicon Graphics has over competitors.

The control points which determine the wireframe model in the surface editor can be written to another program in another window, the shaded surface window. In this new window, the wireframe patch is subdivided into a mesh of very small polygons (triangles). A light source is added, and the brightness at each vertex of the mesh is calculated. What was previously wireframe can now be viewed as a filled object. For a smooth, attractive appearance, Gouraud-shading is used for the polygon fill. To remove hidden surfaces from the solid body, the z-buffering technique is used. On the IRIS, there is hardware support for z-buffering which makes for fast z-buffering.



Of the family of surface editor demos (surfcar, surfjet, surfegg and surfabstract), surfcar is the most frequently used. The car body shape is the most familiar to clients. The rest of this document will refer to the surface editor demo simply as surfcar. The z-buffered, Gouraud-shaded window will be called zshadecar.

1) **Running Surfcar**

To run surfcar, you must be in the window manager. Type surfcar (or any other surface editor program) and hit return. Create a large window for surfcar; it should dominate your screen. Now while the cursor is over the surfcar window, press the right mouse button. A pop-up menu will appear. Select the choice *attach* from the menu. Now the input devices of the keyboard and mouse are focused upon the surfcar program. There are two parts to the surfcar window. The bottom portion is a seven item menu. The top portion is a wireframe object with small squares (control points) and a background. The window should look like the diagram shown on the previous page. (No, the diagram doesn't look a whole lot like a 3-D car, but I am not an artist.)

2) **Above the Menu**

Above the seven item menu is a wireframe car body. The car body is a series of bicubic curves which forms a surface. The small squares are the control points which, in conjunction with the bases, mathematically determine the shape of the surface. The control points can be moved in any of three directions along the x, y or z axes to change the shape of the surface.

To pick and move a point

To move a control point, roll the mouse and position the cursor over a control point. The left mouse button moves the control point in the x direction; the middle button, in the y direction; and the right button in the z direction. Next press one of the three mouse buttons and **hold the button down**. While the button is held down, move the mouse to the left or right. Until the button is released, the control point will move in the direction specified by the mouse button. When the button is released, the control point will be in a new place.

Little known fact--The wireframe of the object and its control points cast shadows on the walls of the background. The shadows of the control points are also pickable and moveable. Try it and see!

When you press the mouse button, if your aim is a little off, your cursor may not be positioned over a control point. If you don't hit any control points, the cursor changes shape, and the shape will reflect which button is pressed. That is, the cursor will look like a control point and the direction of movement specified by the mouse button.



If you forget in which direction the mouse buttons move points, the changing cursor shape can jog your memory. Just move the cursor away from any control point, press the mouse button, and see which direction the arrow points.

Common Problem--moving the mouse up and down

A common error is to move a control point by moving the mouse up and down,

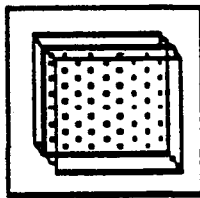
rather than left to right. In any orientation, moving the mouse up and down will not move the control point at all.

3) Getting Around with Two Menu Buttons

You only need to use two buttons from the seven item icon menu to get through the simplest demo. Ignore all the other buttons until the next section of this documentation.

plane of symmetry

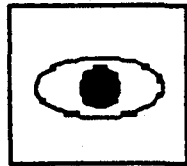
The rightmost menu choice (see diagram below) defines a plane of symmetry which longitudinally divides the wireframe patch in half. Press any mouse button over the rightmost menu choice. When the wireframe is split, one side is not visible. Any changes (movement of control points) in the visible side will also be done in the side which is not visible, and the object will be kept symmetric. This is a nice effect.



To restore visibility to both sides of the wire frame, again press any mouse button over the rightmost menu choice. This menu choice turns on and off (toggles) the symmetry feature.

view button (eye)

By pressing and releasing a mouse button over the eye menu choice (see diagram below), one can change the view of the wireframe body. The different mouse buttons (left, middle and right) enter viewing modes which are not exited until another mouse button is pressed again. At this point, you should only use the **RIGHT** mouse button to exit any viewing mode. Read on while trying this on a machine.



Press and release the **LEFT** mouse button over the eye menu choice. This enters a mode whereby the wireframe model spins around while the background remains fixed. Without pressing any mouse buttons, rolling the mouse around moves the object. To exit this mode, press and release the **RIGHT** mouse button. If you accidentally use another mouse button to exit the viewing mode, read the 'Common Problem' hint below. Note that when you exit the mode, the wireframe returns to the same position it started in.

Press and release the **MIDDLE** mouse button over the eye menu choice. In this mode, the background disappears, and the wireframe is depthcued as it rotates. To exit the mode, press and release the **RIGHT** mouse button. When you exit this mode, the wireframe again returns to the same position it started in.

Finally, press and release the **RIGHT** mouse button over the eye menu choice. In this mode, the background and car rotate together. To exit the mode, again press

and release the **RIGHT** mouse button. This time, however, when you exit the mode, the background and car *will stay* in their new orientations.

Common Problem--cursor looks funny, surfcar not responding

If you exited one of the above viewing modes by clicking the **left**, rather than **right**, mouse button, a new window unexpectedly will be created. You should notice the change in the shape of the cursor, and open the new window on the screen.

Making this new window is not always undesirable. The next section explains how to create this window to enhance your presentation.

spawning a new window with zshadecar

The wireframe from the surfcar program can be viewed as a solid, smoothly shaded figure (zshadecar) in another window. You can spawn zshadecar directly from surfcar.

In surfcar, enter any viewing mode by pressing and releasing any of the three mouse buttons over the eye menu choice. You had previously restricted yourself to exiting by pressing and releasing the right mouse button. This time, use the **left** mouse button to exit.

The cursor will change shape, requesting that a window be opened. Then you should anchor this newly created window onto the screen. You will still be attached and can go back to the surfcar program, but now there will be two windows on the screen. Most likely, you want to be attached to the new zshadecar window. Strike the PF4 key to detach from surfcar. Next, attach to zshadecar, via the window manager pop-up menu.

In this new window, the wireframe patch is subdivided into a mesh of very small polygons (triangles). A light source is added, and the brightness at each vertex of the mesh is calculated. What was previously wireframe can now be viewed as a filled object. For a smooth, attractive appearance, Gouraud-shading is used for the polygon fill. To remove hidden surfaces from the solid body, the z-buffering technique is used. On the IRIS, there is hardware support for z-buffering which makes for fast z-buffering. (I've already said that before, but it is an appropriate time to say it again.)

For more specifics on how to run the zshadecar program, see the man page for zshadecar.

Common Problem--can't use the newly created window

After spawning the zshadecar, it is very common to try to attach to the zshadecar before detaching from surfcar. Typically, you will repeatedly press the right mouse button without seeing a pop-up menu. Look closely, usually pressing the right mouse button will cause the cursor to change shape to:



The simple solution is to strike the PF4 key.

Common Problem--can't z-buffer zshadecar

Once attached to the zshadecar program, the right mouse button pops up a

menu of object manipulations. If you select the z-buffer choice and nothing happens, two things could be happening. Your machine may have fewer than 32 bitplanes, in which case, z-buffering is not possible due to lack of bitplanes. Otherwise, there may be a double buffered program outstanding. In that case, detach from the zshadecar program (strike PF4). Seek out and destroy all double buffered programs, including the original surface editor. Now reattach your input devices to the zshadecar program and try z-buffering again.

reviewing the modes

entering the viewing mode

mouse button viewing mode

left	car moves, background stationary
middle	car depthcued, background disappears
right	car and background move

exiting the viewing mode

mouse button viewing mode

left	exit and spawn zshadecar window
right	exit quietly

Common Complaint--why does it flash so much

Whenever you detach from the surfcar program, the program goes from double buffer to single buffer mode. This is to accommodate the zshadecar program. To z-buffer the smoothly shaded object, there can not be any existing double buffer programs. Thus, the solution is to make a *sleeping* surfcar program go into single buffer mode. Going from single to double buffer or vice-versa causes the flash.

4) **The Complete Basic Surfcar**

So here is the concise overview of a good surfcar demo. Bring up the surfcar window. Make it nice and big. Attach to it. Go into your favorite viewing mode and spin the wireframe around. *Do not* spawn the zshadecar window yet. Turn on the plane of symmetry. Modify a few of the control points on the car, being sure to emphasize that the points can be moved in any of three dimensions. Put tail fins on the car and make wheel wells. Go wild.

Now spawn off the zshadecar window with the shape you have created. Show the client a fully Gouraud-shaded object with hidden surfaces removed by IRIS z-buffering. This should take about five minutes of your demonstration.

You know enough to give an impressive demo with the surface editor programs. You can stop right here, practice and consider yourself quite proficient at this demo. Anything learned after this point is largely a bonus.

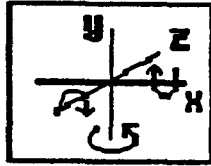
2. Mastering surfcar

This section discusses everything else there is to know about surfcar.

1) **Rotating the Points**

The entire set of control points can be rotated independent of the background. The

points can be rotated 90 degrees at a time around a particular axis. The leftmost menu item (showing the three axes and the curly arrows representing rotation) is essential for this operation.



To rotate the control points, move the cursor over one of the curly arrows in this menu button. Pressing and releasing the **LEFT** mouse button rotates the control points (and wireframe car body) around the axis shown in the diagram. Try this over one of the arrows. Wait until the car body has settled into a new position before trying to do anything else.

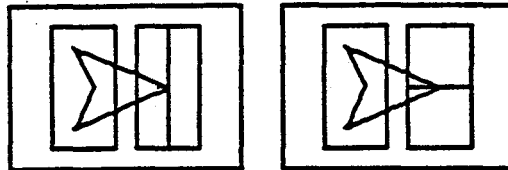
Pressing and releasing the **RIGHT** mouse button over a curly arrow rotates the points and body around the axis shown, but in the **OPPOSITE** direction from the **LEFT** mouse button. The middle mouse button rotates the object in the same direction as the left mouse button.

Common Problem--not hitting a curly arrow

If the cursor is not over an arrow when the mouse button is pressed, the menu item will signify the error by flashing red. To stop the menu item from flashing, you can move the cursor over one of the control points and, as if you were editing the object, press any mouse button.

2) **Double Your Pleasure--Double Your Fun**

The third and fourth menu choices (shown below) double the number of control points. Depending upon which menu choice is selected, the doubling takes place in one of the two 'directions' along the patch. ('Directions' is an imprecise term. More accurately the control points are doubled along the *s* and *t* variables in the parametric equations for the bicubic patch. It's okay to say 'directions'; no one will be that picky.)



Doubling the number of control points slows the performance of the demo. Doubling the points once is okay, but more than once noticeably reduces the speed and thus impact of the demo. If you do double the number of points, make sure you attract your client's attention and that he or she is sufficiently awed.

Warning--no way to undo doubling

Once you have doubled the number of points, there is no way to go back to the original number of points.

3) **More Planes of Symmetry**

The three rightmost menu choices (see figure below) turn on and off planes of symmetry. At any given time, only two planes of symmetry are possible. The two possible planes divide the two 'directions' along the patch. At any time, the third plane of symmetry will not be allowed, and choosing the third plane will cause the menu item to flash.

In the default position, the two rightmost (sixth and seventh) menu items are allowable planes of symmetry. The fifth menu item does not divide the object across one of its directions, so it is ineligible. As the control points and object are rotated (See previous section entitled 'Rotating the Points. '), the disallowed plane of symmetry changes.

