

UNISYS

Basic UNIX[®] (System V, Release 4) Usage Workshop

Student Guide

Copyright[©] 1992 Unisys Corporation.

Unisys is a registered trademark of Unisys Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

Unisys Release 1.0.1

February 1992

AL 3822

Printed in U S America
UE 7415

The names, places, and/or events used in this publication are purely fictitious and are not intended to correspond to any real individual, group, company, or event. Any similarity or likeness to any real individual, company, or event is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed license or agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such license or agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded to Unisys Corporation, Education Publications, P.O. Box 1110, Princeton, NJ 08543 U.S.A.

Contents

Course Description	v
Agenda	vii
About This Course	ix
Module 1. Operating System Overview	
Module 2. Basics	
Module 3. File Systems	
Module 4. Text Editors	
Module 5. Directory and File Management	
Module 6. Directory and File Access Management	
Module 7. Basic Communications	
Appendix A. Terminal Setup	
Appendix B. Standard Directories and Files	
Appendix C. Command Summary	
Appendix D. Basic Networking Utilities	

Course Description

Audience

Application users, programmers, and system support personnel

Prerequisites

Knowledge of data processing is desirable

Objective

Upon successful completion of this course, the student should be able to perform basic user functions using the UNIX operating system.

Description

This entry-level course introduces fundamental principles of the UNIX operating system and teaches the skills needed to use UNIX to perform basic user functions. It also satisfies prerequisites for advanced UNIX courses.

This instructor-led course includes section summaries and practical exercises to supplement structured discussions. Hands-on activities are provided throughout this course.

Topics

- UNIX operating system
- File system organization
- Text editors
- File and directory management
- File and directory access control
- Communication activities

Duration

3 days

Agenda

Day 1

- **Module 1. Operating System Overview**
 - UNIX Development
 - UNIX Features
 - UNIX 4.0 features
 - Components

- **Module 2. Basics**
 - Terminal orientation
 - Start a UNIX session
 - Command format
 - Command execution cycle
 - Common command line errors
 - Control the UNIX session
 - Set and change login passwords
 - Use online reference commands
 - End a UNIX session

- **Module 3. File Systems**
 - File types
 - File organization
 - Path name file references
 - File and directory names
 - Special characters in file names

Day 2

- **Module 4. Text Editors**
 - Overview of UNIX text editors
 - Line editor ed
 - Create file
 - Command format
 - Address lines
 - Display text
 - Enter text
 - Modify text
 - Escape to UNIX shell
 - Recovery procedure

Agenda

- **Module 4. Text Editors (continued)**
 - Screen editor vi
 - Identify terminal type
 - Create file
 - Position cursor within file
 - Enter text
 - Modify text
 - Use line editor commands
 - Change vi environment
 - Recovery procedure

- **Module 5. Directory and File Management**
 - Directory management
 - Display current directory
 - List directory contents
 - Create directories
 - Change current directory
 - Remove directories

 - File management
 - Classify files
 - Display file contents
 - Copy files
 - Link files
 - Move and rename files
 - Print files
 - Locate files
 - Search files for text patterns
 - Sort file contents
 - Remove files

Day 3

- **Module 6. Directory and File Access Management**
 - Security overview
 - User types
 - Access permission types
 - Display file and directory permissions
 - Change access permissions
 - Change default permissions
 - Change file ownership
 - Change file and user group association

- **Module 7. Basic Communications**
 - Communication concepts
 - Local communication utilities

About This Course

Student Guide Organization

This course is directed towards individuals with diverse data processing experience. It presents a broad exposure to fundamental UNIX principles and provides practical guidance to perform basic UNIX user functions. References to product documentation are provided to direct the student.

This student guide consists of seven modules, or sections, and four appendices. Each module begins with a set of learning objectives, providing structure and purpose to the body of information contained within the module. A module summary and practical and written exercises are provided in each module. Longer modules contain multiple, staggered practical exercises providing hands-on activities on selected topics in addition to the module summary and the final module exercise.

The format for module pages in this student guide is shown below. The left page contains descriptive text pertaining to the information appearing on the right page. Product information documentation is listed at the bottom of the left page. The right page generally contains text, tables, or graphics.

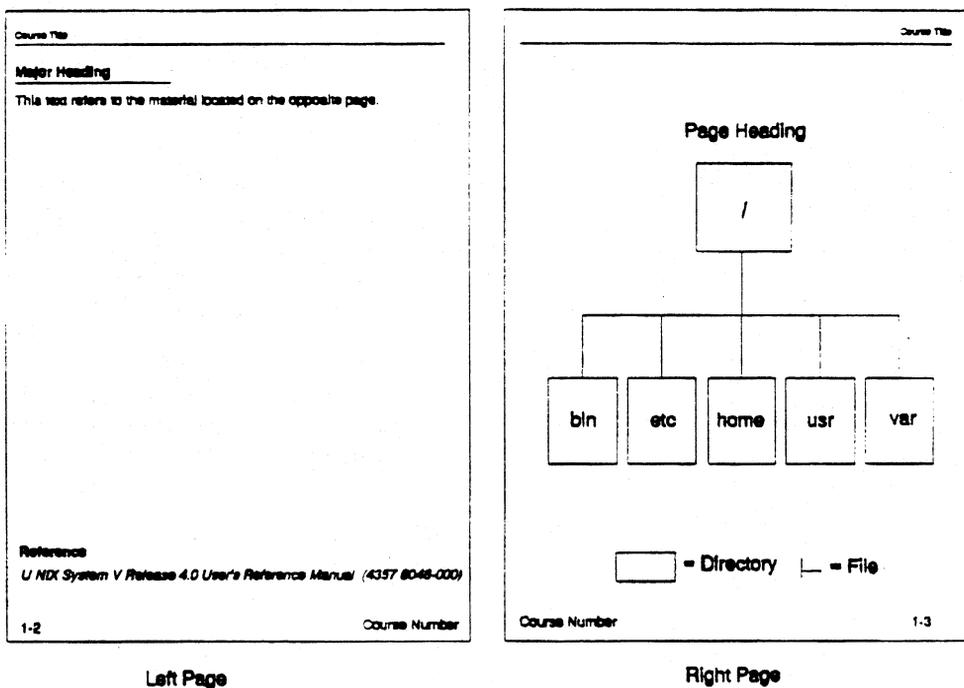


Figure 1. Page Layout

Typographic Conventions

The following conventions are used throughout this document.

Convention	Represents
Bold	Command name Command line entry
<i>Italics</i>	File name Directory name Variable information Documentation title
Courier	Terminal entry Terminal display
< >	Input that does not appear on the screen when typed, such as TAB, ESCAPE, and RETURN keys
<^char>	Control characters that do not appear on the screen when typed. The circumflex (^) represents the control key, usually labeled Ctrl. To type a control character, hold down the control key while pressing the specified character. For example, to enter <^d>, hold down the control key while pressing the letter d key – the letter d does not appear on the screen.
[]	Command options and arguments considered optional are enclosed in brackets. Brackets should not be entered as part of the command line.

Unisys Reference Documentation

Abundant information on UNIX is available in Unisys publications and in commercial textbooks. It is important to be familiar with the available resources in order to locate desired information quickly and efficiently.

This section presents an orientation to Unisys reference documentation. Descriptions of key standard reference documentation are provided below.

- *Release Notes* Describe the capabilities and features of the 4.0 UNIX operating system
- *Software Installation and Operations Guide* Contains operating system and software product installation procedures, and basic startup and maintenance procedures
- *Administrator's Guide* Describes system administration and maintenance functions such as system setup and configuration, user and device management, file system administration, backup and restore procedures, print service administration, and system security functions
- *Administrator's Reference Manual(s)* Contains descriptions of administrative commands, file formats, and special files
- *Error Message Manual* Provides assistance in interpreting error messages and determining probable causes and solutions for hardware and software problems; contains an index of alphabetic error messages to facilitate quick access
- *Programmers's Guide* Presents an overview of the UNIX system programming environment and tutorials on various programming tools
- *Programmer's Reference Manual(s)* Describes programming commands, system calls, library routines, formats, and miscellaneous utilities
- *User's Guide* Presents an overview of the UNIX operating system and tutorials on user-related topics like text editing, print services, electronic mail, and network communication
- *User's Reference Manual* Describes user commands
- *Network User's and Administrator's Guide* Directed to users of remote services and to the system administrator setting up and maintaining file sharing capability

Of the documents listed above, the organization of the reference manuals containing command descriptions requires further examination.

Unisys Reference Documentation

Document Title	Number
<i>Release Notes</i>	4357 7592-000
<i>UNIX System V Release 4 Installation and Operation Guide</i>	3915 2483-000
<i>UNIX System V Release 4 Administrator's Guide</i>	3915 2442-000
<i>UNIX System V Release 4 Network User's and Administrator's Guide</i>	3915 2467-000
<i>UNIX System V Release 4 Error Message Manual</i>	3915 2624-000
<i>UNIX System V Release 4 X Window Access Operation Guide</i>	7431 1473-000
<i>Commercial Secure: Security Features User's Guide</i>	7431 2810-000
<i>Commercial Secure: Trusted Facility Tuning Manual</i>	7431 2802-000
<i>UNIX System V Release 4 Tuning Guide</i>	3915 2525-000
<i>UNIX System V Release 4 Administrator's Reference Manual</i>	V1 - 4357 7451-000 V2 - 4357 7469-000
<i>UNIX System V Release 4 User's Reference Manual</i>	4357 7444-000
<i>UNIX System V Release 4 User's Reference Manual Supplement</i>	3915 2962-000
<i>UNIX System V Release 4 User's Guide</i>	3914 9398-000
<i>UNIX System V Release 4 Programmer's Guide Supplement</i>	3915 2921-000
<i>UNIX System V Release 4 Programmer's Guide: Support Services and Application Tools</i>	3914 9463-000
<i>UNIX System V Release 4 Programmer's Guide: ANSI C and Programming Support Tools</i>	3914 9414-000
<i>UNIX System V Release 4 Programmer's Guide: Character User Interface: FMLI, ETI</i>	3914 9406-000
<i>UNIX System V Release 4 Programmer's Guide: STREAMS</i>	3915 2608-000
<i>UNIX System V Release 4 International Enhancements Guide</i>	3915 2566-000
<i>UNIX System V Release 4 Programmer's Guide: Networking Interfaces</i>	3914 9422-100
<i>UNIX System V Release 4 Device Driver Programmer's Guide</i>	3915 2988-000
<i>UNIX System V Release 4 BSD/XENIX Compatibility Guide</i>	3915 2574-000
<i>UNIX System V Release 4 ANSI C Transition Guide</i>	3915 2590-00

Unisys Reference Manual Organization

Traditionally, UNIX reference manuals containing command descriptions adhere to the organization used by AT&T, the developer of UNIX. The AT&T organizational model consists of eight standard sections described below.

- | | |
|------------------|---|
| <i>Section 1</i> | Contains alphabetic descriptions of commands and utilities for administrators, users, and programmers |
| <i>Section 2</i> | Describes the system calls used to interact with the kernel |
| <i>Section 3</i> | Describes the library of subroutines available for programmers |
| <i>Section 4</i> | Describes main system files |
| <i>Section 5</i> | Contains various information and miscellaneous facilities, like character set tables and macro packages |
| <i>Section 6</i> | Games, not included in Unisys documentation |
| <i>Section 7</i> | Describes system special files, like device files |
| <i>Section 8</i> | System maintenance procedures, not included in Unisys documentation |

Unisys has grouped the *Section 1* commands for users, administrators, and programmers into separate volumes containing commands and file format references extracted from the standard AT&T documents and Unisys customized commands.

Each set of reference manuals contains a table of contents listing the command entries in alphabetic order by section, where multiple sections exist in the document, and a permuted index to locate a command by topic, followed by command entry descriptions.

Each command name appears in the format `command (AT&T_section_number)`, for example the `date(1)` command. The section number may also include a letter designating a grouping of like commands. For example, **C** refers to communication, **M** refers to maintenance, and **G** refers to graphics.

Unisys Reference Manual Organization

Unisys Manual	Extracted AT&T Section	Example
<i>User's Reference Manual</i>	1 User commands commands and utilities	mandex(1) passwd(1) uucp(C)
<i>Administrator's Reference Manual</i>	1 Administrator commands and utilities	boot(1M) fsck(1M) mkfs(1M)
	4 File formats	inittab(4) passwd(4) profile(4)
	5 Miscellaneous facilities	regex(5) signal(5) term(5)
	7 Special files	filesystem(7) termio(7)
<i>Programmer's Reference Manual(s)</i>	1 Programmer commands and utilities	cc(1) lint(1) sdb(1)
	2 System calls	sysfs(2) symlink(2)
	3 Subroutines	sleep(3C) basename(3G) trig(3M)
	4 File formats	a.out(4) core(4)
	5 Miscellaneous facilities	ascii(5) math(5)

Reference Manual Entries

Each manual entry is formatted to include an appropriate subset of the following headings.

<i>NAME</i>	Entry name and function; may include related (secondary) entries
<i>SYNOPSIS</i>	Format of command, system call or library routine
<i>DESCRIPTION</i>	Overview of command usage or topic
<i>EXAMPLES</i>	Examples of command syntax or usage, where appropriate
<i>FILES</i>	File names referenced by the command
<i>EXIT STATUS</i>	Value(s) set when command terminates
<i>RETURN VALUES</i>	Value(s) returned during command execution
<i>NOTES</i>	Helpful information or special considerations
<i>SEE ALSO</i>	Pointers to related information
<i>DIAGNOSTICS</i>	Diagnostic message interpretations
<i>WARNINGS</i>	Potential misuse, restrictions, limitations, or boundaries
<i>BUGS or RESTRICTIONS</i>	Known faults, deficiencies, or limitations.

Two sample manual pages are illustrated on the next page. The corresponding software utility package, Essential Utilities in this case, appears across from the command entry. Notice that the headings are not identical for each command description.

Reference Manual Entries

pwd

pwd(1)	Essential Utilities
NAME	pwd - working directory name
SYNOPSIS	pwd
DESCRIPTION	pwd prints the pathname of the working (current) directory.
SEE ALSO	cd(1).
DIAGNOSTICS	Cannot open .. and Read error in .. indicate possible file system trouble and should be referred to a UNIX system administrator.
NOTES	If you move the current directory or one above it, pwd may not give the correct response. Use the cd(1) command with a full pathname to correct this situation.

cd

cd(1)	Essential Utilities
NAME	cd - change working directory
SYNOPSIS	cd [directory]
DESCRIPTION	<p>The cd command changes to a new working directory. If directory is not specified, the value of shell parameter \$HOME is used as the new working directory. If directory specifies a complete path starting with /, .. or .., directory becomes the new working directory. If neither case applies, cd tries to find the designated directory relative to one of the paths specified by the \$CDPATH shell variable. \$CDPATH has the same syntax as, and similar semantics to, the \$PATH shell variable. cd must have execute (search) permission in directory.</p> <p>Because a new process is created to execute each command, cd would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the shell.</p>
SEE ALSO	pwd(1), sh(1). chdir(2) in the Programmer's Reference Manual.

Online Reference Commands

The following UNIX commands provide online access to reference manual information.

man

This command accesses reference manual entries on the system. The information displayed is the same as the printed reference manuals described previously.

mandex

This command provides access to a menu-driven indexing system to search selected online manuals for a subject or command.

Online Reference Commands

- **man**

Online reference manual

- **mandex**

Menu-driven indexing system to online manuals

1

Operating System Overview

Objectives

Upon completion of this module, you should be able to

1. Describe the development of the UNIX operating system.
2. List and describe features of the UNIX operating system.
3. Describe the components of the UNIX operating system.

Reference

Documentation referenced in this module

- *UNIX System V Release 4 User's Reference Manual (4357 7444-000)*

UNIX Development

UNIX is the name of an operating system and its family of related utility programs. UNIX provides an interactive, multiuser, and multitasking computing environment. As a result of its growing popularity, UNIX has emerged as a standard operating system.

The development and evolution of the UNIX operating system can be divided into two distinct periods — before and after its release as a commercial product.

Pre-Commercial Era

UNIX was developed in 1969 at the research division of AT&T, Bell Laboratories (now UNIX Systems Laboratories, Inc.), by a programmer named Ken Thompson for a research project on a DEC computer. Originally, it was written in machine-dependent assembly language for a single-user environment. This restricted its use to a specific computer and to one user at a time.

In the early 1970's, the UNIX operating system was rewritten in a higher-level, machine-independent language called B. It provided a multiuser operating environment allowing multiple users to access and to use computer resources. The B language was modified and renamed C in 1973, allowing any computer that has a C language compiler (to translate C instructions into the computer's internal language) to run the UNIX operating system. Most of the UNIX operating system is written in C, with the exception of the machine-dependent device driver code.

During the mid 1970's, universities and government agencies were licensed to run UNIX. As a result, many computer science majors have learned UNIX. In addition, universities, like the University of California at Berkeley, have made significant additions and changes to UNIX. Berkeley's enhancements became so popular that one version of UNIX is called the Berkeley Software Distribution (BSD), initially released in 1980.

Post-Commercial Era

UNIX was licensed for commercial use in 1981 as a result of students trained in UNIX moving into industry and seeking a similar computing environment. This marked the second significant period in UNIX development.

Since its initial implementation and subsequent commercial release, UNIX has continued to evolve with enhancements and refinements incorporated into successive versions. A variety of look-alike operating systems are also available today. UNIX 4.0 (also referred to as System V.4 or SVR4) is the current release. Features of this operating system release are described later in this module.

UNIX Development

- **Pre-Commercial Era**

- | | |
|--------------|--|
| 1969 | UNIX developed by Ken Thompson at Bell Laboratories (AT&T) |
| Early 1970's | UNIX rewritten in machine-independent language – B |
| 1973 | UNIX rewritten in modified language – C |
| Mid 1970's | Universities and Government agencies licensed to run UNIX |
| 1980 | Berkeley released UNIX version – BSD |

- **Post-Commercial Era**

- | | |
|------|--|
| 1981 | UNIX licensed for commercial use |
| 1983 | System V.1 represents a move toward standardization |
| 1984 | System V.2 adds more improvements – system performance and documentation |
| 1986 | System V.3 released |
| 1991 | System V.4 released |

UNIX Features

When the UNIX operating system was developed, many computers were still single-user, batch systems permitting only one user at a time to input data for processing and preventing other interaction with the system while it was processing the data.

Multuser

UNIX is a *multiuser* operating system, allowing more than one user to access system resources by communicating (interacting) directly with the computer via terminals. This capability increased system resource utilization and reduced the overall cost significantly.

Multitasking

The *multitasking* feature allows any one user to perform several tasks at a time. Several tasks, or jobs, can be run in the background while another job is displayed on the terminal. Some operating environments (like the Korn shell) provide job control capability to control program execution, that is, to switch back and forth between jobs or to suspend and resume jobs. Overall, the multitasking capability of UNIX increases user productivity and efficiency.

Portable

A major advantage of UNIX is its portability, the ability to move it from one type of computer to another. This is possible because most of the operating system is written in a form that is not bound to specific hardware. This feature has become increasingly attractive to hardware vendors having difficulty supporting proprietary systems. UNIX satisfies the need for a generic operating system that can be adopted to other machines. Many applications are now available. More recently, software development allows UNIX and MSDOS® operating systems to coexist on the same system (Merge software product). Initially, UNIX was used on a DEC minicomputer. It can now be used on microcomputers to mainframes, requiring only one operating system to be learned. Software manufacturers facing the additional cost of converting their products to run under different operating systems also find strong advantage in developing applications to run under UNIX.

Powerful Development Tools

UNIX includes a large collection of several hundred utility programs to perform user tasks (such as *sort*) that can be used like building blocks and combined to create customized programs or new commands. The variety, power, and flexibility of UNIX programming support tools offer attractive advantages to programmers in designing applications. UNIX was designed by and for programmers.

MS-DOS is a registered trademark of Microsoft Corporation.

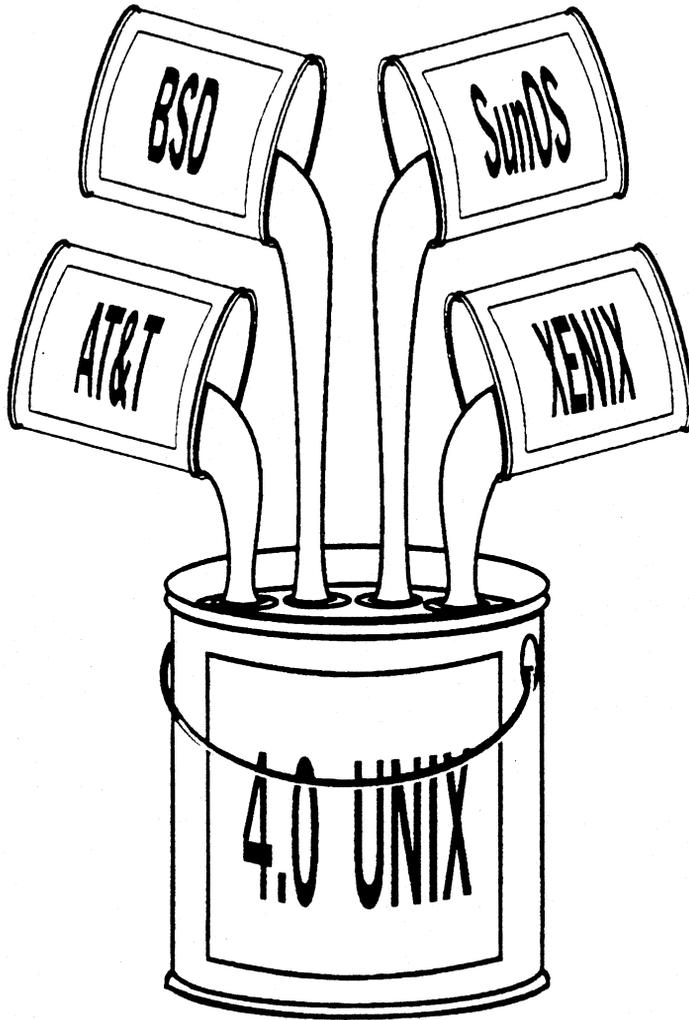
UNIX Features

- **Multuser**
- **Multitasking**
- **Portable**
- **Powerful and flexible development tools**

UNIX 4.0

UNIX 4.0 is the Unisys implementation of AT&T's System V Release 4 (SVR4) version of the UNIX operating system. The goal of UNIX 4.0 is to unify the important variations of UNIX into one robust product that conforms to industry standards and is compatible with the four major variants of UNIX: AT&T, Berkeley (BSD), SunOS, and XENIX operating systems.

UNIX 4.0



UNIX 4.0 Features

This section describes user-related features of UNIX 4.0 operating system.

Commands

Several new, modified, or enhanced commands have been incorporated from BSD, SunOs, and XENIX. Commands not included from the original command sets can still be accessed through compatibility packages for BSD and SunOS.

Additional Interfaces

- Shells

The shell is the user interface to the operating system. Releases prior to UNIX 4.0 provided two shells for users which are still accessible, the Bourne and C shells. Two additional shells are now available, the Korn and job shells.

- User Interface

FACE (Framed Access Command Environment) is a menu-driven interface to perform basic user tasks (like managing files or printer operations) as well as administrative functions.

Virtual File System

A file system is an organized collection of files and directories. One file system type existed prior to UNIX 4.0. The *virtual file system* architecture allows multiple and different file system types to coexist on a system. This concept is derived from BSD.

Quotas

User disk usage can be limited using a quota mechanism. This feature is configured and maintained by the system administrator. Error messages are provided to the user when usage limits have been reached.

Security

Maintaining overall system security is primarily the responsibility of the system administrator. UNIX 4.0 provides improved security measures through audit trailing and changes in passwords and permissions.

UNIX 4.0 Features

- **New, modified, enhanced commands**
- **Additional interfaces**
- **Multiple file system types**
- **User disk usage quotas**
- **Improved security**

UNIX Components

The UNIX system consists of three major components: the kernel, the shell, and commands. Each component is described below.

Kernel

The *kernel* is a program that is loaded into memory when the system is booted. It is considered the manager, or nucleus, of the operating system because it manages memory, allocates processor time, and controls input and output. The kernel functions are transparent to the user.

Shell

A *shell* program is started for each user at log in. It functions as an interface between the user and the kernel by interpreting and initiating command execution. Several shells (Bourne, C, Korn, or job) are available. A shell is assigned to a user by the system administrator when the user account is created. The administrator may change a user's shell at any time. The Bourne shell remains the standard shell in UNIX 4.0 and for this course.

The shell is also a programming language to create new commands or design applications.

Commands

A *command* is a program that performs a task. UNIX offers several hundred commands to use, which may be combined to create customized programs (new commands). Application programs, like MAPPER® or ORACLE®, can also be accessed through the shell.

MAPPER is a registered trademark of Unisys Corporation.
ORACLE is a registered trademark of Oracle Corporation.

UNIX Components

- **Kernel** *Colonel.*
- **Shell**
- **Commands**

Summary

- UNIX is a multiuser, multitasking, portable operating system.
- UNIX was developed in a research environment (1969) by programmers at Bell Laboratories (AT&T). Improvements developed in an academic environment (Berkeley) have also been incorporated.
- UNIX is written almost entirely in C, a machine-independent language.
- UNIX 4.0, the Unisys implementation of AT&T's SVR4 version of UNIX, incorporates features from AT&T, Berkeley, SunOS, and XENIX operating systems.
- User features of UNIX 4.0 include
 - New, modified, or enhanced commands
 - Additional shells
 - Support for multiple file system types
 - User menu system for user functions
 - Enhanced security
 - Quota system to limit user disk usage
- The three components of the UNIX system are
 - Kernel The program that manages the operating system
 - Shell One of several command interpreter programs that interface between the user and the kernel
 - Commands Individual programs that perform a particular task

Exercise

1. True or false. UNIX was originally written in a machine-dependent assembly language for a single-user environment.
 - a. True
 - b. False

2. Briefly define the following terms.
 - a. Multiuser *More than one user on at once*
 - b. Multitasking *Each user can run more than one program*
 - c. Portable *can be run on different platforms*

3. UNIX is written almost entirely in the following language. Indicate your choice.
 - a. assembly
 - b. B
 - c. C

4. Unisys UNIX 4.0 incorporates features exclusively from Berkeley. Indicate your choice.
 - a. True
 - b. False

5. Enter the letter designation of the description from column B that matches the item in column A.

Column A

__ kernel

__ shell

__ commands

Column B

a. A machine-dependent programming language

b. Individual programs that perform a given task

c. A program that manages the operating system

d. A command interpreter program

2

Basics

Module Objectives

Upon completion of this module, you should be able to access and use UNIX.

The supporting module objectives include the ability to

1. Operate a terminal.
2. Log in.
3. Execute a command line.
4. Set and change login passwords.
5. Access online reference manual information.
6. Log out.

Reference

Documentation referenced in this module

- *UNIX System V, Release 4 System User's Reference Manual* (4357 7444-000)
- *Basic UNIX Usage Student Guide, Appendix A* (UE 7415)
- *Advanced UNIX (System V, Release 4) Usage Workshop* (UE 7417)

Accessing UNIX

This module leads you through a typical UNIX session. First, a basic orientation is presented to the terminal that is used to communicate with UNIX. Frequently used control keys are identified. The procedures to log in to UNIX, to execute basic commands, and to log out of UNIX are described. Interpretations of common command usage error messages are provided. Ways to correct typing errors and to stop command execution are described.

Terminal Orientation

A variety of terminals can be used with UNIX. Before a terminal can be used, it must be connected to the system using cables and configured in UNIX by the system administrator in order to be recognized by the UNIX system.

Regardless of the differences among terminals, the following similarities are identified.

- **Power switch** – Ensure the power is turned on. There may be a power LED light on the keyboard. This light should remain illuminated while the terminal is used.
- **Cable connections** – All cable connections to the system and to power outlets must be secure. Cable connections are usually located at the rear of the terminal. If the power switch is turned on but the terminal does not display a login screen, ensure that all cables for the terminal are connected securely. If the terminal remains inactive (no login screen), consult your system administrator.
- **Brightness control** – Most terminals allow screen brightness (intensity) adjustment to suit the user's comfort. This control is usually located at the side or front of the monitor.
- **Keyboard** – Regardless of the keyboard type, all keyboards contain a set of typewriter keys and additional keys used to perform special functions. Frequently used function keys are described Appendix A.
- **Terminal Setup** – Control parameters that determine terminal communication and general operation for Unisys-supported terminals are described in Appendix A.

Reference:

- *Basic UNIX Usage Student Guide, Appendix A, Terminal Setup*

Terminal Orientation

- **Power switch**
- **Cable connections**
- **Brightness control**
- **Keyboard**
- **Terminal setup**

Starting a UNIX Session

This section describes the UNIX login procedure.

Logname

Before you can use UNIX, you must identify yourself to UNIX by logging in with a logname, usually provided by the system administrator. UNIX has a special file, */etc/passwd*, containing information about all system users. Only users with entries in this file may gain access to UNIX. This is a UNIX security feature.

At the login prompt, enter the logname in lowercase and press <RETURN>. UNIX is case sensitive; using uppercase at the login prompt indicates that your terminal is capable of displaying only uppercase characters. Do not include spaces as you enter the logname.

Password

If you are logging in for the first time, you will be prompted to set a password at this time. Use the following rules to select a password:

- A password must be at least 6-8 characters in length. This length is actually defined in the */etc/default/passwd* file.
- Each password must contain at least two alphabetic characters and one number or special character (symbol).
- Each password must be different from the logname. Reverse or circular shift of the logname is generally not permitted.

Enter the password at the prompt and press the <RETURN> key. Notice the password entry is not displayed for security. A prompt is displayed to reenter the password. The two password entries must match or a prompt is displayed to try entering a password again. If the password entered does not satisfy the criteria described above, appropriate messages will be displayed as guidance.

You may notice a slight delay at subsequent logins as the system validates your logname in the */etc/passwd* file and your encrypted password in the */etc/shadow* file.

The `passwd` command used to change the login password is described later in this module.

Shell Command Prompt

After a successful login, several messages are displayed concerning the last login date/time, copyright information, disk usage, new mail, and current news (bulletin) items. This information is displayed by the */etc/profile* file which sets the initial user computing environment. Finally, the shell prompt, usually the \$ character, is displayed indicating that the shell command interpreter is awaiting user input in the form of a command.

The computing environment can be customized for individual users (for example, changing the shell command prompt) by designating the changes in the user's *.profile* file. This topic is described in the Advanced UNIX Usage Workshop (AL 3823) course.

Reference

- *Advanced UNIX (System V, Release 4) Usage Workshop, Module 5*

Starting a UNIX Session

1. **Enter logname**
2. **Set password**

Login: **logname**

You don't have a password. Choose one

passwd **logname**

New password:

Re-enter new password:

Command Execution Cycle

The shell analyzes the command line by separating its various components using the blank as the separator; this is known as *parsing*.

- The shell examines the command line for the presence of any special characters to interpret.
- Assuming that the characters up to the first blank refer to a command, the shell searches for a *program* file by the same name.
- If it locates the program, the shell checks if the requesting user has access permission to use the command.
- The shell continues to examine the remaining command line for format consistency.
- Finally, it requests the kernel to execute the program, passing all valid options and arguments to the command program.
- While the kernel copies the command program file from disk into memory and executes it, the shell remains inactive (sleeps) until the program has completed. The executing program in memory is called a *process*.
- When the command process completes execution, control returns to the shell which displays another prompt to signal that it is ready for the next command.

Command Execution Cycle

- **Shell**

- Interprets any special characters
- Analyzes command line components
- Searches for command program file and checks access permission to command
- Requests kernel to execute command program
- Passes valid command options and arguments to command program
- Sleeps (inactive) during command execution
- Displays shell command prompt after program completes execution

- **Kernel**

- Copies command program from disk into memory
- Executes program as a process
- Returns control to shell after command terminates

UNIX Commands

UNIX offers a diverse palette of several hundred commands to use. Commands are entered in a general format at the shell prompt. Pressing the <RETURN> key signals the end of a command. The shell examines the command entry and initiates appropriate action. This action may be to execute the command program or to display a message that an error has occurred.

Command Line Components

A command entry may include information other than the command name.

- **Options** A command option modifies the way the command works. It is usually preceded by a dash (-). Often multiple options can be combined and preceded by a single dash.
- **Arguments** A command argument is typically a file name or some information supplied to the command. Some commands require multiple arguments, others require none at all.

A command followed by any options or arguments is called a *command line*. The components of a command line appear in a particular order.

\$ command [-options] [arguments]<RETURN>

The UNIX command line has the following characteristics:

- A space (blank) or tab separates each command line component. Although multiple spaces or tabs are permitted, they are ignored by the shell during command line processing.
- The <RETURN> key terminates the command line.
- Command options are immediately preceded by a dash (or plus) character.

Note: Optional information is enclosed in square brackets throughout this manual for instruction purposes only. *Do not* include square brackets in command line entries.

Examples

The examples at the right illustrate sample UNIX command lines with corresponding descriptions.

UNIX Commands

command [*-options*] [*arguments*]

Example 1

\$ who	Command only
\$ who -H	Command and option
\$ who am i	Command and arguments

Example 2

\$ ls	Command only
\$ ls -a	Command and option
\$ ls -al	Command and multiple options
\$ ls -al /etc	Multiple options and argument

Example 3

\$ date	Command only
\$ date +%d	Command option preceded by + symbol (<i>%d</i> refers to a display format code)

Command Line Errors

The next page illustrates common errors encountered during command line execution and the corresponding error messages displayed by the shell. These error messages may appear obscure at first. UNIX may seem terse at times, but recall that UNIX was developed in a research environment by programmers for programmers. With experience, these and similar messages will become familiar and easier to interpret.

Common error messages are listed and described below.

- *Command not found* Command name is misspelled or entered uppercase; command is not located in the command search designation
- *Permission denied* User does not have access permission to use the command or access file
- *Cannot open file* File name is misspelled or doesn't exist; file not located in designated directory

Examples

Example 1 shows incorrect use of the `cp` (copy) command. The command requires source and destination files to be named.

Example 2 illustrates UNIX case sensitivity. All UNIX commands are lowercase. If the command is referenced in uppercase, it is not recognized. Similarly, entering a file name in uppercase is incorrect if the file is named in lowercase when it is created.

In Example 3, the secondary command prompt (`>`) is displayed when a backslash (`\`) character immediately precedes the `<RETURN>` key. The secondary prompt is displayed at other times when the shell is still expecting some input. In a general sense, the `\` character indicates the continuation of the command line over multiple lines. Specifically, the `\` is a special character which directs the shell to ignore the next character, in this case the `<RETURN>` used to signal the end of the command line. The shell continues to read this as a single command line until it encounters a `<RETURN>` not preceded by a backslash.

Command Line Errors

- **Error conditions**
 - Command not found
 - Permission denied
 - Cannot open file

Example 1

```
$ cp
cp: Insufficient arguments (0)
Usage: cp [-i] [-p] f1 f2
        cp [-i] [-p] f1 ... fn d1
        cp [-i] [-p] [-r] d1 d2
```

Example 2

```
$ DATE
DATE: not found
```

Example 3

```
$ cd \
```

Secondary prompt

Controlling the UNIX Session

There may be occasions when typing errors require correction, command lines need to be reentered, or command execution must be interrupted. The keys to perform these functions may vary depending on the current terminal settings. These keys are listed in the table at the right and described below. Current settings may be viewed or altered using the `stty` command. The `stty -lcase` command can also be used to convert uppercase terminal display to lowercase.

Erase Character

The erase key deletes the previous characters, one at a time. Depending on the terminal setup, the `<backspace>` key may be enabled to perform this function also.

Delete Line

The entire command line can be deleted (killed) any time before the `<RETURN>` key is pressed. The cursor moves down to the next line awaiting further input. The shell does not display another shell prompt; the previous line is simply ignored.

Interrupt Command Execution

A command currently executing can be terminated, or discontinued. When the interrupt key is pressed, the operating system sends an interrupt signal to the command and to the shell. The command stops execution immediately. The shell displays another prompt and waits for the next command to be entered.

Reference

- *UNIX System V, Release 4 User's Reference Manual, stty(1)*

Controlling the UNIX Session

Function	Keys
Erase character	^h or BS (F12)
Kill (delete) line	^u
Interrupt command execution	^c

Passwords

The login password was introduced earlier as a UNIX security feature. There is also a `passwd` command that allows a user to list, to set, or to change the login password. This command may also be used by a privileged system user, called a *superuser*, to set, or to change passwords and password attributes associated with any user. Passwords are stored in the `/etc/shadow` file which is readable only by the superuser. It also contains information called *password aging* defining the period during which each user's password is valid. Refer back to page 2-4 to review the rules to select a password before proceeding in this section.

Changing Passwords

The procedure to change a password is similar to setting a password, with the exception that the user is first prompted to enter the old password. The new password must be different from the old one by at least three characters. The `passwd` command checks the `/etc/default/passwd` and/or the `/etc/shadow` files for aging information to determine the period during which the password is valid and if the password can be changed. Superusers are not prompted to supply the old password.

Listing Password Attributes

A user may display password attributes and aging information for her/his logname using the `passwd -s` command. The format of the display is `name status [mm/dd/yy min max warn]` where

<code>name</code>	User login Id
<code>status</code>	Password status: PS indicates password is set, LK indicates the login is locked, and NP indicates no password
<code>mm/dd/yy</code>	Date password was last changed
<code>min</code>	Minimum days before next password change
<code>max</code>	Maximum days password is valid
<code>warn</code>	Days before password expires (relative to max) that user is warned

In the last example on the next page, *user1* has a login password (PS). The password was last changed 6 August 1991. The user will be reminded to change the password seven days before the password expires 168 days after the last change. Refer to the *User's Reference Manual* for further description as needed.

Reference

- *UNIX System V, Release 4 User's Reference Manual, passwd(1)*

Passwords

`passwd [-s]`

- **Changing passwords**

```
$ passwd
Enter old password:
Enter new password:
Re-enter new password:
```

- **Listing password attributes**

```
$ passwd -s
user1 PS 08/06/91 0 168 7
```

Handwritten annotations:

- ↑ *userid* (points to `user1`)
- ↑ *has a password* (points to `PS`)
- ↑ *last pw change date* (points to `08/06/91`)
- ↑ *cannot change pwd for x days* (points to `0`)
- ↑ *must be changed* (points to `168`)
- ↑ *days warning* (points to `7`)

Online Reference Commands

man(1) Command

The **man** command locates and prints the named manual entry. Multiple command names, separated by blanks, may be supplied at the command line.

man uses the **pg** (page) command to display the information one screen at a time. Press <RETURN> to view the next screen, or press <l> to view the next line. Press the letter *h* to display a list of other useful **pg** commands, like searching or scrolling forward and backward.

The examples on the next page illustrate using **man** to access information about a command (**mandex**), to specify a particular manual section for a command (**passwd**) (as described in the front matter), and to designate multiple commands (**who** and **date**).

Reference

- *UNIX System V, Release 4 User's Reference Manual, man(1)*

Online Reference Commands

```
man [section] man_entry
```

Examples

```
$ man mandex
```

```
$ man 4 passwd
```

```
$ man who date
```

: q quit
:/text locate
:.?n

Online Reference Commands

mandex(1) Command

This command provides menu-driven search capability of online reference manuals on selected topics. The basic steps to use **mandex** are listed below .

1. Execute the **mandex** command. Alternatively, keywords may be designated in the command line to access manual pages pertaining to that topic.
2. Select reference manuals to open and/or designate the topic to find or the string of characters to search.

The examples on the following page show two command structures for the **mandex** command. In the first example, the **mandex** menu system is invoked as described above. In the second example, topical keywords are designated at the command line. **mandex** searches and identifies manuals containing the designated keywords.

The initial **mandex** screen listing the available online reference manuals is illustrated on the next page. The prompt *Manual 1 of 4* names the total number of reference manuals listed and the number of manuals currently selected. To move the arrow pointer, use the up and down arrow keys. A list of options is displayed. To select an option, enter the first character of the option in uppercase. **mandex** does not recognize lowercase entries.

The search (S) function can be used to locate subjects in the online manuals. Select (highlight) the manuals to search by moving the pointer (arrow keys) and pressing S to mark the selection. Press C to start the search and designate the search string or keyword(s). **mandex** brings into view manuals containing the designated search pattern.

Reference

- *UNIX System V, Release 4 User' Reference Manual, mandex(1)*

Online Reference Commands

mandex *[keyword(s)]*

Examples

\$ **mandex**

\$ **mandex set password**

Online Manuals Indexing System (mandex)

Book Shelf

Online Manuals

->**User's, Programmer's, Administrator's Reference Manual Pages**

Network User's Reference Manual Pages

X Window Access Commands

Secure UNIX Reference Manual Pages

Manual 1 of 4

Help Quit Open Manual Search

Selection:

Ending a UNIX Session

Two methods can be used to end a UNIX session.

- Press `<^d>` at the shell prompt.
- Execute the `exit` command at the shell prompt.

The terminal displays the logout prompt and information pertaining to the elapsed time since the user logged in.

It is important to log out for the following reasons.

- **Security** – If you remain logged in and leave the terminal area, the system is vulnerable to access by unauthorized users, and your files are available for use by others.
- **Account billing** – If your organization uses system accounting features to charge for system resource utilization, you will be charged for the resource time you remain logged in.

Ending a UNIX Session

- **Press `<^d>`**
- **Execute `exit` command**

Summary

- The *logname* identifies the user to the UNIX system and is used to prevent unauthorized access. It is entered at the login prompt.
- The login procedure requires *logname* and optional password entries.
- The standard shell command prompt is the \$ character.
- The command line format is *command [-options] [arguments]*.
- The <RETURN> key signals the end of a UNIX command line.
- The *shell* processes the command line.
- A command is a *program* read into memory for execution by the kernel.
- A *process* is a command program executing in memory.
- The **passwd** command allows a user to set or to modify user passwords, and to list password attributes.
- Commands providing access to online reference information are **man** and **mandex**.
- To end a session in UNIX, press <^d> or execute the **exit** command.

Practical Exercise

Perform the following activities at your terminal using the logname assigned by your instructor. Where appropriate, record the command line entered to perform the activity. Use available references as needed.

1. Log in to UNIX.
2. Change your password. Log out and log in to verify that the new password is in effect.
3. Use the **who** command to display the users currently logged in. Display this information with column headings. Now, display similar information for only your logname.
4. Enter `datte` and press `<RETURN>`. Record the response.
5. Enter `datte`, but do *not* press `<RETURN>`. Correct the error and press `<RETURN>`. The command should have executed successfully. If not, repeat this step.
6. Enter `datte`, but do *not* press `<RETURN>`. Cancel this command line and execute it again.
7. Use the **date** command to display the current system date and time in the following formats:
 - a. Date as `mm/dd/yy` $+ \%m / \%d / \%y$
 - b. Time as `hh:mm:ss` ~~$+ \%H : \%M : \%S$~~ $+ \%H : \%M : \%S$ -OR- $+ \%T$
 - c. Day of year ~~$+ \%j$~~ $+ \%j$
8. Use the **stty** command to determine the current keys used to erase characters.

`stty -a`
9. Use the **cal** command to determine the day of the week of your birthday in 1992.

`cal ^ 2 ^ 1992`
10. Log out.

Optional Exercise

1. List the two items usually required to log in to UNIX.
userid
password

2. Describe the basic components of a command line.
command options arguments

3. Circle the item(s) that is/are not a function of the shell.
 - a. Issues shell prompt
 - b. Parses the command line
 - c. Locates command program
 - d. Checks program's access permission
 - e. Allocates system resources
 - f. Interprets special characters on the command line prior to execution
 - g. Passes command options and arguments to the command program
 - h. Copies the command program into memory and executes it
 - i. Sleeps until program completes execution

4. Distinguish between the terms *program* and *process*.

5. True or false? Press `<^d>` to signal the end of a command line. If false, explain.
 - a. True signals you off
 - b. False

Optional Exercise

6. Match the function in column B with its corresponding description in column A.

Column A	Column B
— Erases character	a. ^c
— Deletes (ignores) line	b. ^d
— Interrupts command execution	c. ^u
	d. ^h

7. Indicate whether the following statement is true or false. The `passwd` command allows an ordinary user to change any logname password. If false, explain.

- a. True
- b. False — *Security error*

8. Match the description in column B with the corresponding command in column A. Use available reference as needed.

Column A	Column B
— <code>passwd</code>	a. Display active users on the system
— <code>date</code>	b. Access online reference
— <code>mandex</code>	c. Set/change user name
— <code>who</code>	d. View current system date and time
— <code>stty</code>	e. Display user logname
— <code>exit</code>	f. End a UNIX session
— <code>logname</code>	g. View/change terminal settings
— <code>cal</code>	h. Change login password
— <code>man</code>	i. View system calendar
	j. Access menu-driven search index to online manuals

System

3

File Systems

Module Objectives

Upon completion of this module, you should be able to use the UNIX file system.

The supporting module objectives include the ability to

1. Describe the organization of UNIX files.
2. Reference files in the file system using path names.
3. List conventions used to name files.
4. Use shell special characters in file name substitutions.
5. Direct the shell to ignore special meaning of metacharacters.

Reference

Documentation referenced in this module

- UNIX System V, Release 4 System User's Reference Manual (4357 7444-000)
- UNIX Usage Student Guide, Appendix B (UE 7415)

UNIX File System

The UNIX file system is organized into a collection of files, or places to store information. There are four types of UNIX files: directory, ordinary, special, and symbolic link. These files are organized in a certain arrangement so that they can be accessed easily.

UNIX File Types

- A directory maintains information about other files and directories: the file name, who can access the file, the size of the file, when it was created or last modified. Directories are useful in grouping related files together. A directory does not contain data. It points to other files and directories.
- An ordinary file is a collection of characters that reside on some storage media, like a disk. It may contain text for a letter, program code, or any information that is stored for future use.
- A special file, like a directory, does not contain data. It is basically a pointer to a device, like a disk drive, terminal, or printer. Special files associated with devices are located in the /dev directory.
- A symbolic link is an ordinary file that points to another file anywhere in the UNIX file system. *public @USE*

UNIX File Types

- **Directory** d
 - Pointer to other files and directories
 - Does not contain data
- **Ordinary** -
 - Collection of characters stored for future use
 - Contains variable data (text, program code)
- **Special**
 - Pointer to a device
 - Does not contain data
 - Located in /dev directory
- **Symbolic link** l
 - Pointer to another file in the UNIX file system

p named-pipe-file
b block special device
c char ✓ ✓

UNIX File Organization

UNIX files are organized in a hierarchy, or levels, similar to a corporate organizational chart. This arrangement provides an effective way to organize, to retrieve, and to manage information.

At the highest level of this hierarchy is the root directory from which all subordinate files and subdirectories branch. In this sense, the UNIX file organization can be viewed as an inverted tree with the root at the top and all branches descending from it. All other files and directories on the system are connected to the root directory, which is represented by the slash, /.

A subdirectory may contain other files and subdirectories, which may also contain files and subdirectories, and so on. A parent-child relationship exists between a directory and its subordinate files and directories. Each parent directory maintains information about the files and subdirectories one level below it. A user is assigned a unique, initial directory at login called the home directory. As a user changes location in the file hierarchy, each directory location becomes the current directory until the location is changed again.

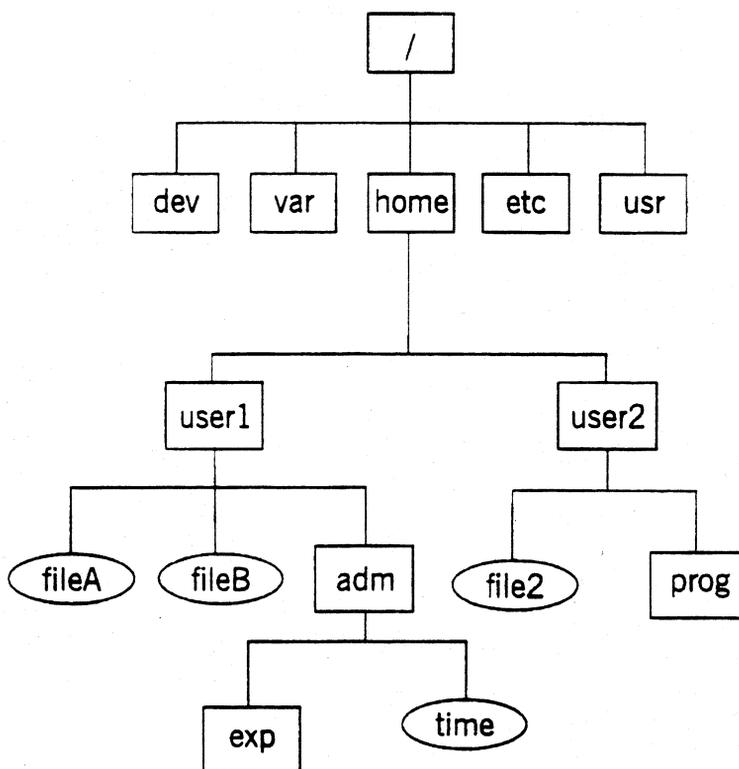
A sample UNIX file hierarchy is provided on the following page. Appendix B contains a listing of major UNIX system directories and files.

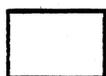
Reference

- UNIX Usage Student Guide, Appendix B, Standard Directories and Files

UNIX File Organization

- Method to organize, retrieve, and manage information
- Arranged in levels – hierarchical
- Supports parent-child relationship between directory and subordinate files and directories



 = Directory

 = File

14401-12

Path Names

A path name uniquely identifies a file or directory. Path names are frequently used as arguments to commands (`cd /etc`), or displayed by commands (like `pwd` or `tty`).

A path name references the location of the file or directory in the UNIX file system. In effect, it specifies the path, either from the root or current directory, to the target file or directory. The path designation includes the names of all directories from the starting directory to the file; each directory name is separated by a forward slash (`/`).

Path names can be designated two ways.

Absolute

An absolute, or full, path name references the location of a file or directory starting from the highest level of the file hierarchy, the root directory. Because the symbol representing the root directory is a `/`, a full path name always begins with a `/` and is followed by the sequence of directory names, each separated from the next by a slash, leading to the desired file or directory.

Using the file hierarchy on the next page, the full path name to the fileA is `/home/user1/fileA`.

Relative

A relative path name references the location of a file or directory relative to the current directory. The path designation does not begin with a `/` because the starting point is not the root directory. It begins with a directory name followed by a slash and the sequence of directory names leading to the target file or directory.

Using the file hierarchy on the next page, the relative path name to the time file from the user1 directory is `adm/time`.

Relative path names can reference files and directories above or below the current directory. Two shorthand notations are useful in relative path name designations. The `.` (dot) is another reference for the current directory. If the current directory is `user`, the path names `adm/time` and `./adm/time` reference the same file. The single dot is more useful in some commands, like `cp subdir1/file1 .`, which copies `subdir1/file1` to the current directory. The `..` (dot dot or double dot) references the parent directory one level above the current directory. Using the file hierarchy on the next page, if the current directory is `adm`, then `..` refers to the `user1` directory.

Multiple `..` notations can be used in a path name, each one referencing the next ascending parent directory. For example, if the current directory is `prog`, then `../..` refers to the home directory. The first `..` references the `user2` directory, and the second `..` references the home directory.

To access files in other directories, the path name must ascend to the parent directory of the target file and then descend. For example, if the current directory is `adm`, the path name to the `user2` directory is `../user2`. The two `..` notations in this path name reference the `user1` and home directories in ascending order, and then descends to the directory `user2`. From the `adm` directory, the path name to `file2` in the `user2` directory is `../user2/file2`.

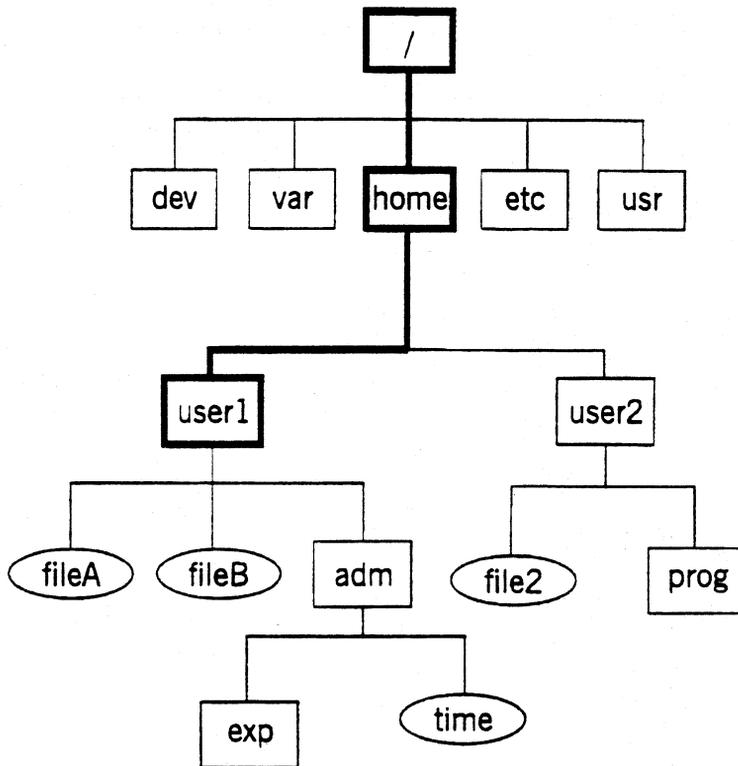
Path Names

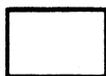
- **Absolute (full) path**

/home/user1/fileA

- **Relative path**

adm/time



 = Directory

 = File

14401-12

Naming Files and Directories

Conventions used in naming files and directories are described below.

- It is advisable to give files and directories meaningful and descriptive names.
- File names should begin with a letter. Avoid using a +, -, or . as the first character.
- Avoid using special characters because they have a different meaning to the shell. Shell special characters are described next.

`? @ # $ ^ & * () ' [] \ | ; ' " < >`

- Do not use spaces in file names. The shell interprets the space as a separator in command lines. Instead, use a period (.) or an underscore (_).
- UNIX distinguishes between uppercase and lowercase letters. For example, `report Report REPORT rePort` are all distinct, different file names.
- File names should be unique within a directory. The same file name can be used, however, in different directories.

File names can be a maximum length of 14 characters for an `s5` file system type and 256 characters for a `ufs` file system type. (File system types are described later in this module.)

File name extensions can be used to provide more meaningful file names. The descriptive extension (suffix) is generally preceded by a period. UNIX uses `.c` to denote C language programs, and `.sh` to refer to shell programs.

Naming Files and Directories

- **Select meaningful, descriptive names**
- **Names should begin with a letter**
- **Avoid using special characters**
- **Do not include spaces in names**
- **UNIX distinguishes between uppercase and lowercase**

Shell Special Characters

Some characters have a special meaning to the shell. These characters are interpreted first when the shell encounters them during command line processing.

Some Bourne shell special characters used throughout this manual are listed on the next page. Special characters used in file name substitutions are described next in this module. Other special characters are described in subsequent modules. These characters are mentioned here so that you avoid using them as regular characters until their meaning is understood.

Note: Other shells use special characters not included here. Refer to the `ksh(1)` or `csh(1)` entries in the reference manuals.

- **File name expansion**

- * Match zero or more characters
- ? Match any single character
- [] Match list or range of characters

- **Command execution**

- or + Denote command options

< Redirect input from named file

input from file

output to file

> Redirect output to named file

add to file

>> Append output to named file

→ output to another program

| Redirect command output to another command

← # of background jobs

& Execute command in background

[1] 10843 ← jobid

; Execute multiple commands in same command line

() Group commands for combined output

(who;ls) > file

“ Back quotes (grave accent) used in command substitution

PWD

PS

PG

- **Remove special meaning**

" " Ignore special meaning of all characters except \$, ' (grave), and \

'' Ignore special meaning of all characters

\ Ignore special meaning of next character

Reference

- UNIX System V, Release 4 User's Reference Manual, `csh(1)` and `ksh(1)`

Shell Special Characters

- **File name expansion**

•

?

[]

- **Command execution**

- or +

<

>

>>

echo \$TERM
echo \$PATH

|

&

;

()

..

- **Remove special meaning**

""

*literal except \$ - *

''

literal

\

ignore ^{next} special char

File Name Substitution

Three special characters allow reference to groups of files or directories in a single command line. These characters are called metacharacters or wild cards because they are substituted by the shell for the actual characters in file names. Recall from Module 2 that special characters are interpreted by the shell before the command program is executed. The command receives the expanded (actual) file names. Metacharacters used in file name expansion are described below.

- **Asterisk**

The `*` character matches zero or more characters. For example, the designation `b*` matches file names beginning with `b`; `*m*` matches files containing an `m` anywhere in the name; `b*e` matches file names beginning with a `b`, ending with an `e` with zero or more characters in between; and `*e` matches file names ending with `e`. An `*` by itself refers to all files in the current directory.

- **Question Mark**

The `?` character matches any single character. For example, `p?` matches any file name having two characters, the first one being the letter `p`. Multiple question marks can be used to match multiple character positions. For example, `???s` matches file names consisting of four characters ending in `s`.

- **Brackets**

The `[]` symbols enclose a list of characters. Any characters listed inside the brackets can match a single character. For example, `report.ju[n]` matches files `report.jun` and `report.jul`. A hyphen separating characters within brackets denotes a range of characters. For example, `memo[1-4]` matches the files `memo1`, `memo2`, `memo3`, and `memo4`. If the first character in the brackets is an `!`, the sense of the match is inverted, that is, it matches any character except those enclosed in brackets. For example, `*[!o]` matches any file that does not end with a lowercase `o`.

These special characters save time in typing and can reduce errors because there is less typing. Importantly, they can be used to reference files when the exact file name is not known.

Examples

The examples at the right show a listing of files using the `ls` command. The special characters described in this section are used to illustrate the output using the sample files.

File Name Substitution

- * Match zero or more characters
- ? Match any single character
- [] Match the list or range of characters

Examples

```

$ ls
File.new      file1      file5
fig           file2      fin
filea        file3      fit
file.new     file4      fun

$ ls fi?      exactly 3 chars
fig
fin
fit

$ ls f??
fig
fin
fit
fun

$ ls *.new
File.new
file.new
g/n in 3rd position

$ ls fi[gn]
fig
fin
5th position

$ ls file[135]
file1
file3
file5
no match 1-4 any other ok.

$ ls file[1-4]
filea
file5

```

Quoting Special Characters

To use a special character literally so that the shell does not interpret its special meaning, enclose the character in a pair of double or single quotes or precede it with a backslash. The shell treats a quoted special character as a regular character. Quoting is particularly important in writing shell programs to control the interpretation of special characters.

- **Double Quotes**

Enclosing special characters in a pair of double quotes, " ", directs the shell to ignore all special characters except the dollar symbol (\$), the back quote or grave accent (`), and the backslash (\).

- **Single Quotes**

Single quotes, ', are more restrictive. All special characters enclosed within single quotes are ignored.

- **Backslash**

Generally, the backslash, \, is the same as enclosing a single character in single quotes. When a backslash is used, it must precede each character being quoted. For example, to remove the special meaning of **, use **.

Examples

The echo command displays designated command arguments on standard output. It is used in the examples to display the effect of quoting special characters.

In the first example, the command line contains the special characters *, ?, and []. Since no quoting is used, the shell interprets these characters and substitutes the appropriate file names.

In Example 2, two special characters are quoted using double quotes. The * is ignored, but the \ is still interpreted by the shell. The shell treats the backslash at the end of a command line as a line continuation. This is used for typing long commands over multiple lines.

Examples 3 and 4 show two methods to direct the shell to ignore the special characters * and \. Enclose the special character in a pair of single quotes or precede each by a backslash.

Quoting Special Characters

Symbol	Description
" "	Shell ignores all special characters except \$, ' (grave accent), and \
' '	Shell ignores all special characters
\	Shell ignores next special character

Example 1

```
$ echo *
File1 File2 fileA fileB newfile testfile
$ echo file?
fileA fileB
$ echo [Ff]ile?
File1 File2 fileA fileB
```

Example 2

```
$ echo "*"
*
$ echo "\"
"
>
<^c>
```

Example 3

```
$ echo '* \'
* \
```

Example 4

```
$ echo \* \\
* \
```

Summary

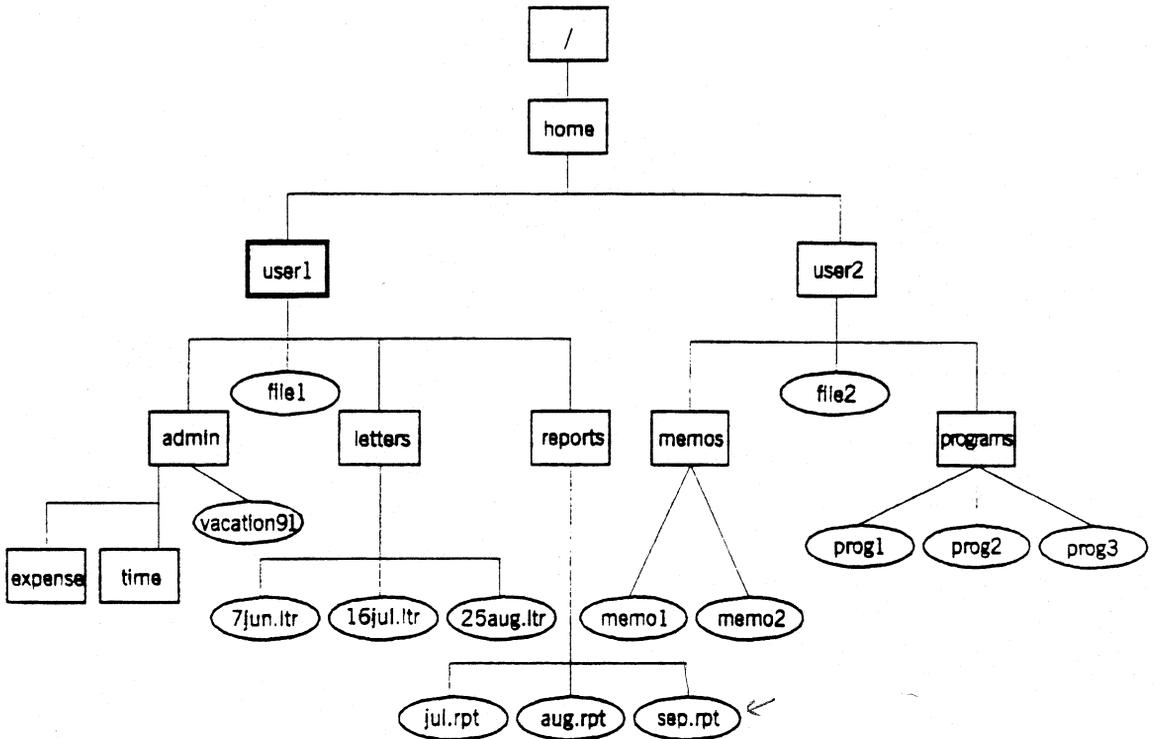
- The UNIX file system is an organized collection of files arranged in a hierarchy.
- The four file types are directory, ordinary, special, and symbolic link.
directory normal device @use
- Directories maintain information about subordinate files and directories.
- The login directory is called the home directory. It is designated for each user in the `/etc/passwd` file.
- The current directory refers to the user's active directory in the file hierarchy.
- The parent directory refers to the directory one level above the user's current directory.
- A path name uniquely identifies a file or directory in the file hierarchy. It can be designated two ways.

absolute (full)	References the location of a file or directory starting from the root directory. The path name always begins with a forward slash.
relative	References the location of a file or directory relative to the current directory. The path name never begins with a forward slash.

- Conventions used in naming files include
 - Select meaningful, descriptive names
 - Names should begin with a letter
 - Avoid using special characters
 - Do not include spaces in names
 - UNIX distinguishes between uppercase and lowercase
- The maximum file name length is 14 characters (s5 file system); 256 characters (ufs file system).
- Special characters used by the shell in file name substitution are
 - * Match zero or more characters
 - ? Match any single character
 - [] Match the list or range of characters
- Quoting directs the shell to ignore the special meaning of metacharacters. This mechanism allows control of metacharacter interpretation by the shell, particularly in programs.
 - " " Shell ignores all special characters except \$ ' (back quote) and \ .
 - ' ' Shell ignores all special characters
 - \ Precedes special character; same as enclosing a single character in single quotes

Exercise

Use the sample file hierarchy provided to write the absolute and relative path names for each file or directory below. Assume the current directory is /home/user1 for each step.



4401-8

1. user1's current directory

Absolute path name : /home/user1

Relative path name : .

2. user1's file1

Absolute path name : /home/user1/file1

Relative path name : file1

3. sep.rpt file

Absolute path name : /home/user1/reports/sep.rpt

Relative path name : reports/sep.rpt

Exercise

4. user1's parent directory

Absolute path name : /homeRelative path name : ..

5. programs directory

Absolute path name : /home/user2/programsRelative path name : ../user2/programs

6. user2's mem01 file

Absolute path name : /home/user2/memos/mem01Relative path name : ../user2/memos/mem01

7. vacation91 file in the admin directory

Absolute path name : ^{/home/user2}
/admin/vacation 91Relative path name : admin/vacation 91

8. user2's file2

Absolute path name : /home/user2/file 2Relative path name : ../user2/file 2

9. 16jul.ltr file

Absolute path name : /home/user1/letters/16jul.ltrRelative path name : letters/16jul.ltr

10. user2's prog3 file

Absolute path name : /home/user2/programs/prog3Relative path name : ../user2/programs/prog3

Optional Exercise

1. What is the UNIX file system?

2. Describe the four types of UNIX files.

directory ordinary
special symbolic-link

3. Match the descriptions in column B with the terms listed in column A.

Column A

Column B

__ home directory

__ current directory

__ parent directory

a. User's active directory

b. Directory one level above current directory

c. Login directory

4. Distinguish between absolute and relative path names used to reference files and directories in the UNIX file system hierarchy.

absolute starts with /

5. Name four conventions used to name UNIX files and directories.

6. Interpret the following file designations using file name metacharacters:

*.?

[Ff]*.rpt

*ltr[5-9]

memo?

*[248]

7. Shell special characters can be used as regular characters (without special meaning) by using apostrophes.

8. Contrast the characters used by the shell to ignore special characters.

9. Which of the quotes below is interpreted by the shell to ignore all special characters?

a. ""

b. ''

10. True or false. Preceding a special character by a \ (backslash) has the same effect as enclosing it in a pair of single quotes.

4

Text Editors

Module Objectives

Upon completion of this module, you should be able to create and to manipulate file text using `ed` and `vi` editors.

The supporting module objectives include the ability to

1. Invoke each editor.
2. Address lines in `ed` line editor.
3. Use cursor and window positioning keys in `vi` screen editor.
4. Differentiate editor operating modes.
5. Manipulate file text using editor commands.
6. Use editor assistance.
7. Cancel editing changes.
8. Modify the editing environment in `vi` screen editor.
9. Save the editing session.
10. Exit the editor.

Reference

Documentation referenced in this module

- UNIX System V, Release 4 User's Reference Manual (4357 7444-000)
- UNIX System V, Release 4 User's Guide (3914 9398-000)

UNIX Text Editors

Text editors are UNIX programs used to create and to modify file text. Unlike word processing programs, text editors do not format text.

UNIX has separate programs to handle formatting of text, tables, or equations. These programs generally require embedded format codes within the text. The coded text file is used as input to a format processor that interprets the embedded codes and formats the text accordingly. A file requiring extensive formatting is handled more easily by using a word processing program that combines text editing and formatting capabilities. Two simple text formatters, `pr` and `fmt`, are described in the next module.

Types of Editors

UNIX provides several text editors. Some editors process text on selected lines (`ed`, `ex`, `edit`); others manipulate text a screen at a time (`vi`, `vedit`). Still others provide restricted capability (`red`, `view`). A file created using one editor can be modified using any other editor.

Most UNIX editors are interactive, accepting instructions and performing the requested functions. The noninteractive stream editor, `sed`, edits a file according to editing commands designated in the command line or contained in a separate program file. This editor is useful to make noncritical, global changes not requiring user interaction. It is frequently used in programs.

Characteristics

<i>Commands</i>	Each text editor recognizes a distinct set of commands, which are not part of the shell command set.
<i>Editing buffer</i>	<p>When a text editor is invoked, a temporary work area called a <i>buffer</i>, is established in memory. It contains text entry and modifications made during the editing session. The contents of the buffer exists only for the current editing session. If the editing session is terminated, the contents of the buffer must be saved or the text and/or changes are lost. Fortunately, the editor issues a message when this is about to happen, reminding the user to save the editing buffer.</p> <p>When an existing file is edited, the file is copied into the buffer. Any changes made to the file copy in the buffer can be saved or discarded to leave the original file intact.</p>
<i>Modes of operation</i>	Editors operate in one of two modes: command or input mode. <i>Input mode</i> allows text entry. <i>Command mode</i> allows text manipulation (copy, move, delete, etc.) In a typical editing session, the user switches between input and command modes as needed. The default mode when editors are invoked is the command mode.

This module describes the use of the line editor `ed` and the screen-oriented editor `vi`. Although `vi` is the more versatile, user-friendly, and, thus, the more frequently-used editor of the pair, the section on `ed` basics is included for students who may, in the future, need to perform system maintenance functions requiring an editor when `vi` is unavailable. Operating system release tapes continue to include `ed` because it is a significantly smaller program to contain on tape while it provides necessary basic editing functions.

UNIX Text Editors

- **Purpose**

- Create and manipulate text
- Do not format text

- **Types**

- Line-based (**ed**, **ex**, **edit**)
- Screen-oriented (**vi**, **vedit**)
- Restricted features (**red**, **view**)

- **Characteristics**

- Distinct command sets
- Editing stored in temporary *buffer*
- Modes of operation
 - Input mode to create text
 - Command mode to manipulate text

Section I: Line Editor `ed`

`ed` is a line editor. Editing commands affect only the current line unless otherwise specified in a line address. When `ed` is invoked, the last line of an existing file becomes the current line so that new text can be added right away.

Creating a File

Follow the steps below to create a file using `ed`.

1. *Invoke the editor by entering `ed file_name`; press `<RETURN>`.*

If a file name is omitted, it is important to provide a file name before exiting the editor or the data will be lost.

`ed` displays `?file_name` to indicate a new file. If the file exists, the number of bytes (characters) in the file is displayed instead.

The editor is now in command mode.

2. *Add text by entering the letter `a`; press `<RETURN>`.*

The text input command, `a`, appends (adds) text after the current line. Enter text and press `<RETURN>` at the end of each line. The editor considers text up to the `<RETURN>` as a single line.

To quickly correct typing errors in input mode, use the `stty` erase character, usually `<^h>`, to delete the character to the left of the cursor. The `<BS-F12>` key can also be used.

Other commands to input and to modify text are described later in this module.

3. *End text input mode by entering a period (`.`) on a line by itself; press `<RETURN>`.*

All input up to the `.` is considered text. The period exits the input mode and returns to command mode.

4. *Save the text in the editing buffer by entering the `w` (write) command; press `<RETURN>`.*

The number of characters saved is automatically displayed.

If a file name was not designated when the editor was invoked, enter a file name after the `w` command separated by a space. For example, `w newfile`.

5. *End the editing session by entering the `q` (quit) command; press `<RETURN>`.*

The shell prompt is displayed indicating that the shell is ready for the next UNIX command.

Reference

- *UNIX System V, Release 4 User's Reference Manual, ed(1)*

Line Editor ed

```
$ ed newfile  
?newfile
```

```
a
```

This sample file is created using ed line editor.

The a command invokes the input mode to enter text.

The . ends text input and returns to command mode.

The w command writes the buffer to the named file.

The q command exits the editor.

```
.
```

```
w
```

```
256
```

```
q
```

```
$
```

Getting Assistance

The following commands provide assistance during the editing session.

P Displays a * prompt in command mode.

ed does not automatically display prompts to indicate which operating mode is active. It is helpful to issue this command as soon as the editor is invoked so that the * prompt is displayed whenever the command mode is active.

Alternatively, the **ed** option **-p prompt** issued at the shell prompt allows the user to designate a command prompt other than the default *.

h or **H** Displays a help message for the most recent error, represented by the ? symbol.

ed does not automatically display error messages. When errors are encountered, **ed** displays a ? symbol. The **h** command displays a brief help message for the most recent error.

Alternatively, it is advantageous to invoke continuous help throughout an editing session by executing the **H** command as soon as the editor is invoked. Continuous help can be turned off at any time by pressing **H** again.

Getting Assistance

Command	Description
P	Display * prompt for command mode
h	Display a help message for last error
H	Provide continuous help messages for editing session

Practical Exercise

Create a file in `ed` called `edfile.xx` (where `xx` are your initials) using the steps listed on page 4-4. Include the following text. Be sure to press `<RETURN>` at the end of each line. Save the buffer contents before ending the editing session and returning to the shell. This file will be used in subsequent exercises.

Text:

This is line 1 of text.

This is line 2 of text.

This is line 3 of text.

This is line 4 of text.

This is line 5 of text.

`<TAB>`This is line 6 of text.

`<TAB><TAB>`This is line 7 of text.

`<TAB><TAB><TAB>`This is line 8 of text.

This is line 9 of text.

This is line 10 of text.

ed Command Format

The format of an ed command appears below.

[address]command[argument(s)]

where:

address	Specifies a particular line or range of lines in the buffer affected by the command that follows. If a line address is omitted, the command affects the current line in the editing buffer. The <i>current line</i> refers to the active line in the editing buffer. This may be the last line when the file is opened in the editor or the last line referenced in a command.
command	Designates a single character as the editing action. If a command is omitted, the default p (print) command is executed.
argument(s)	Provides additional information required by some commands, typically a file name or another line address.

Examples

The first example illustrates the **w** (write) command to save the contents of the buffer.

The second example designates a file name argument to the **w** command. Regardless of the file specified when the editor is invoked, this command saves the changes in the buffer to another file. This is a useful method to keep versions of the same file separate.

In the last example, the part of the buffer designated by the line address is saved to the named file. The designation *1,8* refers to lines 1 through 8 of the file.

Reference

- *UNIX System V, Release 4 User's Guide, Appendix D, Quick Reference to ed Commands*

ed Command Format

[address]command[argument]

Examples

w<RETURN>

w newfile<RETURN>

1,8w file2<RETURN>

Line Addressing

A *line address* designates the line or range of lines to be affected by the command that follows it. If a line address is omitted, the command affects the current line in the buffer.

The line address may be numeric, symbolic, or a combination. `ed` assigns a line number reference to each line relative to its position (sequence) in the buffer. These line numbers may reference different text as text is moved, copied, or deleted. Line number 0 can be used in line addresses to refer to the beginning of the file. Symbolic line addressing is useful when the exact line number is unknown.

.	Current line
\$	Last line
+	Next line
-	Previous line

The *comma* indicates a range of lines in an address. For example,

2,5	Lines 2 through 5
.,7	Current line through line 7
.,\$	Current line through the last line
1,\$	Line 1 through the last line (the entire file)

Line Addressing

- **Designates line(s) affected by ed command**

- **Address types**
 - Numeric
 - Symbolic
 - Combination of numeric and symbolic

- **Line range indicator is the comma (,)**

Displaying Lines

Recall that **ed** commands are single-character designations.

Several commands can be used to display text. Again, if a line address is omitted, the command affects the current line in the buffer.

p Print the current or designated lines of text.

For example, **1,\$p** displays all lines of text in the editing buffer. This is the default **ed** command if no other command is designated. The designation **,p** is another way to reference all lines.

n Displays the current or designated lines of text with relative line *numbers*.

For example, **1,\$n** displays all lines of text with line numbers. The designation **,n** is another way to reference all lines.

l Display the current or designated lines of text *listing* invisible characters, like tabs and indents. Each tab is represented by the **>** symbol.

For example, **5,9l** displays lines 5 through 9 with invisible characters. The designation **,l** is a shorthand method for the same designation.

= Display the line number of a line of text. For example, **. =** prints the line number of the current line and **\$=** prints the line number of the last line.

/pattern Display the next line containing the designated *pattern* of characters.

To continue searching forward for the same pattern, enter **/** and press **<RETURN>**. When the last occurrence has been located in the file and the search is repeated, **ed** will circle (or wrap) to the beginning of the file to continue the forward search.

?pattern To reverse the direction of the search, enter **?** and press **<RETURN>**. To search backward for a different pattern of characters, enter **?new_pattern** and press **<RETURN>**. When the last backward occurrence has been located in the file and the search is repeated, **ed** will circle (or wrap) to the end of the file to continue the backward search.

Displaying Lines

Command	Description
p	Display lines of text
n	Display text lines with numbers
l	Display text lines with unprintable characters
=	Display line number
/<i>pattern</i>	Display next line containing <i>pattern</i>
?<i>pattern</i>	Display previous line containing <i>pattern</i>

Practical Exercise

Invoke the editor using the *edfile.xx* created in Exercise 1. Write the line address and command used to perform each step below.

1. Print the contents of the file two ways, using a complete address and the shorthand method.
2. Print the current line. Now, display the last line with its line number.
3. Display lines 6 through 8. Now, print the same line range displaying any unprintable characters. How are the two displays different? Which symbol represents the tab?
4. Print the previous line. Now, display the next line.
5. Use the search command for the following steps.
 - a. Print the next line containing the character pattern *line*.
 - b. Continue the search for the same pattern until the end of the file is reached. Repeat the search again to force **ed** to wrap to the beginning of the file to find the next line containing the pattern.
 - c. Search for the previous occurrence of *line*. What happens when you repeat a backward search beyond the first line?

Entering Text

The three text input commands are:

- a** Append (add) text
- i** Insert text
- c** Change (replace) text

Append (a)

The *append* command, **a**, was introduced earlier to add text to a file. The **a** command adds text after the current or designated line address. Recall from the description of **ed** command format, that an optional line address can be specified to affect lines other than the current line. Line numbers shift to accommodate the change in the text sequence. For example, **4a** appends text after line 4.

Insert (i)

The *insert* command, **i**, inserts text before (above) the current or designated line address. An optional line address can be specified to affect lines other than the current line. For example, **4i** inserts text before line 4.

Change (c)

The *change* command, **c**, replaces the current or designated line(s). An optional line address can be specified to affect lines other than the current line. For example, **4c** changes or replaces line 4.

Corrections can be made during input mode on a given line using the **stty** erase character, typically **<^h>**, or **<BS-F12>**. This correction method is effective only on the active line containing the cursor before the **<RETURN>** key is pressed. Other commands to correct and to modify text are described subsequently.

To exit any input command mode, enter a period by itself on a line followed by a **<RETURN>**.

Entering Text

Command	Description
a	Append text after current or designated line
i	Insert text before the current or designated line
c	Change (replace) the current or designated line(s)
.	Exit input mode and return to command mode

Practical Exercise

Invoke the editor using *edfile.xx*. Write the command used to perform each text input indicated below. Be sure to exit the input mode after each entry.

1. Insert your *name* above line 5.
2. Replace line 8 with your *birthday*.
3. Add your *job title* to the beginning of the file.
4. Add your *city name* after line 2.
5. Insert your *street address* above the last line.

Discard these changes in the buffer and exit the editor by issuing the **q** command twice. An error prompt, **?**, is displayed after the first **q** command indicating that **ed** normally expects to write the contents of the buffer before exiting. The second **q** command exits the editor without saving the buffer contents.

Modifying Text

Primary commands to modify text include:

u	Undo the last change
d	Delete text
m	Move text
t	Copy (transfer) text
s/old/new	Substitute <i>old</i> text with <i>new</i> text

Undo Last Change (u)

One of the most important and useful commands, **undo** reverses, or restores, the last change. This command presents a more convenient alternative to discarding all editing changes and quitting the editor when only an occasional change needs to be reversed. It is important to note that **undo** only affects the last change. Pressing **u** repeatedly toggles (alternates) between reversing and recreating the last change. For example, the effect of the command **1,\$d**, which deletes the entire contents of a file, can be easily reversed within the editing session by executing the **u** (undo) command.

Delete (d)

The *delete* command, **d**, deletes the current or designated line(s). An optional line address can be specified to affect lines other than the current line. For example, **2,4d** deletes lines two through four.

Move (m)

The *move* command, **m**, requires two line addresses. The address of the line (or line range) to be moved precedes the command; the line number following the command specifies the target location. For example, **.,\$m0** moves the current line through the last line *after* line 0, or to the beginning of the file.

Copy (t)

The *copy* (transfer) command, **t**, also requires two line addresses. The address of the line (or line range) to be copied precedes the command; the line number after the command designates the target location. For example, **.,\$t0** copies the current line through the last line to the beginning of the file.

A variety of other commands can be used in **ed**. Descriptions of these commands are deferred to the **vi** section because they are also available in the screen editor.

Modifying Text

Command	Description
u	Undo (reverse) last change
d	Delete text
m	Move text
t	Copy text
s/old/new	Substitute text

Substituting Text

The editing commands described so far affect entire lines. The *substitute* command, *s*, replaces text within lines.

The format of the substitute command is:

```
[address]s/old_text/new_text/command
```

where:

<i>address</i>	Line(s) to be substituted
<i>s</i>	Substitute command
<i>/old_text</i>	Text to be replaced
<i>/new_text</i>	Replacement text
<i>/command</i>	Optional command(s)

The components of the substitute command are usually delimited (separated) by *slashes*. However, other characters can be used, providing the same character is used throughout a given substitute command.

By default, only the first occurrence of *old_text* is replaced on a given line, unless other optional commands designate otherwise.

For example, using *edfile.xx* created earlier, the command *s/is/was* substitutes the first occurrence of *is* with *was* on the current line and displays the line. The result may be unexpected since the first occurrence of *is* appears in the word *This*, which now appears as *Thwas*. (Fortunately, the undo command quickly reverses this undesirable change.)

If the command includes a line range before the *s* command, the replacement affects all lines specified in the address. For example, *1,\$s/is/was* replaces all first occurrences of *is* with *was* and displays the last line changed. In our sample file, all lines would begin with *Thwas*. (Time again for the undo command.)

To substitute all occurrences of *old_text* with *new_text*, designate the (optional) global command, *g*, after the last delimiter. For example, *s/is/was/g* replaces all occurrences of *is* with *was* on the current line. The current line becomes *Thwas was line n of text*. Similarly, *1,\$s/is/was/g* replaces all occurrences of *is* with *was* in the entire file. (The undo command is becoming increasingly useful.)

An alternative method used in substitution is *g/string/s/old/new*. This format substitutes *old* with *new* globally in the file, but only in lines containing *string*.

Substituting Text

Command	Description
s/old_text/new_text	Replace first occurrence of <i>old_text</i> with <i>new_text</i> and display line
[.,.]s/old_text/new_text	Replace first occurrence of <i>old_text</i> with <i>new_text</i> on addressed lines and display last line
[.,.]s/old_text/new_text/g	Replace all occurrences of <i>old_text</i> with <i>new_text</i> on addressed lines and display last line

Escaping to the Shell

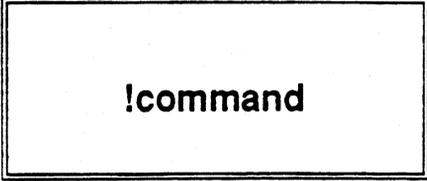
During an editing session, it may become necessary to execute a UNIX command. The escape command, `!`, allows temporary access of the shell to execute one or more commands.

The `!` is entered in command mode and is followed by the desired UNIX command and `<RETURN>`. For example, `!ls<RETURN>`. The designated command is executed and its output is displayed on the screen. A single `!` following the command output is displayed indicating that the editing session has resumed. The command output does not become part of the file.

To execute multiple commands, separate each command in the series by a semicolon (`;`). For example, `!pwd ; date; who` executes the designated commands in sequence and displays all output before resuming the editing session.

The designation `!sh` exits to the shell until `<^d>` is pressed or the `exit` command is executed from the shell. A message is displayed to press any key to resume the editing session.

Escaping to the Shell



!command

Examples

!date

!who ; date; pwd

!sh

ed Recovery

When an editing session terminates abnormally, UNIX attempts to save the contents of the editing buffer to the file *ed.hup* in the current directory. Once the contents of the file is checked, it should be renamed as soon as possible because the file *ed.hup* will be overwritten at the next recovery.

ed Recovery

- ***ed.hup* file**
 - Contains contents of editing buffer
 - Located in current directory
 - Rename file to save edited data

Practical Exercise

Invoke the editor using *edfile.xx*. Write the commands used to perform each of the following steps below.

1. Delete the last line. Display all lines to verify that the last line is deleted. Restore the line and verify that text is restored.
2. Move lines 3 through 5 after line 7. Verify the results by displaying the file with line numbers. Notice the line numbers reflect the new position of text. Move the text back to the original location. Verify that the lines are in the original sequence.
3. Copy the last three lines to the beginning of the file. Display the file text to verify the duplicate lines. Reverse this change and verify.
4. Substitute the word *is* with *was* on lines 3 through 6. Verify the substitution by displaying the lines. Restore the line and verify.
5. Substitute all occurrences of *line* with *sentence* in the entire file. Display the results. Reverse this change and verify.
6. Escape to the shell to display the date and time.
7. Discard all edits before quitting the editor.

Section II: Screen Editor vi

This section describes how to create and to edit files using vi.

vi (visual) is an interactive screen editor. Unlike ed, vi displays a window (screen) of text at a time. It allows the user to move the cursor to any point on the screen or in the file and to scroll text forward or backward to reveal text above or below the current window.

Generally, vi is a more versatile, flexible editor than ed. The more distinct operating features of vi include:

- When vi is invoked, an editing buffer is established to contain text input and editing.
- vi operates basically in two modes like ed: *text input mode* and *command mode*. Text input mode is used to enter text into a file. Command mode is used to manipulate (edit) text. vi has another command mode called *last line* (or colon) mode allowing the use of additional commands.
- vi commands are not followed by a <RETURN> like ed commands. Also, vi commands do not generally display on the screen.

The commands described in this section represent basic and useful commands. However, vi offers an extensive array of commands. The user is encouraged to explore the diverse palette of vi commands as experience develops, and as need dictates.

Reference

- *UNIX System V, Release 4 User's Reference Manual, vi(1)*

Screen Editor vi

- **Window displays screen of text at a time**
- **Buffer contains text input and editing**
- **Operating modes include**
 - Command mode (default)
 - Text input mode
 - Last line mode to use additional commands
- **Distinct set of commands**
 - Commands are not followed by <RETURN>
 - Commands are not displayed on screen

vi Setup

Because **vi** is dependent on the specific characteristics and capabilities of each terminal, it is necessary to identify the type of terminal being used. If a terminal is not defined, **vi** displays an appropriate message when the editor is invoked.

To define the terminal type, assign the appropriate terminal name from the file */etc/termcap* (or the linked *usr/share/lib/termcap* file) to the shell variable **TERM** and export the value to other processes requiring this information. For example, to identify a UVT-1224 terminal, enter the following line at the UNIX shell prompt:

```
TERM=uvvt1224 ; export TERM
```

Note: Shell variables are all uppercase; spaces are not included between the variable name and its value. The variable assignment and **export** command can also be designated on separate command lines.

The terminal designation is effective only for the current login session. To define the terminal automatically for each login, include the entry above in the *.profile* in the home directory.

Other **vi** environment options will be described later in this module.

vi Setup

- Define terminal type
- Format

```
TERM=terminal_type  
export TERM
```

Example 1

```
$ TERM=uvvt1224  
$ export TERM
```

Example 2

```
$ TERM=uvvt1224 ; export TERM
```

Creating a File

The procedure to create a file in **vi** is similar to the procedure used in **ed**.

1. *Invoke the editor by entering **vi file_name**.*

If a file name is omitted, be sure to provide a file name before exiting the editor. **vi** displays the reminder message, *No current filename*, to request a file name.

The screen display appears different from **ed**. The cursor appears on the top line followed by a series of tildes (~) marking empty lines. The file name appears on the last line followed by a designation within brackets specifying a new file or the number of characters in an existing file.

The editor is in command mode.

2. *Add text by entering an input command, like **a** (append); do not press <RETURN>.*

Notice the letter **a** does not appear on the screen. Enter text and press <RETURN> at the end of each line.

3. *End the input mode by pressing the <ESCAPE> key.*

All input up to the <ESCAPE> is considered text.

4. *Save the contents of the buffer and exit the editor.*

vi offers several commands to write the buffer contents and to exit the editor. If a file name was designated when **vi** was invoked, the **ZZ** command combines writing the buffer and exiting **vi**. You can also use the line editor commands, **w** and **q** preceded by a colon, like **:w** or **:q**, or combine them into a single command, **:wq**. Notice that when a **:** is entered, the cursor moves to the last line on the screen. The commands following the colon also appear in this location. Press <RETURN> to execute the designated command.

Refer to the next page to view a sample **vi** screen during an editing session.

:q to exit

Creating a File

```
$ vi testfile
```

```
Use an input command to enter text.  
Press <RETURN> at the end of each line.  
Press <ESCAPE> to exit the input mode.  
Save the editing changes in the buffer and  
exit the editor.
```

```
<ESCAPE>
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
:wq<RETURN>
```

```
"testfile" [New file] 5 lines, 174 characters
```

Practical Exercise

Create the following files using vi editor. Be sure to press <RETURN> at the end of each line. Save the buffer contents before ending the editing session and returning to the shell. These files will be used in subsequent exercises.

1. Create a file called *vi*file.xx (where xx are your initials) using the steps listed on page 4-36. Enter the following text as it appears below. Errors will be corrected in subsequent exercises.

Advantages of Exercise

Ninety minutes of daily exercise, such as walking, dancing, or swimming is all it takes to degrade the body that is already poorly prepared to handle daily stress.

It has been proven that rrrrregular daily exercise cannot improve work performance, or overall mental aptitude.

As body circulation diminishes, one's general attention suffers as well. You look and feel worse reflecting a more opsitive altitude.

2. Create another file called *numbers* containing the numerals 1 through 50, one number per line along the left margin.

Positioning the Cursor

The cursor can be moved on the screen several ways. The keys to position the cursor on the screen are listed in the table at the right.

To move in increments of characters or lines, precede the cursor movement key with the desired number. For example, `5<h>` moves the cursor five characters to the left.

Reference

- *UNIX System V, Release 4 User's Guide, Appendix E, Quick Reference to vi Commands*

Positioning the Cursor

Direction	Key
Left	< h > Left arrow <backspace>
Down	< j > Down arrow
Up	< k > Up arrow
Right	< l > Right arrow <spacebar>

Positioning the Cursor

In addition to moving by character and line position, the cursor can be moved by word, sentence, or paragraph segments, as well as to other locations on the screen.

Most of the commands listed in the table at the right can be preceded by a number designating an increment.

Reference

- *UNIX System V, Release 4 User's Guide, Appendix E, Quick Reference to vi Commands*

Positioning the Cursor

Text Segment	Symbol	Description
Word	w	Forward one word
	e	End of current word
	b	Back one word
Sentence	(Beginning of previous sentence
)	Beginning of next sentence
Paragraph	{	Beginning of previous paragraph
	}	Beginning of next paragraph
Screen	H	Home (top of screen)
	M	Middle of screen
	L	Last line on screen
File	G	End of file
	1G	Beginning of file
	nG	Line number designated by <i>n</i>

Scrolling Text

Text that is not in the current window can be displayed, or scrolled, into view using the commands listed in the table on the next page.

Press `<^f>` to scroll forward through a file to display the next screen, or press `<^b>` to scroll backward to display the previous screen. Alternatively, press `<^d>` to scroll down through the file to display the next half screen, or press `<^u>` to scroll up (backward) through the file to display the previous half screen of text. Similarly, press `<^e>` to display the next line at the bottom of the screen, or press `<^y>` to display the previous line at the top of the screen.

Reference

- *UNIX System V, Release 4 User's Guide*, Appendix E, Quick Reference to vi Commands

Scrolling Text

Scroll by	Command	Direction
Full Screen	^f	Forward
	^b	Backward
Half Screen	^d	Down
	^u	Up
Line	^e	Next line
	^y	Previous line

Searching for Text

The search commands provide a quick way to reposition the cursor to a specific location in the file containing the designated character pattern.

The search commands in vi are basically the same as the search commands in ed. There are two additional commands, n and N. Search commands are described below.

- /pattern** Search forward for designated pattern of characters. Repeat search for same pattern by entering /.
- ?pattern** Search backward for designated pattern of characters. Repeat backward search for same pattern by entering ?.
- n** Repeat the previous search in the same direction
- N** Repeat the previous search in the reverse direction

all cases.
to locate a string &
remove the line

:g/text/d

Searching for Text

Command	Description
<i>/pattern</i>	Search forward for <i>pattern</i>
/	Repeat forward search for same pattern
<i>?pattern</i>	Search backward for <i>pattern</i>
?	Repeat backward search for same pattern
n	Repeat previous search in same direction
N	Repeat previous search in reverse direction

Practical Exercise

Invoke the editor using the *vi*file.xx created earlier.

1. Use the cursor positioning keys listed on pages 4-40 and 4-43 to move the cursor on the screen by characters, words, sentences, paragraphs, etc.
2. Use vi's search feature to search forward for the pattern *daily*. Use the **n** and **N** commands to repeat the search for the same pattern in the same and the opposite direction.
3. Exit the editor.

Invoke the editor using the *numbers* file.

4. Use the keys described on page 4-43 to position the cursor at the end of the file, at the beginning of the file, and to a specific line number.
5. Use the keys described on page 4-44 to scroll text into the current window by full screen, half screen, and one line at a time.

Getting Assistance

The commands described below provide assistance during the editing session.

:set showmode Displays status message for the current text input mode in the lower right area of the screen. Status message remains displayed during the current input mode. When no status message is displayed, vi is in command mode.

This feature is active for the current editing session. Methods to automatically display input prompts when vi is invoked are described later in this module.

u Like the ed command, **undo** restores the last change in the editing buffer.

U Restores all changes on the current line containing the cursor.

. Repeats the last editing change. The redo **.** (period) command is helpful when the same change is made in several places within a file. Perform the editing once to store the change in the buffer. Thereafter, reposition the cursor to the desired location and use the **.** command to repeat the last edit. The change can be "reused" until another change replaces it in the buffer.

A
I
o
O
Append
Insert
Add a line } go to text mode

Getting Assistance

Command	Description
:set showmode	Display current input mode
u	Undo last change
U	Undo all changes on current line
.	Redo last change

Entering Text

vi offers several commands to input text. They are summarized below. After entering text, press <ESCAPE> to exit input mode before executing other vi commands. Remember that typing errors can be corrected on a line in input mode using the stty erase key, typically <^h>, or <BS-F12>.

- a** Append (adds) text after the cursor
- A** Append text after the line
- i** Insert text before the cursor
- I** Insert text before the line
- o** Open a line below the current line and enter text until the <ESCAPE> key is pressed
- O** Open a line above the current line and enter text until the <ESCAPE> key is pressed
- r** Replace a single character at the cursor
- R** Replace text continuously until the <ESCAPE> key is pressed

Entering Text

Function	Command	Description
Append	a	Add text after cursor
	A	Add text after line
Insert	i	Insert text before cursor
	I	Insert text before line
Open line	o	Open line below cursor and enter text
	O	Open line above cursor and enter text
Replace	r	Replace single character at cursor
	R	Continuous replacement until <ESCAPE>

D = erase to end-of-line

Practical Exercise

Invoke the editor using the *vi* file *xx* for this exercise using the text input commands. Remember to press <ESCAPE> to exit input mode.

Turn on the text input prompt. Observe the different prompts for each input command as you perform the following steps.

Position the cursor midway on any line for each step below to observe any cursor movement as each command is invoked.

1. Enter *your name* after the cursor. Exit input mode and undo this change.
2. Enter *your name* at the end of the line. Exit input mode and undo this change.
3. Insert *your name* before the cursor. Exit input mode and undo this change.
4. Insert *your name* at the beginning of the line. Exit input mode and undo this change.
5. Open a line below the current line and enter *your name*. Exit input mode and undo this change.
6. Open a line above the current line and enter *your name*. Exit input mode and undo this change.
7. Replace the character at the cursor with *any number*. Notice the input message is removed as soon as the replacement character is entered. Undo this change.
8. Replace the text starting at the cursor location with *your name*. Exit input mode and undo this change.
9. Quit the editor without saving changes using the **q** (quit) command. Notice that **vi** displays a reminder message that the changes in the buffer have not been saved. The command **:q!** overrides saving the buffer contents and exits the editor discarding all changes.

Modifying Text

The primary functions that modify text are described below. The commands that perform these editing functions are presented individually on the following pages.

- Delete* Delete one or more characters or a text range.
- Move* Two-command procedure to delete designated text into the buffer and put it at the cursor position.
- Copy* Two-command procedure to copy designated text into the buffer and put it at the cursor position.
- Substitute* Replace one or more characters with text input until <ESCAPE> is pressed.
- Change* Replace designated text segment with text input until <ESCAPE> is pressed.

Deleting Text

- **Input mode**

- **stty** erase character, usually `<^h>`
- `<BS-F12>`

- **Command mode**

- **x** command deletes characters
- **d** command deletes designated text segment

*Delete
char
key*

*press space to delete
one char under cursor*

*dw delete word
d\$ eol
d\$ delete start of line to cursor
dd delete line
3dd delete 3 lines*

Moving Text

Moving text involves three steps.

1. *Delete the text into the temporary buffer.*

Delete the desired text segment using the **d** command followed by the appropriate text range symbol. Again, to affect multiple text segments, precede the **d** command with a number representing the increment. For example, **2dd** deletes two lines (current and the next lines) into the buffer.

2. *Move the cursor to the desired location.*

3. *Place the text at the desired location in the file.*

Use the **p** (put) command to place the text *after* the cursor. This command should be used immediately after the delete command since the buffer stores only the last change. Similarly, the **P** (uppercase) command puts the deleted text *before* the cursor.

As long as this change remains in the buffer, the **put** commands can be issued repeatedly to recall the change at various locations throughout the file.

VI control file (like entry\$ in @IPF)

VI ~~EXRC~~
^
exrc

must not
have any blank
lines

vi^ .exrc

:set all
to view settings

Moving Text

- **Procedure**

1. **Delete text into temporary buffer with `d` command**

@LND

2. **Move cursor to desired location**

3. **Place text in new location**

- Put text after cursor with `p` (lowercase) command

@LND

- Put text before cursor with `P` (uppercase) command

- **Same text can be placed repeatedly in other locations in the file**

Copying Text

Copying text involves a similar procedure.

1. *Yank (copy) the text into the temporary buffer.*

Yank (copy) the desired text segment using the **y** command followed by the appropriate text range symbol. Again, to affect multiple text segments, precede the **y** command with a number representing the increment. For example, **2yy** copies two lines (current and the next lines) into the buffer.

2. *Move the cursor to the desired location.*

3. *Place the text at the desired location in the file.*

Use the **p** (put) command to place the text *after* the cursor. Use this command after the **yank** command since the buffer stores only the last change. Again, the **P** (uppercase) command puts the deleted text *before* the cursor.

As long as this change remains in the buffer, the **put** commands can be issued repeatedly to recall the change at various locations throughout the file.

Copying Text

- **Procedure**
 1. **Yank (copy) text into temporary buffer with `y` command**
 2. **Move cursor to desired location** @LNY
 3. **Place text in new location**
 - Put text after cursor with `p` command @LNP
 - Put text before cursor with `P` command
- **Same change can be copied repeatedly to other locations in the file**

Substituting Text

This command combines the delete and append functions by replacing one or more characters on a line with text input until the <ESCAPE> key is pressed. A \$ symbol appears in place of the last character to be replaced marking the end of the text replacement. For example, **4s** deletes four characters starting at the cursor position and invokes the input mode to append replacement text of any length until the <ESCAPE> key is pressed. The **substitute** commands are described in the table at the right.

Alternatively, the line editor command can be invoked from last line mode by preceding the command with a colon. Position the cursor before executing the command or use a line address in the command as needed. For example, **:1,\$s/help/assistance/g** replaces all occurrences of *help* with *assistance*.

The **substitute** commands do *not* accept a text range symbol to replace words, sentences, paragraphs, etc. The **change** command, discussed subsequently, is useful to replace these larger text segments.

Substituting Text

Command	Description
s	Replace single character and append text
ns	Replace <i>n</i> characters and append text
S	Replace entire line

First occurrence on each line

{

 :s/ from /to / current line

 :2,10 s/ from /to / line 2 - 10

 :1,\$ s / from /to / all lines

 :.,.+3s/ from /to / current lines → line+3

 :s/ from /to /g all occurrences.

Changing Text

The **change** command, **c**, is similar to the **substitute** commands. The **c** command replaces the designated text segment with text input until the <ESCAPE> key is pressed. Again, the **\$** symbol appears in place of the last character to be replaced marking the end of the text replacement. For example, **3cc** replaces three lines of text starting at the line containing the cursor with the text entered until the <ESCAPE> key is pressed.

Examples

In the first example, **cw** changes the word containing the cursor with text entered until the <ESCAPE> key is pressed. In Example 2, **7cc** removes seven lines starting with the current line, and invokes the input mode for text entry until the <ESCAPE> key is pressed. The last example illustrates a different text range; **4c)** removes four sentences and allows text entry until the <ESCAPE> key is pressed.

Changing Text

- Replaces text identified by text range symbol

Examples

cw ← delete word

7cc ← delete 7 lines

4c) ← delete 4 sentences.

Practical Exercise

Invoke the editor using the *vifile.xx* to perform the editing below. Write the commands used below each item. Use the search command to position the cursor quickly to the area requiring correction.

- Edit paragraph one following the instructions below.
 1. Substitute the characters *ninety* with *thirty*.
 2. Change the text specifying physical activities to activities of your choice.
 3. Change the phrase *degrade the* with *maintain a healthy and fit*.
 4. Replace the text *already poorly* with *better*.
 5. Add the text *and to ward off illness* at the end of the sentence.
 6. Move this paragraph after paragraph two.

- Edit the paragraph beginning with *It has been proven . . .* using the instructions provided below.
 7. Delete the extra *r*'s in the word *regular*.
 8. Delete the characters *not* in the word *cannot*.
 9. Replace the character *p* with *t* in the word *aptitude*.

- Edit the third paragraph following the instructions below.
 10. Substitute the words *diminishes* with *improves*, and *worse* with *better*.
 11. Change the phrase *attention suffers* with *alertness improves*.
 12. Correct the spelling of the last two words to *positive attitude*.

Compare the completed text with the edited version below. Make any additional corrections needed. Write the corrections to the file *vifile.ok*.

Advantages of Exercise

It has been proven that regular daily exercise can improve work performance, or overall mental attitude.

Thirty minutes of daily exercise, such as walking, dancing, or swimming is all it takes to maintain a healthy and fit body that is better prepared to handle daily stress and to ward off illness.

As body circulation improves, one's general alertness improves as well. You look and feel better reflecting a more positive attitude.

Using Line Editor Commands

Line editor commands can also be used in **vi** by preceding the line editor command with a colon, (:). When the colon is pressed, the cursor moves to the last line on the screen and displays the command as it is being entered; press **<RETURN>** to execute the command.

Frequently used line editor commands are described below. Commonly used variations of these commands are also included in the descriptions. For complete command descriptions, refer to the **vi(1)** entry in the User's Reference Manual.

:!command

exec"

like calling
up a copy
of
command.com

Allows temporary access of the UNIX shell to perform one or more shell commands. Multiple commands are separated by a semicolon (;). After the output of the last command is displayed, the message *[Hit return to continue]* is displayed to indicate that the editing session has resumed.

To escape indefinitely to the shell, enter **:sh**. The shell prompt indicates access of the UNIX shell. To return to the editing session, press **<^d>** or execute the shell command **exit**.

:f

\$file :=

Display the current file name, the current line number containing the cursor, and the percent of the file at the cursor location. The command **:f newfile** renames the current file in the buffer.

:r

etc
append
@add

Read in the contents of the current or named file below the current line. The command **:r !command** reads in the output of the designated command below the current line.

:e

old

Replace the current file in the buffer with the named file. Issues a warning message if the current file was modified and needs saving before replacing the file in the buffer.

The command **:e!** discards all changes to the current file and continues the editing session without exiting the editor. Similarly, **:e! newfile** discards the changes pertaining to the current file and replaces the file in the buffer with the named file.

:w

EXIT
REP

Write the changes in the buffer to the current file. Precede the command with a line address to save only the designated lines to a file. In this case, another file name is usually designated after the **w** command to keep the original file intact and the file "versions" separate.

When attempting to (over)write an existing file, **vi** issues a warning message that the file exists. The **:w!** command overwrites the named file with the contents of the editing buffer.

:q

x

Exit the editor. Generally, this command is used with or after the write command. The command **:q!** exits the editor discarding all changes.

:set

set

Modify the editing environment. Set command options are described subsequently.

Reference

- UNIX System V, Release 4 User's Reference Manual, **vi(1)**

Using Line Editor Commands

Command	Description
!:command	Escape to the UNIX shell to execute named command
:f	Display or rename current file
:r	Read in contents of named file
:e	Edit other files
:w	Save the editing buffer
:q	Quit the editor
:set	Modify the vi environment

:set all

Changing vi Environment

A number of options can be defined to modify the editing environment in vi. These options perform many functions, like displaying line numbers, ignoring case during pattern searches, or displaying invisible characters (tab, indent). Earlier in this section the command `:set showmode` was used to display input command prompts.

The vi command to enable (turn on) and to disable (turn off) these options is called `set`.

Display set Options

To view the list of all `set` command options, execute the command `:set all`. To view the current setting of a particular option, use the command `:set option?`. Or to display options that have been changed, execute `:set<RETURN>`.

Set Options

Some options are enabled by designating the option name with the `set` command, as in `:set option`. These options are disabled by preceding the options name with `no`. For example, `:set number` displays line numbers during editing; `:set nonumber` does not display line numbers during an editing session.

Other options have associated values that can be changed. For example, `:set window=5` sets the screen display size to five lines. To change the window size, assign another value to the option, like `:set window=10`.

Multiple options can be set at the same time in a single `set` command. For example, `:set number showmode` displays line numbers and input command status messages.

The `set` options can be defined several ways.

- Set options during an editing session. In this case, the options are effective only for the current editing session.
- Set options for automatic execution.
 - Include the `set` command and options in `.profile` to execute at login. The format of the command entry is `EXINIT='set option1 option2'`. `EXINIT` is a variable read by vi. Enclose the entire `set` command in quotes, for example, `EXINIT='set showmode number'`. Use the `export` command to make this information available to other processes.
 - Create a file called `.exrc` in the user home directory containing the `set` command and options. This file is executed only when vi is invoked. The format of the command entry is the same as when options are set within an editing session, like `set option1 option2`. Notice the `EXINIT` variable is not used, for example, `set showmode number`.

Changing vi Environment

- **Display set options**

- All options `:set all`
- Named option `:set window?`
- Modified options `:set<RETURN>`

- **Set options**

- Single option `:set number`
- Multiple options `:set showmode window=10`

- **Unset options**

- Single option `:set nonumber`
- Multiple options `:set noshowmode window=15`

- **Automate execution of set options**

- `.profile` entry `EXINIT='set showmode number'`
`export EXINIT`
- `.exrc` entry `set showmode number`

config for a user → *ini for vi*

vi Recovery

UNIX attempts to save the contents of the buffer when a vi session terminates abnormally.

The contents of the buffer is saved to a file in the home directory having the same name as the edited file. Use the following command to access vi with the recovery file in the buffer.

vi -r file_name

Continue to edit the file, or perform a write and quit the editor.

vi Recovery

vi -r file_name

- Invokes editor with recovery file in the buffer

sh run batch file

Practical Exercise

Invoke the editor using the *vi*file.xx to perform the following activities. Write the command used below each item.

1. Set the options to display line numbers and the input command prompts.
2. Exit to the UNIX shell to display **who** is currently using the system.
3. Replace the file in the buffer with the *numbers* file created in a previous exercise. If you are prompted to first save the buffer for *vi*file.xx, discard any changes and replace the file in the editing buffer. Go to the end and again to the beginning of the current file to confirm that *vi*file.xx is not accessible. Also, notice that the **set** options from Step 1 remain effective for the current session regardless of the file in the buffer.
4. Read in the contents of *vi*file.xx below line 12. Read in the contents of *vi*file.ok to the beginning of the current file.
5. Verify the name of the current file in the buffer.
6. Remove the line numbers from the display.
7. Save the changes to the *numbers* file and replace the file in the buffer with *vi*file.xx.
8. Rename the current file to *vi*file.old.
9. Read in the current date and time at the end of the file. Insert your name at the beginning of the file.
10. Save the buffer to the file *vi*file.new. Exit to the shell to display **who** is currently using the system. Exit the editor discarding any changes to *vi*file.old.

ed Summary

- The purpose of a UNIX text editor is to create and to manipulate text. It does not format text.
- UNIX offers several text editors.

Line based (**ed, ex, edit**)

Screen oriented (**vi, vedit**)

Restricted access (**red, view**)

- The characteristics of UNIX editors include
 - Distinct command sets
 - A temporary work area called a *buffer* stores editing
 - Separate text input and command modes
- The line editor described in this section is called **ed**. It processes text on a line basis.
- The following steps create a file using **ed**.
 1. Invoke the editor by entering **ed file_name** and press <RETURN>.
 2. Add text by entering the letter **a**; press <RETURN>.
 3. End text input mode by entering a period on a line by itself; press <RETURN>.
 4. Save the text in the editing buffer by entering the **w** (write) command; press <RETURN>.
 5. End the editing session by entering the **q** (quit) command; press <RETURN>.
- Commands providing assistance during an editing session are
 - P** Displays a * prompt in command mode.
 - h or H** Displays a help message for the most recent error, represented by the ? symbol.

ed Summary

- The format of an **ed** command is

[address]command[argument(s)]

where:

address Specifies a particular line or range of lines in the buffer affected by the command that follows.

command Designates a single character as the editing action.

argument(s) Provides additional information required by some commands, typically a file name or another line address.

- Line addresses can be *numeric*, *symbolic*, or *both*. Symbolic addressing is useful when the exact line number is unknown.

. Current line

\$ Last line

+ Next line

- Previous line

- The *comma* indicates a range of lines.
- Commands used to display text lines include.

p Print the current or designated lines of text.

n Display the current or designated lines of text with relative line *numbers*.

l Display the current or designated lines of text *listing* invisible characters, like tabs and indents.

Display the line number of a line of text.

/pattern Display the next line containing the designated *pattern* of characters.

?pattern Display the previous line containing the designated *pattern*.

- Text input commands are:

a Append text

i Insert text

c Change text

ed Summary

- The **.** (period) command exits text input mode.
- Commands used to modify text include:
 - u** Undo the last change
 - d** Delete text
 - m** Move text
 - t** Copy (transfer) text
 - s/old/new** Substitute *old text* with *new text*
- The **!**command designation allows temporary exit to the UNIX shell from the editor to perform one or more UNIX commands.
- The **w** (write) command saves the contents of the editing buffer.
- The **q** (quit) command exits the editor.
- The *ed.hup* file in the user's current directory contains the contents of the editing buffer when an editing session terminates abnormally.

Optional ed Exercise

Select at least one subsection from the `ed` tutorial in Section 6 of the User's Guide to review. Complete the exercise at the end of the selected section and check your progress using the answers provided.

Reference

- *UNIX System V, Release 4 User's Guide*, Section 6, Screen Editor `ed` Tutorial

vi Summary

- **vi** is an interactive screen editor displaying a window (screen) of text at a time.
- **vi** is dependent on the capabilities of the specific terminal. The terminal type must be designated using the following entry, either on one command line or separately.

```
TERM=uvvt1224 ; export TERM
```

- The terminal can be automatically designated for each login by entering the **TERM** assignment and the **export** command in the *.profile*.
- The steps to create a file in **vi** are:
 1. Invoke the editor by entering **vi file_name**.
 2. Add text by entering an input command, like **a** (append); do not press <RETURN>.
 3. End the input mode by pressing the <ESCAPE> key.
 4. Save the contents of the buffer and exit the editor.
- The cursor can be moved on the screen several ways. These keys are described on pages 4-40 through 4-43 of this module.
- Text not in the current window can be scrolled into view using the keys described on pages 4-44 and 4-45.
- Search commands provide a quick way to move the cursor to a particular location in a file containing the character pattern specified.

/pattern	Search forward for designated pattern of characters. Repeat search for same pattern by entering / .
?pattern	Search backward for designated pattern of characters. Repeat backward search for same pattern by entering ? .
n	Repeat the previous search in the same direction
N	Repeat the previous search in the reverse direction

- Commands providing assistance during an editing sessions are:

:set showmode	Displays status message for the current text input mode in the lower right area of the screen.
u	As with the ed command, undo restores the last change in the editing buffer.
U	Restores all changes on the current line containing the cursor.
.	Repeats the last editing change.

vi Summary

- The input commands available in vi are:

a	Append (adds) text after the cursor
A	Append text after the line
i	Insert text before the cursor
I	Insert text before the line
o	Open a line below the current line and allows text input
O	Open a line above the current line and allows text input
r	Replace a single character at the cursor
R	Replace text continuously until <ESCAPE>

- The primary editing functions used to modify text are:

<i>Delete</i>	Delete one or more characters or a text range
<i>Move</i>	Two-command procedure to delete designated text into the buffer and put it at the cursor position
<i>Copy</i>	Two-command procedure to copy designated text into the buffer and put it at the cursor position
<i>Substitute</i>	Replace one or more characters with text input.
<i>Change</i>	Replace designated text segment with text input.

- Line editor commands can be used in vi by preceding the command with a colon.

!command	Allows temporary access of the UNIX shell to perform one or more shell commands.
:f	Display the current file name, the current line number containing the cursor, and the percent of the file at the cursor location.
:r	Read in the content of the current or named file below the current line.
:e	Replace the current file in the buffer with the named file.
:w	Write the changes in the buffer to the current file.
:q	Exit the editor.
:set	Modify the editing environment.

vi Summary

- The **vi** editing environment can be modified using **set** command options. These options can be set during an editing session using the format **:set option1 option2**. Options set during an editing session are effective for the current editing session only. To set options for automatic execution at login, enter **EXINIT='set option1 option2' ; export EXINIT** in the *.profile*. To set options for automatic execution when the editor is invoked, enter **set option1 option2** in a user-created file called *.exrc*.

Display **set** options during an editing session:

:set all Displays all options and current settings

:set option? Displays status of named option

:set<RETURN> Displays options that have been changed

- If an editing session terminates abnormally, the contents of the buffer can be recovered by invoking **vi** with the **-r** option and naming the file that was in the editing buffer.

Optional vi Exercise

Select at least one subsection to review from the vi tutorial in Section 7 of the User's Guide. Complete the exercise at the end of the selected section and check your progress using the answers provided.

Reference

- *UNIX System V, Release 4 User's Guide, Section 7, Screen Editor vi Tutorial*

5

Directory and File Management

Module Objectives

Upon completion of this module, you should be able to manage UNIX directories and files.

The supporting module objectives to manage directories include the ability to

1. Display the current directory.
2. List the current directory.
3. Create new directories.
4. Change the current directory.
5. Remove directories.

The supporting module objectives to manage files include the ability to

1. Classify file types.
2. Display file contents.
3. Copy files.
4. Move and rename files.
5. Link files.
6. Print files.
7. Locate files.
8. Locate patterns in files.
9. Sort file contents.
10. Remove files.

Reference

Documentation referenced in this module

- *UNIX System V, Release 4 System User's Reference Manual* (4357 7444-000)
- *Advanced UNIX (System V, Release 4) Usage Workshop* (UE 7417)

UNIX Directories and Files

UNIX offers several hundred utilities to perform a variety of user tasks. This module is comprised of two sections providing descriptions of the most fundamental commands to manage directories and files. Section 1 introduces commands that enable you to organize and use a directory structure. Section 2 contains commands that access and manipulate files in the file system structure.

UNIX Directories and Files

- **Section 1 – Managing Directories**

- Display the current directory
- List the current directory
- Create new directories
- Change the current directory
- Remove directories

- **Section 2 – Managing Files**

- Classify file types
- Display file contents
- Copy files
- Move and rename files
- Link files
- Print files
- Locate files
- Locate patterns in files
- Sort file contents
- Remove files

Section I: Directory Management

Displaying the Working Directory

pwd – Display the working directory name

Description

This command displays the path name of the working (current) directory. **pwd** does not use any options or arguments. This command is useful to confirm the current location in the file system before or after using other commands. For example, it is often helpful to verify the current directory before creating new files or directories. Similarly, **pwd** is useful to confirm the working directory after changing the directory location.

Options

None

Examples

The example at the right shows that the current directory is */home/user1*.

Reference

- *UNIX System V, Release 4 User's Reference Manual, pwd(1)*

Displaying the Working Directory



pwd

Example

```
$ pwd  
/home/user1
```

cd

Creating Directories

mkdir - Create one or more directories

Description

Directories are useful to group related files together. Therefore, it is advisable to develop a logical naming scheme that will allow quick access and retrieval of the desired files.

The **mkdir** (make directory) command is used to create one or more directories. Enter the command followed by one or more names.

The same file naming conventions are used to name directories. Again, the maximum length of the name depends on the file system type: 256 characters for a *ufs* file system type, or 14 characters for an *s5* file system type.

The default access permission mode defined for new directories allows files in the directory to be searched and listed (read and execute access) or modified (write access) by any system user. However, the owner (creator) may designate a different directory access mode using the **-m** option. Refer to Module 6 for information pertaining to file and directory access permissions.

Options

- m** Specify the permission access mode
- p** Create named parent directories

Examples

Example 1 illustrates the basic use of **mkdir** to create the directory *subdir1* in the current directory. In the second example, three directories (*memo*, *letter*, and *report*) are created by one command. Example 3 uses the **-m** option to designate a different access permission for the new directory *mydir*. The last example uses the **-p** option to create the *monthly* directory and its parent directory, *report*, in one command. If the options **-m** and **-p** are used together, the new parent directory inherits the same access mode designated by the **-m** option.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **mkdir(1)**

Creating Directories

```
mkdir [-options] dirname(s)
```

Example 1

```
$ mkdir subdir1
```

Example 2

```
$ mkdir memo letter report
```

Example 3

```
$ mkdir -m 700 mydir
```

Example 4

```
$ mkdir -p report/monthly
```

Listing Directory Contents

ls - Display the contents of a directory

Description

The **ls** (list) command lists the contents of the named directory. Without arguments, **ls** lists the contents of the current directory. For multiple screen output, press <Hold Screen> or <^s> and <^q> to stop and resume screen scrolling.

The **ls** has many options that change the output of the display. The more frequently used options are listed below. The default display lists the directory contents in multiple columns with entries sorted down the columns.

The **-l** option provides descriptive information about each file and directory listed.

Options

- a** List all entries, including entries beginning with . (period)
- i** Display the inode number for each entry
- d** Display the directory name, not its contents
- l** Display long list with descriptive information
- r** Reverse order of list
- t** Sort by modification time
- x** Display multi-column output sorted across columns
- F** Display slash (/) after directories, asterisk (*) after executable files, or ampersand (&) after symbolic link files
- R** Display recursive listing of subdirectories

Examples

In the first example, the **ls** command is used without options or arguments to list the files and directories in the current directory. To list subdirectories and files, use the **-R** option. In Example 2, **ls** lists the contents of the *report* directory. The last example illustrates the use of multiple options. The option **-a** displays files beginning with a period. The single . refers to the current directory and .. refers to the parent directory. The option **-i** lists the inode number associated with each file or directory. An *inode* contains the control information about a file or directory displayed using the **-l** option, including the file type, permission access, number of links, owner name, group name, length of the file in bytes, date and time of last modification. Notice the file type (*l*) and name designation (*stats->report/statsfile*) for a symbolic link file.

Reference

- *UNIX System V, Release 4 User's Reference Manual, ls(1)*

Listing Directory Contents

ls [-options] [directories or files]

Example 1

```
$ ls
file1
letter
memo
progl
report
stats
```

Example 2

```
$ ls report
annual
monthly
statsfile
```

Example 3

```
$ ls -ld report
drwxr-xr-x  2  user1  admin  512  Jul 9  11:02  report
```

Example 4

```
$ ls -ial
total 14
534 drwxr-xr-x  2  user1  admin  512  May  13  08:34  .
231 drwxr-xr-x  2  user1  admin  512  May   7  10:30  ..
389 -rw-r--r--  1  user1  admin  952  May  13  16:45  .profile
894 -rw-r--r--  1  user1  admin 3457  Aug  23  14:17  file1
456 drwxr-xr-x  2  user1  admin  512  Jun   4  15:04  letter
732 drwxr-xr-x  2  user1  admin  512  Jul   9  11:02  memo
497 -rwxr-xr-x  1  user1  admin 1672  Jul   7  13:23  progl
648 drwxr-xr-x  2  user1  admin  512  Jul   9  11:02  report
832 lrwxr-xr-x  2  user1  admin  12   Jul  16  09:52  stats->report/statsf
```

Inode	Access	Links	Owner	Group	Size	Modification	Date/Time	File
-------	--------	-------	-------	-------	------	--------------	-----------	------

Changing Directories

cd – Change working directory

Description

The **cd** command is used to change the working (current) directory. There are no options for this command. The new directory name is designated as the argument to the **cd** command. Relative or absolute path names may be used as the directory name designation. **cd** without an argument returns to the home (login) directory of the user. The **pwd** command is useful to confirm the new current directory.

Options

None

Examples

In Example 1, the **pwd** command shows the current directory is */home/user1*. The **cd** command is used to change directories to the subdirectory *report*. The output of **pwd** displays the full path name of the current directory, */home/user1/report*.

The second example uses a relative path name, *../letter*, to designate the directory name. Recall from Module 3, that the *..* is a shorthand notation referring to the parent directory and that it is used to ascend the file hierarchy. The **pwd** command confirms the current directory is */home/user1/letter*.

Example 3 is very useful. Regardless of the current directory in the file hierarchy, **cd** without arguments always returns to the home (login) directory of the user.

Reference

- *UNIX System V, Release 4 User's Reference Manual, cd(1)*

Changing Directories

```
cd [directory]
```

Example 1

```
$ pwd  
/home/user1  
$ cd report  
$ pwd  
/home/user1/report
```

CD go to signon direct

Example 2

```
$ cd ../letter  
$ pwd  
/home/user1/letter
```

Example 3

```
$ cd  
$ pwd  
/home/user1
```

Removing Directories

rmdir - Remove one or more directories

Description

The **rmdir** command is used to remove one or more empty directories. The directory to be removed cannot be the current directory, and it must be empty. Only the owner or superuser can remove a directory, unless the parent directory has write access by other users.

Relative or absolute path names may be used as the directory name designation.

Options

- p Remove named directory and parent directory;
prints message about path name being removed
- s Suppress message display from -p option

Examples

In the first example, the empty *memo* directory is removed; **ls** is used to verify that *memo* no longer exists. In Example 2, notice the message displayed when an attempt is made to remove a directory that is not empty. Example 3 shows relative path names to remove two subdirectories, *report/annual* and *report/monthly*. Alternatively, the **cd** could be used to change to the *report* directory; then, only the subdirectories *annual* and *monthly* need to be designated with the **rmdir** command. In the last example, the -p option is used to remove the *sanders* subdirectory and its parent, *oct91*. Notice the confirmation message that is displayed automatically.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **rmdir(1)**

Removing Directories

```
rmdir [-options] [directories]
```

Example 1

```
$ rmdir memo  
$ ls  
file1  
letter  
report
```

Example 2

```
$ rmdir report  
rmdir: report not empty
```

Example 3

```
$ rmdir report/annual report/monthly  
$ ls report  
statsfile
```

Example 4

```
$ rmdir -p oct91/sanders  
rmdir: oct91/sanders: Whole path removed.
```

Section II: File Management

Displaying File Types

file - Display file type classification

Description

In addition to the regular files created by users, some files contain executable code or other data not suitable for displaying on the terminal; the terminal can even "hang" as a result. The **file** command is useful to determine the file type of unfamiliar files before attempting to display them.

file determines the file type of the named file(s) supplied as arguments to the command. Any file can be included in the file list, even files containing characters from supplementary (international) code sets.

The **file** command performs a number of tests to classify the files listed as a command argument. The first part of each file is examined, looking for keywords and special numbers, called magic numbers, identifying the file type. *Symbolic links*, files containing the path name to another file, and the files they reference are also tested if they are included in the named file list, unless the **-h** option is used. **-h** does not check the file type of the symbolic link, and it displays the message *Symbolic link to pathname*.

Primary classification categories are:

<i>data</i>	Data files (unreadable)
<i>commands text</i>	Shell scripts (readable)
<i>xxx text</i>	Language program text files (readable)
<i>executable</i>	Executable files (unreadable)

Other file classifications include:

<i>pure not stripped</i>	Compiled code (unreadable)
<i>directory</i>	Directory file type
<i>empty file</i>	File is empty
<i>cannot open</i>	No such file or directory

Options

-file	Designate input file containing file names to be examined
-h	Do not check symbolic link files

Examples

The first example uses **file** to classify a named file. The file list may also use file name special characters, as in the second example. If the **-h** is not used, **file** checks the file type of the symbolic link. Notice in the output of Example 2, **-h** only reports the name of the symbolic link. Also notice the output for the nonexistent file, *nofile*.

Reference

- *UNIX System V, Release 4 User's Reference Manual, file(1)*

Displaying File Types

```
file [-options] file(s)
```

Example 1

```
$ file file1  
file1:          ascii text
```

Example 2

```
$ file -h * nofile  
file1:          ascii text  
letter:         directory  
memo:           directory  
nofile:         cannot open: No such file or directory  
progl:          commands text  
report:         directory  
stats:          symbolic link to report/statsfile
```

Displaying File Contents

cat - Display contents of named file(s), or join multiple files

Description

The **cat** command displays the contents of the named file(s). File names can be designated using absolute or relative path names. For files longer than one screen display, pause the screen using the <Hold Screen>, or <^s> and <^q>. If no file name is supplied to the command, **cat** accepts input from the terminal (keyboard).

Multiple files are concatenated (joined vertically) on output; the original files remain intact. The joined output can be redirected from the screen to a file using the output redirection symbol > followed by a file name. For example, **cat file1 file2 file3 > allfiles** joins the named files and redirects the terminal output to the file *allfiles*. Redirection is described in the Advanced UNIX Usage Workshop.

Nonprinting characters (tabs, or any ASCII control character) can be displayed using the options **-vet**. This is often helpful in locating errors in program files.

Options

- v** Display ASCII control characters
- e** Display \$ at the end of each line; must use **-v** option
- t** Display tabs as ^I; must use **-v** option

Examples

The command in the first example displays the contents of *file1* and *file2*. In the second example, *file1* and *file2* are concatenated on the terminal display. Example 3 uses the **-vet** options to display nonprinting control characters in a file.

Reference

- *UNIX System V, Release 4 User's Reference Manual, cat(1)*
- *Advanced UNIX (System V Release 4.) Usage Workshop, Module 2, Command Input/Output Redirection*

Displaying File Contents

```
cat [-options] [file(s)]
```

Example 1

```
$ cat file1
This is file 1.
$ cat file2
This is file 2.
```

Example 2

```
$ cat file1 file2
This is file 1.
This is file 2.
```

Example 3

```
$ cat testfile
This      sample      file      contains
several  tabs          and       control   characters
like     or           .
$ cat -vet testfile
This^Isample^Ifile^Icontains      $
several^Itabs^Iand^Icontrol^Icharacters  $
like ^b or ^c.$
```

NROFF

/usr/ucb/nroff vifile.ok | pr -h

Displaying File Contents

pg - Display contents of named file(s) one screen at a time

Description

more

The **pg** command pages through files longer than one screen display. It provides forward and backward search capability, as well as other available commands within the **pg** program.

pg pauses after each screen and prints **:** at the bottom of the screen. Press **<RETURN>** to view the next screen. The **-p string** changes the screen prompt to *string*. The designation (*EOF*): indicates the end of the file. When multiple files are designated as command arguments, the next file can be viewed by pressing **n**; similarly the previous file can be viewed by pressing **p**.

By default, **pg** displays some 23 lines in each screen. This can be altered using the **-n** option, where *n* is a number to set the screen display. **pg** automatically displays the contents of the file beginning with line one. This can be altered using the **+n** option, where *n* is the starting line number to display the named file(s). The **+/pattern** option is used to start the display two lines above the line containing the designated character pattern.

pg also provides a number of commands that can be used within the program. To view the list of available commands, press **<h>**. Commands are available to edit the file (**v**), search forward for a character pattern (**/pattern**) or backward for a pattern (**?pattern**), to escape to the shell to perform one or more commands (**!command**), repeat the previous command (**.**), view the next file named in the command line (**n**), view the previous file named in the command line (**p**), save the entire input in the named file (**s file**), and more.

Options

-c	Clear (redraw) screen before displaying each file
-n	Set the screen size to <i>n</i> lines
+n	Start the display at line <i>n</i>
+/pattern	Display text containing character pattern
-p string	Change default screen prompt

Examples

In the first example, the default **:** prompt, displayed at the bottom of each screen, is changed to **MYPROMPT>**. Notice the prompt string is enclosed in quotes to offset the spaces in the string. Example 2 displays *file2* in 10-line screens beginning with line 10; the **-c** option clears the screen before the next screen. In the last example, *file3* is displayed starting with the screen containing the designated pattern two lines from the top of the screen.

Other commands may be available to display the contents of files. The **more** command also displays file contents a screen at a time; however, it has fewer program commands than **pg**. The **head** command displays the beginning of files (by default, the first ten lines); conversely, **tail** displays the end of files (by default, the last 10 lines).

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **pg(1)**, **more(1)**, **head(1)**, **tail(1)**

Displaying File Contents

pg [*+/-options*] [*file(s)*]

Example 1

```
$ pg -p "MYPROMPT> " file1 file2 file3
```

Example 2

```
$ pg -c10 +10 file2
```

Example 3

```
$ pg +/pattern file3
```

Copying Files

cp - Copy a source file to target file

Description

The **cp** (copy) command duplicates files or directories. The source and target names cannot be the same in the same directory. If the target is an existing directory in the same file system, multiple source files can be copied to it. However, if the target directory does not exist, **cp** creates a file by that name. If the target is an existing file, **cp** overwrites its contents. The **-i** (interactive) option is useful to avoid overwriting existing files. **cp** prompts for confirmation if the target file exists.

By default, the duplicate file retains the permissions of the source file. The **-p** (preserve) option preserves the source file's modification time; otherwise, the modification date/time of the file copy is set to the time the copy is made.

The **-r** (recursive) option is useful to copy a source directory and its files and subdirectories to the target directory within the same file system.

The duplicate source file is assigned a different inode number because it occupies a different storage area on disk from the original.

Options

- i** Interactive; prompt before overwriting existing files
- p** Preserve source file's modification date/time
- r** Recursively copy source directory (its files and subdirectories)

Examples

Example 1 uses the **-i** option to avoid overwriting the target file. A response other than **y** aborts the file copy. In Example 2, three files are copied to the named directory. Remember, if the target directory does not exist, a file by that name is created. In the third example, the files and subdirectories of *dir1/subdir* are copied to the *dir2* directory. The last example illustrates that the **cp** command assigns a different inode to the copy.

Reference

- *UNIX System V, Release 4 User's Reference Manual, cp(1)*

Copying Files

cp [-options] source target

Example 1

```
$ cp -i forma formb
cp: overwrite formb?
```

Example 2

```
$ cp memo1 memo2 memo3 ../backup
```

*3 separate
copies
memo1 copied to ../backup
directory
etc*

Example 3

```
$ cp -r dir1/subdir dir2
```

Example 4

```
$ cp file1 file2
```

```
$ ls -il file1 file2
```

894	-rw-r--r--	1	user1	admin	3457	May 13 16:45	file1
947	-rw-r--r--	1	user1	admin	3457	Aug 23 14:17	file2

Moving Files

mv - Move or rename files

Description

The **mv** (move) command moves or renames files within the same file system. The permissions and modification time of the source file are retained.

The source and target names cannot be the same in the same directory. If the target is an existing directory in the same file system, multiple source files can be moved to it. However, if the target directory does not exist, **mv** creates a file by that name. If the target is an existing file, **mv** overwrites its contents. The **-i** (interactive) option is useful to avoid overwriting existing files. **mv** prompts for confirmation if the target file exists. The **-f** option forces the move without prompting, even if the target file will be overwritten. This is the default; it overrides the **-i** option.

The new source retains its original inode number because it occupies the same storage area on disk as the original.

Options

- i** Interactive, prompt before overwriting existing file
- f** Force moving file without prompting (default)

Examples

Example 1 uses the **-i** option to avoid overwriting the target file. A response other than **y** aborts renaming the file. In Example 2, the *memo1* file is moved to the named directory. If the target directory does not exist, a file by the designated name is created. The last example illustrates that the **mv** command does not change the inode number of the source (original) file. Notice *file1* is no longer recognized; it can only be accessed by its new name, *file2*.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **mv(1)**

Moving Files

`mv [-options] source target`

Example 1

```
$ mv -i forma formb
mv: overwrite formb?
```

rename

Example 2

```
$ mv mem01 ../backup
```

Example 3

```
$ ls -il file1
894 -rw-r--r-- 1 user1 admin 3457 May 13 16:45 file1
$ mv file1 file2
$ ls -il file1 file2
894 -rw-r--r-- 1 user1 admin 3457 Aug 23 14:17 file2
file1: No such file or directory
```

Owner Group World

Linking Files

ln - Create multiple links to the named target

@USE

Description

A *link* is an entry in a directory that points to a file. The operating system creates the first link to a file when it is created. Additional links to a file can be created using the **ln** command. The **ls -l** command shows the number of links to a file. The link number of a directory, however, refers to the number of subdirectories. The **ln** command is generally used to create multiple references to file in other directories. In this way, the local (linked) name can be used in path names instead of the longer source name. A link does not create another copy of a file, merely another pointer to the same data. The file can be referenced by any of the link names. Any changes to a file are independent of the name used to reference the file.

The source can be one or more existing files. The last argument in the command line is assumed to be the target (link), which can be a file or a directory. If the target name is an existing file, **ln** overwrites the target file by default. The **-n** option can be used to avoid overwriting existing target files. Files can also be linked to a directory.

Unless the **-s** option is used, links cannot be created across file systems. The **-s** option creates a *symbolic link* which is a file containing the path name of the file to which it is linked. Symbolic links can span across file systems.

A link is removed using the **rm** command.

Options

- n** Do not overwrite existing target
- s** Create symbolic link

Examples

The first example illustrates creating a link of *file1* to another file, *qtr1*, in the same directory. In the second example, the *qtr1* file is linked to the *report/annual* directory. Since a different file name was not designated in the target, it will have the same name as the source, *report/annual/qtr1*. Example 3 illustrates use of the **-n** option to avoid overwriting an existing file. In Example 4, a symbolic link is created across file systems. In the **ls -il** output, notice the **l** file type designation for symbolic links and the designation in the file name column. The last example illustrates linking multiple source files (*qtr2*, *qtr3*, *qtr4*) to the *report/annual* directory.

Reference

- *UNIX System V, Release 4 User's Reference Manual, ln(1)*

Linking Files

ln [-options] source target

Example 1

```
$ ln file1 qtr1
```

Needs name →
existing file ←

Example 2

```
$ ln qtr1 report/annual
```

Example 3

```
$ ln -n fileX
```

```
ln: fileX: File exists
```

Example 4

```
$ ls -il /fs1/fileA
```

```
24130 -rw-r--r-- 1 user1 admin 952 May 13 16:45 fileA
```

```
$ ln -s /fs1/fileA /fs2
```

```
$ ls -il /fs2/fileA
```

```
24133 lrwxrwxrwx 1 user1 admin 37 Aug 23 14:17 /fs2/fileA -> /fs1/fileA
```

Example 5

```
$ ln qtr2 qtr3 qtr4 report/annual
```

Formatting Files

pr - Format and display the contents of the named file(s)

fmt - Simple text formatter

Description

The **pr** command is generally used in preparation for printing. If a file name is not designated, **pr** accepts input from the terminal (keyboard). The formatted output is displayed on the terminal.

By default, the file is separated into pages. The default format includes a page length of 66 lines, a page width of 72 characters, and 5 lines each for the header and trailer. Each page has a header with the name of the file, date, time, and page number. **pr** provides a number of options to change the default format. The more frequently used options are listed below.

The **fmt** format command is useful to change the width of the output. For example, the command **fmt -w 20 fileZ** formats *fileZ* for a 20-character width. Other options of **fmt** are more suitable for program files. Since **fmt** automatically joins short lines having less than 72 characters (columns), the **-s** option prevents shorter lines from being joined. This may be significant for program files requiring a specific structure.

pr Options

+n	Start display on page <i>n</i> (default is page 1)
-d	Display output double spaced
-h newfile	Change the file name in the header
-ln	Change the page length to <i>n</i> lines
-p	Pause between page display
-t	Suppress display of header and trailer

fmt Options

-w n	Change the width to <i>n</i> columns
-s	Do not join short lines

Examples

The first example combines the **-d** and **-t** options to double space and suppresses printing of the header and trailer for the file *longtext*. The second example replaces the default file name in the header with *Fourth Quarter Summary* and pauses for each new page. In the third example, the *memo.oct* file is formatted using the **fmt** command for a 45-column width. The last example uses the **-s** option to avoid short lines from being joined in a program file.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **pr(1)**, **fmt(1)**

Formatting Files

```
pr [-options] file(s)
fmt [-options] file(s)
```

Example 1

```
$ pr -dt longtext
```

Example 2

```
$ pr -ph "Fourth Quarter Summary" report/annual/qtr4
```

Example 3

```
$ fmt -w 45 memo.oct
```

Example 4

```
$ fmt -s script2
```

Printing Files

lp - Print named file(s)

Description

The **lp** (line printer) command places one or more file in the printer queue. If a file is not named, **lp** accepts input from the terminal (keyboard). A print request identification number is automatically displayed when the command is executed in the form *printer_name-n*, where *n* is a unique number. This identification can be used with other print commands to display print request status, or to cancel a print request. These commands are discussed later. An overview of the print service is provided in LP Print Service Section of the User's Guide. The LP print service is configured and administered by the local system administrator.

lp offers a number of options that affect the printing process. For example, the **-q** option changes the priority of a print request. The priority values range from 0 (high) to 39 (low). A priority limit can be assigned to a user by the system administrator.

Some of the more frequently used options are:

Options

- d ptr** Send print request to the named printer or class of printers
- f form** Print the request on the named form
- m** Notify by mail after printing is completed
- n number** Print number of copies
- q number** Assign priority level to print request
- t title** Print title on banner page (default is no title)
- w** Write message to requesting terminal after printing is completed

Examples

Example 1 requests five copies of the *report/statsfile* to be printed on the default system printer. Notice of completed printing is written to the requesting user's terminal. In the second example, two files are scheduled for printing on the named *laser* printer. Notification of completed printing is mailed to the requesting user.

Reference

- *UNIX System V, Release 4 User's Reference Manual, lp(1)*

Printing Files

```
lp [-options] file(s)
```

Example 1

```
$ lp -wn5 report/statsfile  
request id is lp00-54 (1 file)
```

Example 2

```
$ lp -md laser memo file1  
request id is laser-55 (2 files)
```

Displaying Printer Status

lpstat – Display status of LP print service

Description

The **lpstat** command displays the status of requests in the printer queue, as well as other information pertaining to the LP print service: the status of LP scheduler, the name of the default system printer, the device names of available printers, if the printer scheduler is accepting print requests for the available printers, if the printer is online or offline, and the print requests in the printer queue.

Without options, **lpstat** displays the user's current print requests. **lpstat** provides a number of options to display particular information about the print service. The more useful options are:

Options

-a ptr_list	Display if printer(s) named in list are accepting requests
-d	Display default printer name
-f form_list	Verify valid forms in list
-p ptr_list	Display status of printers in list
-r	Display status of print scheduler (running or not running)
-s	Display status summary of print scheduler; default printer; all printers; available forms, character sets, and print wheels
-t	Display all (total) status information
-u user(s)	Display status of print request for named users
-v ptr_list	Display device names of printers in list

Examples

The first example uses **lpstat** without arguments to list the status of print requests for the current user. Example 2 illustrates the **-u** option to obtain status information about the print requests of *user2*. The **-r** option used in Example 3 indicates that the print scheduler is running. The last example shows a complete status of the print service using the **-t** option.

Reference

- *UNIX System V, Release 4 User's Reference Manual, lpstat(1)*

Displaying Printer Status

lpstat [-options]

Example 1

```
$ lpstat
lp00-33          user1          1181    Sep 16 00:43
```

Example 2

```
$ lpstat -u user2
lp00-32          user2          144     Sep 16 00:43
```

Example 3

```
$ lpstat -r
scheduler is running
```

Example 4

```
$ lpstat -t
scheduler is running
system default destination: lp00
device for lp00: /dev/term/02
device for lp01: /dev/term/02
lp00 accepting requests since Sun Sep 15 06:08:20 EDT 1991
lp01 accepting requests since Sun Sep 15 06:38:34 EDT 1991
printer lp00 is idle. enabled since Sun Sep 15 06:46:50 EDT 1991. available.
printer lp01 is idle. enabled since Sun Sep 15 06:38:45 EDT 1991. available.
```

Cancel Printing

cancel – Cancel printing named print request

Description

The **cancel** command removes a scheduled print request from the print queue. To cancel one or more print requests from the queue, enter the print request identification(s) as an argument to the **cancel** command.

Options

None

Examples

Example 1 illustrates canceling the named print request. The print request identification can be displayed using the **lpstat** command. The second example cancels two print requests sent to printer *lp01*.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **cancel(1)**

Cancel Printing

```
cancel [id] [printer]
```

Example 1

```
$ cancel lp00-33  
request "lp00-33" cancelled
```

Example 2

```
$ cancel lp01-45 lp01-46  
request "lp01-45" cancelled  
request "lp01-46" cancelled
```

Deactivate Printer

disable – Deactivate named printers

Description

The **disable** command places the named printer(s) offline and stops the printer(s) from printing requests. The current print request is also stopped (-c option). The -w option directs the print service to wait until the current request being printed is finished before disabling the printer. The **lpstat** command displays the current status of printers. A default message is displayed by **lpstat** for disabled printers. Other messages can be displayed in the printer status using the **disable -r** command followed by the *reason*.

Printers are activated using the **enable** command, described later.

Options

- r *reason* Display reason printer is disabled in **lpstat** output
- w Wait; finish printing current request before disabling the printer

Examples

The first example disables printer *lp00*. A confirmation message is automatically displayed. In Example 2 the -r option is used to specify a reason printer *lp01* is disabled. This reason will be displayed in the output of the **lpstat** command.

Reference

- *UNIX System V, Release 4 User's Reference Manual, disable(1)*

Deactivate Printer

```
disable [-options] printer(s)
```

Example 1

```
$ disable lp00  
printer "lp00" now disabled
```

Example 2

```
$ disable -r "Clearing paper jam." lp01  
printer "lp01" now disabled  
$ lpstat -t  
scheduler is running  
system default destination: lp01  
device for lp01: /dev/term/02  
lp01 accepting requests since Mon Sep 16 09:38:34 EDT 1991  
printer lp01 disabled since Wed Sep 18 16:38:45 EDT 1991. available.  
Clearing paper jam.
```

Activate Printer

enable - Activate named printer(s)

Description

The **enable** command allows the named printers to print **lp** requests.

Options

None

Examples

The example on the next page illustrates use of the **enable** command to activate the lp00 printer.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **enable(1)**

Activate Printers

```
enable printer(s)
```

Example

```
$ enable lp00  
printer "lp00" now enabled
```

Locating Files

where

find - locate files

Description

The **find** command searches specified directories for files matching a designated expression. Command output is not generated unless an action, such as *-print*, is designated explicitly. The *directory-list* refers to one or more subdirectories from which to start the search. The *expression* refers to one or more criteria (or conditions). Any special character used in the expression must be quoted so that the shell does not interpret it and pass it to the **find** command. Consequently, special characters are generally enclosed in quotes by convention. **find** tests each file in the directory list against the criteria described by the expression and performs the designated action. For example, **find /home -user david -print** searches the */home* directory structure for files owned by user *david* and displays the matching file path names. An action (like *-print*) must be explicitly stated since the output is not automatically displayed. Multiple criteria may be designated. A *space* separating criteria is a logical **AND** operator; the file must meet all conditions specified. For example, **find /home -user david -type d -print** searches the */home* directory for *directory* (*d*) type files owned by user *david*. A *-o* separating criteria is a logical **OR** operator; the file must meet one or both (or all) conditions specified, for example, **find . \(-user david -o -user john \) -print**. Notice the backslash (**) to quote the parentheses.

Expression

Search Criteria:

-atime <i>n</i>	Files accessed <i>n</i> number of days ago
-follow	Follow symbolic links
-links <i>n</i>	Files having <i>n</i> links
-mtime <i>n</i>	Files modified <i>n</i> number of days ago
-name <i>pattern</i>	File name pattern (special characters must be quoted)
-perm <i>nnn</i>	Files having <i>nnn</i> (octal) permissions
-prune	Do not search within directory of preceding pattern
-type <i>char</i>	Files with <i>type char</i> (<i>type f</i> refers to an ordinary file, <i>type d</i> refers to a directory file)
-user <i>logname</i>	Files owned by <i>logname</i>

Action Criteria:

-print	Display file path names
-exec <i>cmd</i> '{}' \;	Execute <i>cmd</i> for matching files; a pair of braces represents each file name being evaluated; a quoted semicolon terminates the action
-ok <i>cmd</i> '{}' \;	Similar to <i>-exec</i> , but prompts separately for each file

Examples

In Example 1, **find** searches the current directory and displays all file path names ending with the suffix *.sh*. In Example 2, **find** searches files accessed more than 30 days ago (+30) and less than 60 days ago (-60) and removes them interactively. In Example 3, **find** searches the */home* directory and displays files modified more than 10 days ago (+10) or files larger than 10,000 characters in length. In the last example, **find** searches the directory structure for *user1* and *user2* and displays files ending in *.rpt* and having a link count of three. Notice the use of quotes around the special characters * (Examples 1 and 4), {} (Example 2) and \ (Example 3).

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **find(1)**

Locating Files

find *directory-list expression*

(*search-criteria action-criteria*)

Example 1

```
$ find . -name '*.sh' -print
```

Example 2

```
$ find . -atime +30 -atime -60 -ok rm '{}' \;
```

accessed *remove all these files*

Example 3

```
$ find /home \( -mtime +10 -o -size +10000c \) -print
```

more than 10 days ago OR 16000 characters

Example 4

```
$ find /home/user1 /home/user2 -name '*.rpt' -links 3 -print
```

2 directories *3 links*

3 or more
3 or less

find . -name passwd -print

↑ ↑ ↑ ↑

current dir *by file name* *file name* *write it on the screen*

find /usr -name lp -print*

find . -mtime -1 -print *find files modified today*

find /etc -name 'group'

Searching In Files

grep - Search files for a pattern

Description

The **grep** command searches one or more files, line by line, for a pattern. The action specified by options is executed on each line containing a matching pattern.

Patterns containing special characters must be quoted to avoid interpretation by the shell. Special characters frequently used by **grep** are:

.	Any single character position
[]	Single position in a range or set
*	Zero or more occurrences of the preceding character
^	Pattern must be at the beginning of a line
\$	Pattern must be at the end of a line

If more than one file is designated as an argument to the command, **grep** precedes each line of output containing the pattern with the name of the file and a colon. The file name is displayed for each occurrence of the pattern in a given file. The **-l** option is useful to display a file name containing multiple occurrences of the pattern only once.

Options

-i	Ignore case (do not distinguish between uppercase and lowercase)
-n	Display line numbers with the output of lines containing the pattern
-l	Display file name only once (for files containing multiple occurrences of the pattern)
-v	Reverse; display all lines not containing the pattern

Examples

In the first example, **grep** searches *fileA* for the *pattern*. Although quotes are not required, they are generally used by convention (to avoid forgetting to use them when really needed). Notice that all occurrences of the pattern are displayed regardless of line position. However, only patterns having the same case are displayed. Example 2 uses the **-i** option to display occurrences of the pattern regardless of its case. Example 3 uses a similar command format, but searches all (*) files in the current directory for the designated pattern. The last example searches all files in the current directory containing the pattern at the beginning (^) of lines.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **grep(1)**

Searching in Files

```
grep [-options] 'pattern' file(s)
```

Input File:

```
$ cat fileA
aaapatternaaa
bbbPatternbbb
patterncccccc
$ cat fileB
AAAAAA
BBBBBB
pattern
CCCCCC
```

Example 1

```
$ grep 'pattern' fileA
aaapatternaaa
patterncccccc
```

Example 2

```
$ grep -i 'pattern' fileA
aaapatternaaa
bbbPatternbbb
patterncccccc
```

Example 3

```
$ grep -i 'pattern' *
fileA:aaapatternaaa
fileA:bbbPatternbbb
fileA:patterncccccc
fileB:pattern
```

Example 4

```
$ grep '^pattern' *
fileA:patterncccccc
fileB:pattern
```

← must start at the beginning of each line.

Sorting Data in Files

sort - sort file content

Description

The **sort** command sorts the lines of the named text files. If a file name is not designated at the command line, **sort** takes its input from standard input, the terminal. The output is directed to standard output, again the terminal, unless otherwise designated using the **-o** option naming an output file. For multiple files, **sort** merges the files before sorting.

sort orders the file contents by their associated ASCII (American Standard Code for Information Interchange) values in the following sequence.

- Special characters (period, space, return)
- Numbers
- Uppercase characters
- Lowercase characters

In this order, numbers are sorted by their ASCII value, so that the numbers 10, 30, 200, 500, and 4000 are sorted in the sequence 10, 200, 30, 4000, and 500. Use the **-n** option to sort numbers by arithmetic value.

By default, **sort** orders the lines on the first field to the end of line. A *field* is a series of characters separated by a space or tab. **sort** also allows designation of alternative fields on which to sort, as well as other characters used to separate field entries.

Options

- | | |
|----------------|--|
| -f | Ignore case (do not distinguish between uppercase and lowercase) |
| -n | Numeric sort |
| -o file | Saves sorted output to named file |
| -r | Reverse the order of sort |
| -tchar | New field separator designated by <i>char</i> |

Examples

Review the contents and sequence of the input file *test*. In Example 1, the input file is sorted without any options. Notice in particular the sequence of numbers, and uppercase and lowercase entries. In Example 2, the **-f** option is used to ignore case sensitivity. This would be significant in files containing names, so that the names like MacLeod and Mack are sequenced correctly. The last example uses the **-n** option to sort numbers by arithmetic values. The **-o** option followed by a file name saves the sorted output to the *test.srt* file.

Reference

- *UNIX System V, Release 4 User's Reference Manual, sort(1)*

Sorting Data in Files

```
sort [-options] [fields] [file(s)]
```

Input File:

```
$ cat test
package
56
box
Banana
234
Pear
.profile
```

Example 1

```
$ sort test
.profile
234
56
Banana
Pear
box
package
```

Example 2

```
$ sort -f test
.profile
234
56
Banana
box
Pear
package
```

Example 3

```
$ sort -n -o test.srt test
$ cat test.srt
.profile
56
234
Banana
box
Pear
package
```

Sorting Data in Files

In the following description, a *field* is a series of characters on a line bounded by blanks (space or tab by default) and the beginning and end of line indicators. Fields used to sort are designated at the command line relative to the first field. The field designation $+n$ directs **sort** to compare lines starting at field n from the first field to the end of each line. For example, $+3$ directs **sort** to start the comparison three fields from the first field, or field four, and to continue the comparison to the end of each line. The field designation $-n$ directs **sort** to stop the comparison n fields from the first field. Without a stop field designation, **sort** continues the comparison to the end of each line. For example, -4 directs **sort** to stop the comparison before the fourth field relative to the first field, or field five. The designation **sort $+3 -4$ file** sorts each line of the input file on the fourth field only.

The sort fields can be further restricted to a range of specific characters. The format is similar to designated sort fields. Append $.n$ to the start and/or stop sort field pointer. For example, $+2.3$ sorts each line of the input files starting at the fourth character in the third field.

Since the default field separator is a space or tab, be sure to designate alternative field separators with the **-t** option to ensure sorting on the desired field.

Multiple start and stop field pointers can be designated to designate an alternative sort field for lines containing duplicate fields.

Options

+n	Skip n fields (from the first field) to start sort
-n	Stop before n fields (from the first field)
-tchar	New field separator designated by <i>char</i>

Examples

The first example sorts the input file on the second field only. Example 2 sorts the input file on the third field and the fifth field for lines containing duplicate values in the third field. The last example designates the colon as the new field delimiter (**t:**) sorts the input file starting with the fifth character of the third field to the end of each line.

Reference

- *UNIX System V, Release 4 User's Reference Manual, sort(1)*

Sorting Data in Files

```
sort [-options] [fields] [file(s)]  
  
      (field.char)
```

Example 1

```
$ sort +1 -2 test
```

Example 2

```
$ sort +2 -3 +4 -5 test
```

Example 3

```
$ sort -t: +2.4 test
```

Removing Files

rm - Remove links to named file(s)

Description

The **rm** command removes the links to a file. When the last link is removed, the file is no longer accessible and the system releases the space occupied by the file for other use. If the file is a symbolic link, the file link is removed; the original file (reference) remains.

Write permission to the file's parent directory is required to remove a file. However, read or write access to the file is not required. In this case, **rm** displays the file's permissions and requests confirmation to remove the file. The file is removed if the response is y (for yes); otherwise the file remains. When the **-f** option is used, **rm** does not request confirmation to remove files without write permission.

To remove multiple files interactively, use the **-i** option. This option requests confirmation to remove each named file. Special characters can be used to reference multiple files without designating each name separately.

The **-r** (recursive) option removes the contents of the named directory, its files, subdirectories, and the directory itself. Confirmation is requested for any write-protected files, unless the **-f** is also used in the same command line. Use this option cautiously. It is advisable to use this option with **-i** to avoid inadvertently removing needed files.

Options

- f** Force removal of write-protected file without confirmation
- i** Remove files interactively; display each file name and prompt to confirm removal
- r** Recursively remove named directory and its files and subdirectories

Examples

In Example 1, all files beginning with *file* and the file *memo* are removed from the current directory. Write-protected files are displayed, requesting confirmation to remove them. The **-f** option removes *employees* regardless of the file's permission. Directory write permission is required, however. In the description of the **rmdir** command, it was stated that the directory to be removed must be empty. To avoid having to remove the directory's files and the directory separately, use the **-r** option to remove the files and subdirectories, as well as the directory itself. Use the **-i** to receive individual prompts requesting confirmation before each file is removed. This is illustrated in the last example.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **rm**(1)

Removing Files

```
rm [-options] file(s)
```

Example 1

```
$ rm file* memo
```

Example 2

```
$ rm -f employees
```

Example 3

```
$ rm -ir report/annual
```

Summary

The UNIX commands described in Section I to manage directories are summarized below.

- **pwd** displays the working directory name
- **mkdir** creates one or more directories
- **ls** displays the contents of a directory
- **cd** changes the working directory
- **rmdir** removes one or more directories

The UNIX commands described in Section II to manage files are summarized below.

- **file** displays file type classification
- **cat** displays the contents of named file(s) or joins multiple files
- **pg** is another command that displays the contents of named files one screen at a time
- **cp** copies a source file to the target file or directory
- **mv** renames the source file or moves source files to a target directory
- **ln** creates multiple references for a file in the same or another directory
- **pr** formats file(s). **fmt** is another formatting command
- **lp** prints the named files on the system or designated printer
- **lpstat** displays status information about the lp print service
- **cancel** removes print requests from the scheduling queue
- **disable** deactivates the named printer(s)
- **enable** activates the named printer(s)
- **find** locates file (names) matching designated conditions
- **grep** searches files containing designated patterns
- **sort** orders file contents according to designated conditions
- **rm** removes links to named files

• \wedge \wedge \wedge kshfile to reset environment
 history - display history
 ~ 117 replays that action

Practical Exercise

Perform the following activities. Record your command line entries for each activity.

1. Create a directory called *class* in your home directory.
2. List descriptive information pertaining only to the *class* directory.
3. Make *class* the current directory.
4. Under *class*, create two directories called *subdir1* and *subdir2*.
5. List recursively the contents of the *class* directory.
6. Return to your home directory.
7. Copy the */etc/issue* file to your home directory having the name *issue.cp*.
8. Display the inode numbers of the original file and the copy from Step 7. Are the numbers the same or different? Why?
9. Link the *issue.cp* as *issue.ln* in the directory *subdir1*. Are the inode numbers the same or different for these files. Why?
10. Record the number of links pointing to the *issue.cp* file.

Practical Exercise

11. Record the inode number of the *issue.ln* file. Move the *issue.ln* file from *subdir1* to *subdir2* having the name *issue.mv*. Are the inode numbers the same or different for these two files? Why?
12. List the contents of *subdir1* and *subdir2* and record any files contained in each of these directories.
13. Print the *issue.cp* file on the local printer requesting a message to display on your terminal when printing is completed. Format this file for double spacing, changing the text in the header to *MY ISSUE*.
14. Display the name of your local printer. Display whether the scheduler is running or not. Display the device name of your printer. Check the status of the print service scheduler. Display all print scheduler status information using one command.
15. Return to your home directory. Recursively remove the *class* directory and its subdirectories and files requesting confirmation before each removal. List the contents of your home directory to verify that the *class* directory no longer exists.

6

Directory and File Access Management

Module Objectives

Upon completion of this module, you should be able to manage directory and file access.

The supporting module objectives include the ability to

1. Modify file and directory permissions.
2. Change file ownership.
3. Alter group association of users, files, and directories.

Reference

Documentation referenced in this module

- *UNIX System V, Release 4 System User's Reference Manual (4357 7444-000)*

Directory and File Access

UNIX controls access to the system through lognames and passwords. File and directory permissions are used to control who can access particular files or directories and the type of access.

When a file or directory is created, UNIX keeps track of who created it by the user's logname. This user is considered the *owner* of the file or directory.

The UNIX access permission scheme allows the owner to share files and directories with other users, to restrict access to certain users, and to limit the type of access to the owner's files and directories. A system superuser has full access to all files regardless of the owner or the access permission.

• ksh file

controls prompt etc

Directory and File Access

- **UNIX controls system access through**
 - Lognames
 - Passwords

- **UNIX directory and file permissions control**
 - Who has access
 - Type of access

- **Directory and file access can be changed by**
 - Owner
 - Superuser

Types of Users

There are three types of users that can access a file or directory.

- The *user* (owner) of the file or directory
- A member of the *group* to which the owner belongs

A group might include individuals in the same department or who are working on a project together. Users belonging to the same group are assigned the same group number (also referred to as the Group Id or GID). This number is stored in the */etc/passwd* file along with other identifying information about each user. The */etc/group* file contains control information about all groups on the system.

- All *other* valid users on the system, as defined in the */etc/passwd* file.

Types of Users

- **User (owner)**
- **Group**
- **Others**

User and Group System Files

The */etc/passwd* and */etc/group* files contain control information about system users and groups. System resource management and user administration are primary responsibilities of the system administrator.

Sample user and group entries are shown at the right with corresponding field descriptions to illustrate the type of information contained within these files.

User and Group System Files

User file – /etc/passwd

Logname	pswd	User Num	Group ID	Comment	home directory	initial program
user1	x	101	100	User 1 account	/home/user1	/sbin/sh
user2	x	202	200	User 2 account	/home/user2	/sbin/sh
user3	x	103	100	User 3 account	/home/user3	/sbin/sh
user4	x	404	200	User 4 account	/home/user4	/sbin/sh
user5	x	105	100	User 5 account	/home/user5	/sbin/sh

user1:x:101:100:User 1 account:/home/user1:/sbin/sh

where:

Logname	user1	
Password	x	(points to <code>/etc/shadow</code> file containing password data)
User Id	101	
Group Id	100	(primary group identification)
Comment	User 1 account	(descriptive area)
Home directory	/home/user1	(initial login directory)
Login shell	/sbin/sh	(initial login shell or program)

Group file – /etc/group

```
grp100::100:user1,user3,user5
grp200::200:user2,user4
grp300::300:user1,user2,user3,user4,user5
```

```
grp100::100:user1,user3,user5
```

where:

Group name	grp100
Password	(generally not used)
Group Id	100 (secondary group identification)
Members	user1,user3,user5

Types of Access Permission

This section introduces the three primary types of permission modes that designate the kind of action an ordinary user can perform with a file or directory.

- 4 Read (r) View or look at the contents of a file or directory
- 2 Write (w) Change the data in a file or the contents of a directory
- | Execute (x) Execute a program in a file or access the contents of a directory

Access permission to a directory has other considerations.

- Directory permissions can affect the ultimate disposition of a file. For example, if the directory allows write permission to all users, the files within the directory may be removed, even if the files do not allow read, write, or execute permission for the user.
- Execute permission is redefined for a directory; it allows searching and listing the contents of the directory.

Note: Other permission types are available in UNIX. Refer to the *User's Reference Manual* for detailed coverage of these permission types.

Reference

- *UNIX System V, Release 4 User's Reference Manual, chmod(1)*

Types of Access Permission

- **Read (r)**
- **Write (w)**
- **Execute (x)**

Checking Directory and File Permissions

The `ls -l` command displays file and directory attributes. A sample output is provided at the right.

The symbols in the first column indicate the file type and the access permissions. The first character in this column, usually a `d` or `-`, represents the file type. The `d` refers to a directory and the `-` refers to an ordinary data file.

The nine characters following the file type represent the access permissions for the file or directory. This designation describes the permissions in three sequences, one for each user type designated in the order: *user* (owner), *group*, and *others*. Each sequence consists of three characters designating the permission type for the respective user, also identified in a particular order: *r* (read) *w* (write) *x* (execute). The `-` (dash) indicates access denied for the permission type corresponding to that position. For example, `drwxr-xr-x` refers to a directory with read, write, and execute permissions for the owner, and only read and execute permission to members of the owner's group and to all other users on the system. The `chmod` command used to change directory and file permissions is presented subsequently.

The *default* permissions for directories (`rwrxrwxrwx`) allow read, write, and execute access for all users (owner, group members, and others). For files, the default permissions (`rw-rw-rw-`) allows read and write access for the owner, members of the owner's group, and all other users. Default permissions can be modified using the `umask` command described later in this module.

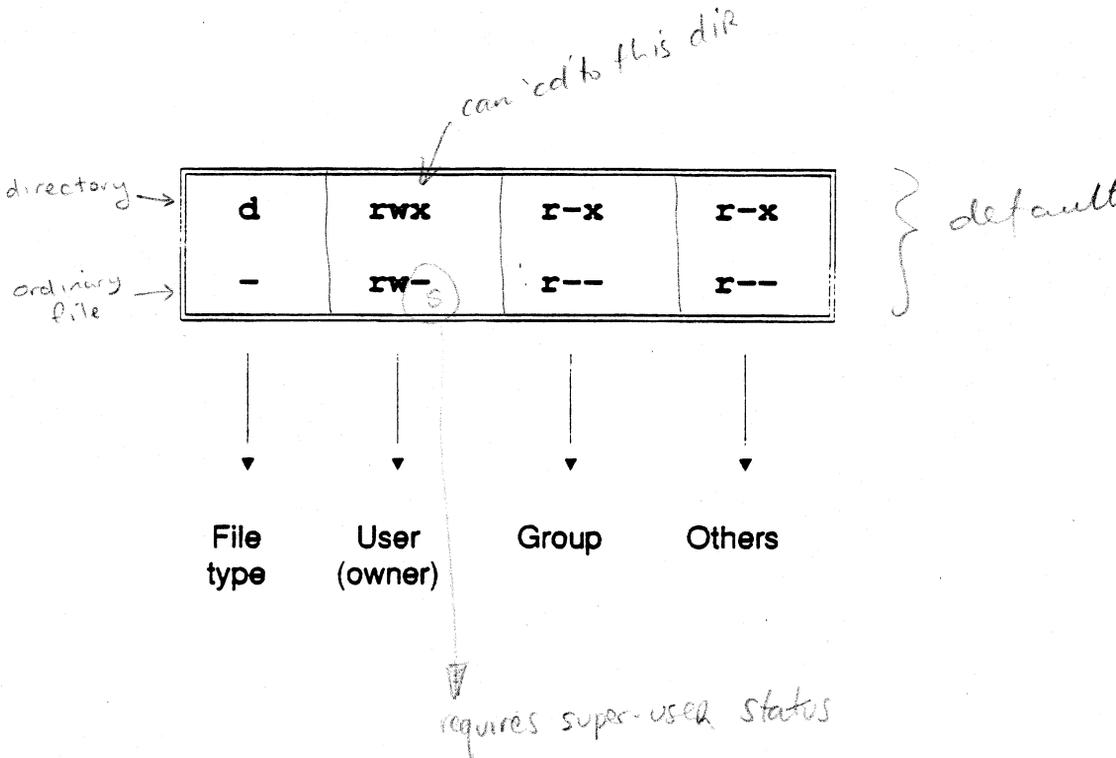
Checking Directory and File Permissions

```
$ ls -l
total 5
```

```
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
-rw-r--r--
-rw-r--r--
```

```
2 user1 grp100 512 Jun 4 15:04 Budget
2 user1 grp100 512 Jun 11 08:47 Expenses
2 user1 grp100 512 Jun 11 09:16 Salaries
1 user1 grp100 984 Jul 9 11:02 mem01
1 user1 grp100 1765 Jul 23 14:17 supplies.auc
```

Directory and File Permissions



Changing Access Permission

chmod - Assigns or changes the permission mode of a file or directory.

Description

The mode may be designated using octal or symbolic format. Only the owner or superuser may change file or directory permissions.

Options

-R Recursively descends the directory and sets the designated mode for each file. Without this option, only files and directories in the current directory are affected.

Permission Mode Formats

The permission mode in the command line is represented in one of two formats - *octal* or *symbolic*. The octal format uses numeric values to represent the permissions for each type of user. The symbolic format uses a combination of letters and symbols to represent the permissions.

These formats are illustrated on the page at the right and described separately on the following pages.

r = 4

w = 2

x = 1

Reference

- *UNIX System V, Release 4 User's Reference Manual, chmod(1)*

Changing Access Permission

```
chmod [ -option ] mode file(s)
```

- **Octal format uses numeric values**

```
$ chmod 744 memo
```

chmod 5543

- **Symbolic format uses letters and symbols**

```
$ chmod ug+x,o-r memo
```

Setuid *set* *Sticky*
groupid *groupid* *bit*
↓ ↓ ↓
S — — — | x x x | | x x x x | | x x x x |

Permission Mode Format – Octal

The page at the right shows the numeric values that may be assigned to each permission type. The sum of these values are designated for each user type. There are eight possible total values (0-7) that represent the permission mode for each type of user. They are:

- 0 No access
- 1 Execute only
- 2 Write only
- 3 Write and execute
- 4 Read only
- 5 Read and execute
- 6 Read and write
- 7 Full access (read, write, and execute)

Examples

Example 1 illustrates using the **chmod** command. The **ls -l** command displays the current permission mode for the *prog1* file. This file is a program file and requires execute permission. The **chmod** command designates full access to the file owner, and read and execute permissions for all other users. The values corresponding to the desired permission are added for each type of user. The resulting access mode is

<i>User</i>	4 (r) + 2 (w) + 1 (x) = 7			
<i>Group</i>	4 (r) + 0 (w) + 1 (x) = 5	7	5	5
		(User)	(Group)	(Others)
<i>Other</i>	4 (r) + 0 (w) + 1 (x) = 5			

Note: Optional permissions are represented by a 4-digit mode. Refer to the **chmod(1)** entry in the User's Reference Manual for further description.

In Example 2, the *Salaries* directory contains sensitive salary information. The **chmod** command is used to restrict access to its files. The owner retains all access privileges to the files in this directory, octal value 7, or *rwX*; group members may list and access the files depending on individual file permissions, octal value 5, or *r-x*; all other users are completely restricted from listing the directory contents, octal value 0, or *- - -*. In this example, although the modes for user and group did not change from the original, they are designated to avoid ambiguity. For example, **chmod 4 Salaries** yields unexpected results. The access mode is set as *004* (*-----r--*), read-only permission to others and all other permissions values to 0 (access denied). The permission mode can be changed at any time by the owner or superuser.

Permission Mode Format – Octal

- Numeric values represent permission mode

Permission Type	User Type		
	User (Owner)	Group	Others
read	4	4	4
write	2	2	2
execute	1	1	1
access denied	0	0	0
Total	7	7	7

Example 1

```
$ ls -l prog1
-rw-r--r-- 1 user1 grp100 1475 Jul 9 11:02 prog1
$ chmod 755 prog1
$ ls -l prog1
-rwxr-xr-x 1 user1 grp100 1475 Jul 9 11:02 prog1
```

Example 2

```
$ ls -ld Salaries
drwxr-xr-x 2 user1 grp100 512 Jun 11 09:16 Salaries
$ chmod 750 Salaries
$ ls -ld Salaries
drwxr-x--- 2 user1 grp100 512 Jun 11 09:16 Salaries
```

Permission Mode Format – Symbolic

The symbolic format uses letters and symbols to designate the permission mode. It consists of three elements.

- User type
- Action
- Permission type

User Type

The following letters refer to the types of user:

- u* User (owner)
- g* Group
- o* Others
- a* All (u, g, o)

Notice the additional user type, *a*, representing all users. This is the default user type if one is not designated explicitly.

Action

The action component signifies how permissions are to be changed. The following symbols designate action:

- + Add permission(s)
- Remove permission(s)
- = Assign permission explicitly

The + and - operators add and remove permissions relative to the current permission mode. The = operator resets all permissions explicitly (exactly as indicated).

Permission Type

The types of permissions were described earlier. Again, they are *r* (read), *w* (write), and *x* (execute). Notice that the - symbol, indicating access denied, is not used in this format.

Examples

In Example 1, execute permission for group and others is removed for the *Budget* directory. The user type codes are *go* (do not include spaces), the action is to remove access (-), and the permission type is execute (*x*). The command line in Example 2 changes the access modes for all users explicitly to read and execute. Notice that the owner's write permission was removed. In Example 3, the command line removes all access to group and others. If the = symbol is not preceded by a group, it defaults to all groups. If the = symbol is not followed by a permission type, it affects all permissions for the user type(s) specified. The last example illustrates how to designate multiple permissions in one command. Write permission is removed from the group and others, and execute permission is granted to all users. The comma is used to separate multiple symbolic modes. Operations are performed in the order indicated. No spaces appear in the entire mode designation.

Permission Mode Format – Symbolic

- Letters and symbols represent permission mode

user
group
other

User Type	Action	Permission Type
u	+	r
g	-	w
o	=	x
a		

Example 1

```
$ ls -ld Budget
drwxr-xr-x  2  user1 grp100   512  Jun  4 15:04 Budget
$ chmod go-x Budget
$ ls -ld Budget
drwxr--r--  2  user1 grp100   512  Jun  4 15:04 Budget
```

Example 2

```
$ ls -l prog2
-rw-----  1  user1 grp100  1986  Jun 13 08:26 prog2
$ chmod a=rx prog2
$ ls -l prog2
-r-xr-xr-x  1  user1 grp100  1986  Jun 13 08:26 prog2
```

Example 3

```
$ ls -l temp
-rwxr-xr-x  1  user1 grp100   512  Jun 20 09:33 temp
$ chmod go= temp
$ ls -l temp
-rwx-----  1  user1 grp100   512  Jun 20 09:33 temp
```

Example 4

```
$ ls -l script
-rw-rw-rw-  1  user1 grp100  2475  Jul 19 11:42 prog1
$ chmod go-w,a+x script
$ ls -l script
-rwxr-xr-x  1  user1 grp100  2475  Jul 19 11:42 prog1
```

Changing Default Permissions

umask - Changes (masks) the default permission modes for new files and directories

Description

Default access permissions are 777 (rwxrwxrwx) for directories and 666 (rw-rw-rw-) for files. These defaults can be changed, or masked, for new files and directories for the current UNIX session or automatically set at login. Typically, a permission mask is defined in the system file */etc/profile*, or it may be included in a customized *profile* in the user's home directory. The contents of these files are executed at login.

Four octal digits (*nnnn*) refer to the permission mask for an optional permission type and for each of the three user types (user, group, and others), respectively. Each value is subtracted (masked) from the corresponding default user-type permission. For example, **umask 022** does not refer to an optional permission type. The owner's permissions remain unchanged (value 0 subtracted from user's current permission); however, write permission (octal value 2) for group and others is removed. The permissions applied to new files or directories after this command is executed are listed below.

Directories

Default permission	7 7 7	(rwxrwxrwx)
umask octal value	- 0 2 2	

Current permission	<u>7 5 5</u>	(rw-rw-rw-)
--------------------	--------------	-------------

Files

Default permission	6 6 6	(rw-rw-rw-)
umask octal value	- 0 2 2	

Current permission	<u>6 4 4</u>	(rw-r--r--)
--------------------	--------------	-------------

Example

Without specifying an octal value, **umask** displays the current value of the mask. The first of the four octal digits refers to optional permission settings mentioned earlier. The **ls -l** output displays the access permissions for an existing file and directory.

The permission mask affects files and directories created *after* the change; permissions for files and directories created *before* the change remain unaltered. In the example, the permissions for *old_dir* and *old_file* remain unchanged after the permission mask (000) is set, while *new_dir* and *new_file* reflect the masked permission modes.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **umask(1)**

Changing Default Permissions

umask [nnnn]

Example

```
$ umask
```

```
0022
```

```
$ ls -l
```

```
drwxr-xr-x  2  user1  grp100      512  Jun  4 15:04  old_dir
-rw-r--r--  1  user1  grp100     2358  Jun  4 11:19  old_file
```

```
$ umask 000
```

```
$ cat > new_file
```

This file is created after
the new permission mask.

```
<^d>
```

```
$ mkdir new_dir
```

```
$ ls -l
```

```
drwxr-xr-x  2  user1  grp100      512  Jun  4 15:04  old_dir
-rw-r--r--  1  user1  grp100     2358  Jun  4 11:19  old_file
drwxrwxrwx  2  user1  grp100      512  Jun  9 13:21  new_dir
-rw-rw-rw-  1  user1  grp100     1561  Jun  9 13:23  new_file
```

Changing File Ownership

chown - Changes the owner of files and directories

Description

Changing file or directory ownership is another way to share information among users.

The *owner* can be referred to by the logname or by the user Id number from the */etc/passwd* file. The file name may be a list of files or directories. Existing permissions are not changed.

Only the owner or a superuser may change the ownership of a file or directory. Ownership cannot be changed back by the original owner; it may be changed, however, by the new owner.

Note: Ownership changes may be restricted on some systems to the superuser.

Options

- R Recursively descends the directory structure changing the ownership. Does not change the owner of symbolically linked files
- h Changes the owner of the symbolic link file

Examples

Example 1 shows the file attributes for the *prog3* file owned by *user1*. The **chown** command is used to change file ownership to *user2*. In Example 2, *user1* is denied permission to change the ownership of the file. Once ownership is changed, it cannot be "taken back" by the original owner. Notice the error message when this is attempted.

Reference

- *UNIX System V, Release 4 User's Reference Manual, chown(1)*

Changing File Ownership

```
chown [ -options ] user file(s)
```

Example 1

```
$ ls -l prog3
-rwxr-xr-- 1 user1 grp100 1765 Jul 2 13:34 prog3
$ chown user2 prog3
$ ls -l prog3
-rwxr-xr-- 1 user2 grp100 1765 Jul 2 13:34 prog3
```

Example 2

```
$ chown user1 prog3
chown: user1: Not owner
```

Group Membership and Access Permissions

A user can belong to more than one group at a time, as long as the user is listed as a member of each group in the */etc/group* file. However, the group Id number specified in the */etc/passwd* file remains the user's primary group. This group Id is associated with the files and directories created by the user. The user and group names can be displayed using the `ls -l` command.

Group membership allows access to the files and directories belonging to members of the same group. The `id` command displays user and group identification. The `groups` command displays the groups to which the user belongs.

Files and directories belonging to groups other than the user's primary group may be accessed using the following commands.

- **chgrp** command changes the group ownership of an existing file or directory.
- **newgrp** command changes the group association of a user. Files and directories created after the user changes to another group reflect the "new" (current) group ownership.

Group Membership and Access Permissions

- **User can belong to multiple groups**

- **Group membership allows access to group files and directories**

- **Commands to display user and group identification**
 - **id** displays current user and group **id**
 - **groups** displays groups to which user belongs

- **Commands to access data belonging to other groups**
 - **chgrp** changes group ownership of existing files and directories
 - **newgrp** changes group association of a user; new files and directories reflect current group ownership

Displaying User and Group Id's

id - Displays the current user and group Id numbers and names

Description

The user (UID) and group (GID) information displayed by the **id** command is taken from the */etc/passwd* file. If a group name is not displayed, the user's primary group defined in the */etc/passwd* file does not exist in the */etc/group* file.

Options

-a Reports all groups to which the user belongs

Examples

Example 1 shows the basic use of the **id** command. Example 2 uses the **-a** option to display all the groups to which the user belongs.

SU *select another user*

SU *sign on as root*

SU *userid* *✓ ✓ ✓ user*

Reference

- *UNIX System V, Release 4 User's Reference Manual, id(1)*

Displaying User and Group Id's

`id [-option]`

Example 1

```
$ id
uid=101(user1) gid=100(grp100)
```

Example 2

```
$ id -a
uid=101(user1) gid=100(grp100) groups=100(grp100),300(grp300)
```

Displaying User Group Membership

groups - Displays group membership of user(s)

Description

The **groups** command displays the group(s) to which the current or designated user(s) belong. The first output entry refers to the user's primary group as defined in the */etc/passwd* file. Other entries list group membership from the */etc/group* file. A duplicate entry may appear if the group is already defined as the primary group for the user in the */etc/passwd* file, and if the user is also listed as a member of the group in the */etc/group* file.

Examples

Example 1 shows a basic output of the **groups** command. The first of the two *grp100* entries refers to user1's primary group defined in the */etc/passwd* file and the second *grp100* entry indicates that user1 is also listed as a member of the *grp100* in the */etc/group* file. Example 2 illustrates using the **groups** command to determine the groups to which another user belongs. In Example 3, group membership for multiple users is indicated in the same command line.

Reference

- *UNIX System V, Release 4 User's Reference Manual, groups(1)*

Displaying User Group Membership

```
groups [user(s)]
```

```
$ grep 'user*' /etc/group
grp100::100:user1
grp200::200:user2
grp300::300:user3,user5
$ id
uid=101(user1) gid=100(grp100)
```

Example 1

```
$ groups
grp100 grp100
```

Example 2

```
$ groups user2
grp100 grp200
```

Example 3

```
$ groups user2 user3
user2 : grp100 grp200
user3 : grp100 grp300
```

Changing a File's Group Ownership

chgrp - Changes the group ownership of an existing file

Description

The group designation can be a group name or a group Id number. If a numeric Id is used, a file can be assigned to a group that does not yet exist in either group or password files.

The file name may be a list of files. The file's existing permissions remain unchanged.

Only the owner or superuser may use this command.

Note: Ownership changes may be restricted on some systems to the superuser.

Options

- R Recursively descends the directory structure changing the ownership. Does not change the owner of symbolically linked files
- h Changes the owner of the symbolic link file

Examples

In the example, the *memo1* file is owned by *user3* belonging to *grp100*. The group ownership of *memo1* is changed to *grp300* and verified with the `ls -l` command. In Example 2, *user3* cannot change a file's group ownership unless the user is the owner or the superuser.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **chgrp(1)**

Changing a File's Group Ownership

```
chgrp [ -options ] group file(s)
```

Example 1

```
$ id
uid=103(user3) gid=100(grp100)
$ groups
grp100 grp300
$ ls -l
total 5
drwxr-xr-x  2 user3 grp100  512 May  7 13:34 .
drwxr-xr-x  2 user3 grp100  512 May 13 10:30 ..
drwxr-xr-x  2 user3 grp100  512 Jun 11 09:16 subdirl
-rw-r--r--  1 user3 grp100  984 Jul  9 11:02 memol
-rw-r--r--  1 user1 grp100 1765 Jul 23 14:17 report5
$ chgrp grp300 memol
$ls -l memol
-rw-r--r--  1 user3 grp300  984 Jul  9 11:02 memol
```

Example 2

```
$ls -l report5
-rw-r--r--  1 user1 grp100 1765 Jul 23 14:17 report5
$ id
uid=103(user3) gid=100(grp100)
$ chgrp grp300 report5
chgrp: report5: Not owner
```

Changing User Group Membership

newgrp - Changes user's current group

Description

This command allows a user to switch to the designated group. The user must be a member of the new group, as defined in the */etc/group* file. The group designation can be the group name or the group Id.

Files and directories created after the **newgrp** command is executed reflects the "new" (current) group ownership.

Without arguments, the user's group Id is restored to the primary group in the */etc/passwd* file. If the first argument is a -, the login sequence is reinitiated logging the user in as a member of the new group.

The current shell is replaced by the **newgrp** command. The current directory remains unchanged and only exported variables retain their values.

Examples

In Example 1, the **id** command displays the user and group names and Id's for *user1*. The **groups** command indicates that *user1*'s primary group (defined in the */etc/passwd* file) is *grp100* and *user1* also has secondary group membership in *grp200* (defined in the */etc/group* file). The display (read access) of *file2* illustrates automatic group access to files belonging to secondary group members (*grp200*), without first switching to the group. However, *user1* cannot access *file3* without membership in *grp300*.

In Example 2, the **newgrp** command is used to change the current group to *grp200*. The **id** command verifies the new group association. When *file4* is created (using output redirection described in the next module), the current group (*grp200*) is associated with the new file.

Example 3 shows the error message displayed because *user1* is not a member of *grp300*. In the last example, the **newgrp** command issued without arguments is used to return to the primary group defined in the */etc/passwd* file.

Reference

- *UNIX System V, Release 4 User's Reference Manual, newgrp(1)*

Changing User Group Membership

newgrp [-] [group]

Example 1

```
$ id
uid=105(user1) gid=100(grp100)
$ groups
grp100 grp200
$ ls -l
total 5
-rw-r--r--  1 user1  grp100   2345 Jul  9 11:02 file1
-rw-r-----  1 user2  grp200   3588 Jul 23 14:17 file2
-rw-r-----  1 user3  grp300   7925 Feb 15 11:02 file3
$ cat file2
This file belongs to user2 in grp200.
$ cat file3
cat: cannot open file3
```

Example 2

```
$ newgrp grp200
$ id
uid=105(user1) gid=200(grp200)
$ cat > file4
This file is created after
user1 changed grp200.
<^d>
$ ls -al file4
-rw-r--r--  1 user1  grp200   4265 Jul 26 09:23 file4
```

Example 3

```
$ newgrp grp300
newgrp: Sorry
```

Example 4

```
$ newgrp
$ id
uid=105(user1) gid=100(grp100)
```

Summary

- Permission modes control access to UNIX files.
- File and directory access can be changed by the owner or a system superuser.
- A superuser retains full access to all files regardless of ownership or access permission.
- The */etc/passwd* and */etc/group* files contain control information pertaining to users and groups.
- The three types of users that access files are the *owner*, *group* members, and all *other* valid system users.
- The three primary types of file access are *read* (*r*), *write* (*w*), and *execute* (*x*).
- Permission settings can be viewed using the `ls -l` command.
- The 9-character permission designation describes the permissions in three sequences, one for each user type, each sequence consisting of 3 characters representing the permissions for the user type.
- Default permissions for directories are *rxwxrwx* and *rw-rw-rw-* for files.
- Access permissions can be changed using the `chmod` command.
- The permission mode can be designated in the `chmod` command line using *octal* or *symbolic* format.

Octal format expresses access permissions as digits, one for each user type. Each digit represents the total value of the permissions for the user type. A fourth digit may be designated for optional permission types.

Symbolic mode represents permissions using letters and symbols to represent the user type, the action (add, remove, or set), and the permission type.
- `umask` command changes default file and directory permissions. The octal digits, one for each user type, refer to permissions that are subtracted from the default permissions and applied to new files and directories. Existing file and directory permissions remain unaffected.

Summary

- File ownership can be changed using the **chown** command.
- Group membership allows access to data owned by group members. The group identified in */etc/passwd* designates a user's primary group. A user can belong to multiple groups. Group membership is designated in the */etc/group* file.
- Current user and group identification can be viewed using the **id** command.
- The **groups** command displays the groups to which the current or designated user belongs.
- Group ownership of an existing file or directory can be changed using **chgrp** command.
- The **newgrp** command allows a user to change to another group or back to the primary group. Files and directories created after the **newgrp** command is executed reflect the "new" (current) group ownership.

Practical Exercise

Change directory to your home directory to perform the following activities at your terminal.

1. Create two short files called *perm1* and *perm2*. The files should contain one or two lines of text each.
2. Display the permission mode for these files. Record the commands used and the permission mode for each file.
ls -l perm?
3. Change the access mode for *perm1* using octal format so that the owner can access and modify the file, group members can only view the file, and all other users are completely restricted from accessing the file. Record the command used. Display the permissions for *perm1* to confirm that the new access mode was changed correctly. *-640*
4. Change the access mode for *perm2* using symbolic format to remove all access by group members and all other users (except the owner). Record the command used. Display the permissions for *perm2* to confirm that the new access mode was changed correctly.
ok
5. Display and record the current **umask** setting.
022
6. Change the default permission to restrict all access by group members and other users for new files and directories. Record the command used.
umask 077
7. Create the file *perm3* containing one or two lines of text.
8. Display and record the permission mode for *perm3*. What effect does the current default permission have on the three files *perm1*, *perm2*, and *perm3*? Explain.
any new files use the umask
9. Change the mode for *perm1* and *perm2* to read and write for all users. Confirm this change.
chmod 777 perm1
10. Change the ownership of *perm3* to your neighbor's logname. Confirm this change. Change the ownership of *perm3* back to your logname. Record the result. Explain.
chgrp *it worked*

Practical Exercise

Your instructor will assign you to groups in order to perform the following activities.

11. View and record your user and group names and associated numeric identification.
12. Record the command used and all group memberships for your logname.
13. Change and record the group membership for *perm1* to a secondary group of which you are a member. Display the new group association for this file to confirm the change. What effect does this change have on group access to *perm1*?
14. Change your current group association to a secondary group of which you are a member. Display and record your current identification. Create another short file called *perm4* and record the group association for this file.
15. Change your group association to a group of which you are *not* a member. Record the results. Return to your primary group (defined in the */etc/passwd* file) and display your current group identification to verify this change.

It said sorry but it didn't say why

Optional Exercise

1. List the types of users who can have permission to work with a file or directory.
2. Name the types of access permissions for a file or a directory.
3. Which command displays file permissions and attributes?
4. Write the default permissions for files and directories. *755
644*
5. Choose the correct command that changes file and directory permissions.
 - a. **newgrp**
 - b. **umask**
 - c. **chown**
 - d. **chmod**
6. Interpret the following permissions in the space below each item.
 - a. **chmod 755 file1**
file1 to default permissions
 - b. **chmod a+x file1**
allow x on all users for file1
 - c. **chmod o= file1**
disallow access for other users
 - d. **chmod u-wx,go-x file1**
disable wx for user, x for the rest
 - e. **chmod a=r file1**
read only file
 - f. **chmod 700 dir1**
full access for owner, none for others
 - h. **chmod =r file1**
7. True or false. The **umask** command affects existing files and directories. If false, explain.
 - a. True
 - b. False *New ones only*
8. Select the command(s) that display user and/or group identification.
 - a. **newgrp**
 - b. **groups**
 - c. **/etc/group**
 - d. **id**

7

Basic Communications

Module Objectives

Upon completion of this module, you should be able to exchange information with other users.

The supporting module objectives include the ability to

1. Define basic communication terms.
2. Create and display system news.
3. Communicate with active terminals.
4. Use electronic mail utilities.

Reference

Documentation referenced in this module

- *UNIX System V, Release 4 System User's Reference Manual (4357 7444-000)*

Communication Overview

Communication is the exchange of information. The rapid evolution of data processing capabilities and the increased demands to exchange information over greater distances have given rise to communication systems. *Data communication* is the transmission of electronic information over distance.

A *communication system* comprises of several hardware components (computers systems, terminals and other peripheral devices, modems, and communication media) that are physically linked, and software that manages the logical connection of the hardware components. Computer systems and terminals generate, process, and receive the data. Modems are devices that convert the digital data generated by computers to a format (analog) that can be sent over communication lines. Familiar types of communication media include telephone lines, cables, microwave, and satellite transmission.

Communication Networks

Data transmission is accomplished over a *communication network*. There are several types of communication networks, depending on the geographical distance covered. Wide Area Networks (WANs) are appropriate for large distances more than 100 km (across countries). Medium Area Networks (MANs) cover distances less than 100 km, typically citywide. Local Area Networks (LANs) cover distances less than 10 km, typically limited to a building or facility. Several types of LANs exist. The differences between them are primarily speed of data transfer, cost of connection to the LAN, and the ease with which different computers can be interconnected.

A communication network consists of *nodes* and links between the nodes. A node refers to the location of hardware devices that handle (generate, process, display, or transmit) information. Therefore, nodes can be mainframes, workstations, terminals, or output devices like printers or plotters. The geometric pattern created by the connection of these devices forms the *topology* of the network. There are several standard patterns, or topologies - star, ring, undefined, and bus. Each topology offers advantages and disadvantages to the overall communication network.

Data Transmission

The speed at which data is transmitted and the amount of data, or traffic, on the line are two important considerations. These considerations depend on the speed of the lines used (low, medium, high), the mode data is transmitted (one character at a time, called *asynchronous*, or groups of characters at a time, called *synchronous*), and the line type which determines the direction (one way, or both ways) data is transmitted along the line.

The transmission of data is managed by software. The primary functions performed by the software are to: 1) establish the logical connection between nodes, 2) control the traffic along the communication lines in the network, 3) detect (and possibly correct) errors, locate points of failure, and recover, 4) provide network administration services, and 5) provide a common understanding between all nodes regarding how information is exchanged between sender and receiver.

A common set of rules must be adhered to by all nodes in the network to avoid conflict in data transmission. This set of rules is called a *protocol*. Several protocols may govern the network, each in charge of a different task. Communication protocols were initially vendor-dependent; each manufacturer having a unique set of protocols for its products. As users began to combine different products, the need to establish industry standards became increasingly apparent. As a result, several standards have been developed by national (ANSI, IEEE) and international (ISO) organizations. The *ISO organization* is responsible for the communication model of layered protocols, called *OSI (Open Systems Interconnection)*, which has become the industry standard.

Communication Overview

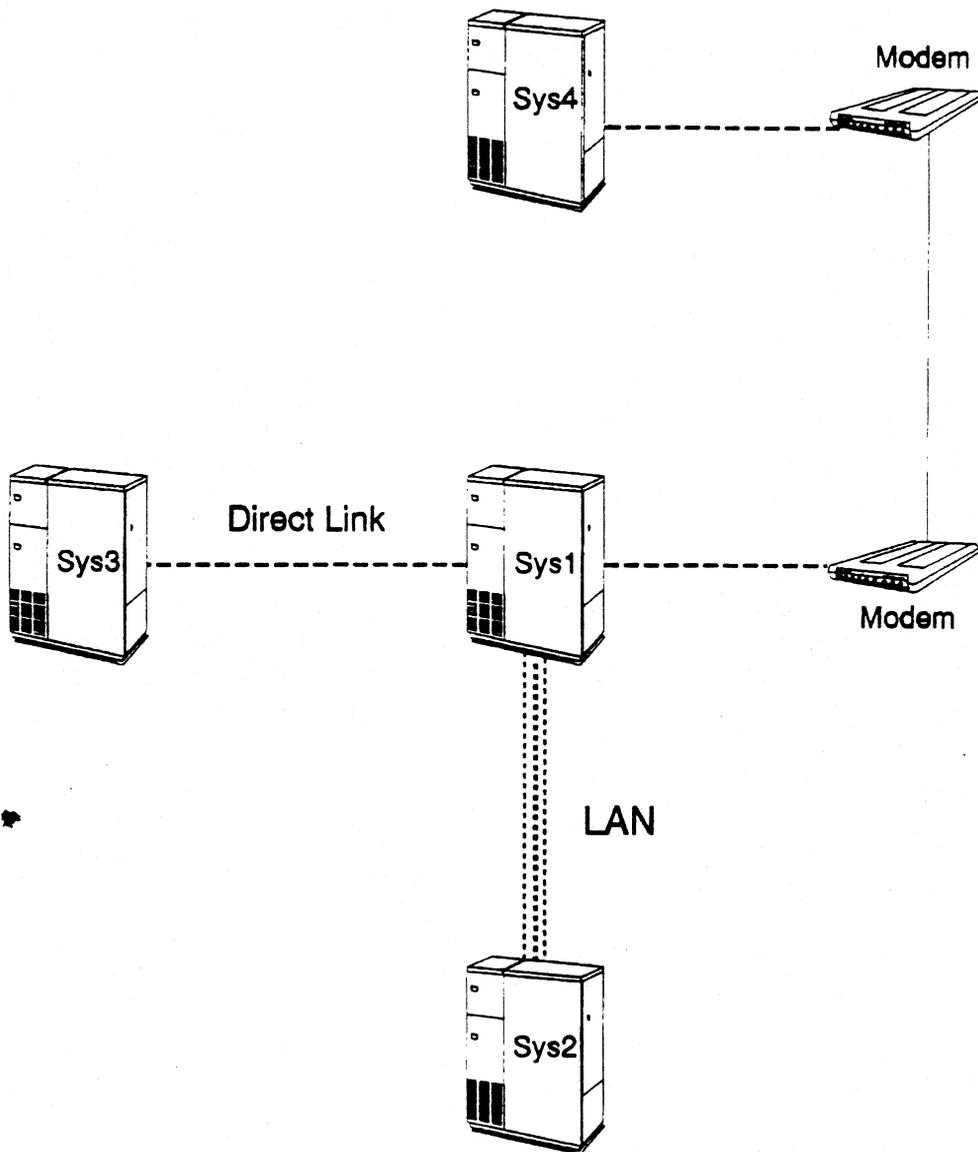
- **Communication is the exchange of information**
- **Data communication is the transmission of electronic information over distance**
- **Data communication system consists of hardware and software**
- **Communication network consists of interconnected devices forming a pattern called a topology**
- **Data transmission is managed by software**
- **Organizations developed to establish industry standard communication protocols – OSI model is current standard**

Sample Communication System

The figure on the next page illustrates a basic communication system. Each system in this configuration supports other terminals and peripheral devices.

sys1 is the node in this sample configuration. *sys3* is directly linked by cabling to *sys1*. The distance between systems is limited by the length of the cable physically connecting the systems. *sys4* is indirectly linked to and communicates with *sys1* using modem devices that convert the digital data generated by computers to a form that can pass over the communication media. Indirect links allow greater distance between systems. *sys2* is indirectly connected to *sys1* in a local area network (LAN). The logical connection of these hardware components is managed by software.

Sample Communication System



UNIX Communications

There are basically two types of communication: *local* and *remote*. UNIX offers a variety of commands to communicate with users on the local (same) system, or with remote (other) systems.

The communications features in this module include local communication utilities. Basic networking utilities are described in Appendix D.

- Local communication
 - View system announcements
 - Send messages (two-way communication with other terminals)
 - Use electronic mail

Reference

- *Basic UNIX Usage Student Guide*, Appendix D, Basic Networking Utilities

UNIX Communications

- **Local – Same system**
 - View system announcements – **news**
 - Send messages – **mesg finger write**
 - Use electronic mail – **mail mailx notify vacation**

talk

Local Communications

Displaying System News

news - Display system news

Description

The **news** feature allows systemwide posting of news items, such as notices, announcements, or bulletins. **news** is automatically executed at login from the file */etc/profile*.

News items are displayed based on *currency*. Currency refers to the modification date of the file *.news_time* in the user's home directory. Only new files more recent than the date of *.news_time* are considered current.

Without arguments, **news** displays all current files in the */var/news* directory. To display a specific news item, designate the name of the news item as an argument to the **news** command.

Options

- a Display all news items regardless of currency
- n Display only names of current news files
- s Display number of current news items

Examples

The first example lists the names of current news items using the **-n** option. In the second example, the **-s** option is used to list the number of current news items. The last example displays the contents of a particular news item by designating the item name as an argument to the **news** command.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **news(1)**

Displaying System News

```
news [-options]
```

Example 1

```
$ news -n  
item4 item3 item2 item1
```

Example 2

```
$ news -s  
4 news items.
```

Example 3

```
$ news item3  
item3 (root) Tue Oct 8 08:40:17 1991  
A staff meeting for Organization X is scheduled for next  
Friday, October 18th, at 8:30 a.m. All department members  
should plan to attend.
```

Sending Messages

mesg - Permit or deny messages sent to terminal

Description

This command permits or denies display of messages sent by another user to your terminal. Without options, **mesg** displays the current setting.

Options

- n Deny message display to terminal
- y Permit message display to terminal

Examples

The first example shows the current setting of **mesg**. In the second example, the **n** designation is used to deny message display to the user's terminal. The command in Example 3 permits message display. This is confirmed by displaying the current setting.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **news(1)**

Sending Messages

```
mesg [-option]
```

Example 1

```
$ mesg  
is y
```

Example 2

```
$ mesg -n  
is n
```

Example 3

```
$ mesg -y  
is y  
$ mesg  
is y
```

Sending Messages

finger - Display information about local and remote users.

Description

This command displays information about each user logged in to a local or remote system. The user information includes the logname, logname description (from `/etc/passwd` file), terminal name, idle time, login time, and location, if known. An `*` at the beginning of the terminal name indicates that the message capability is disabled (turned off with the `mesg` command) for the terminal.

Options

- `-f` Suppress column heading display
- `-q` Display only logname, terminal and login time
- `-l` Display long output
- `-b` Suppress display of home directory and shell

Examples

The first example shows the standard command output. Notice the `*` for the console and `term/15` indicating message capability is disabled for the terminals. In the second example, the `-f` output does not display descriptive column headings. The `-q` option in example 3 shows the quick (three-column output). The last example shows a sample output in long format for one user using the `-l` option. A similar display appears for each active user. Variable information (like the status of user's mail) may also be displayed.

Hy shows terminal file

Reference

- *UNIX System V, Release 4 User's Reference Manual, finger(1)*

Sending Messages

finger [-options]

Example 1

\$ finger

Login	Name	TTY	Idle	When	Where
wjnr	Bill Radogna00)	*console	1:17	Thu 08:10	
root	0000-Admin(0000)	*term/15	1	Thu 08:11	
root	0000-Admin(0000)	term/10	20	Fri 10:19	italy
userb	bourne user	term/12	1:01	Fri 10:25	

Example 2

\$ finger -f

wjnr	Bill Radogna00)	*console	1:17	Thu 08:10	
root	0000-Admin(0000)	*term/15	1	Thu 08:11	
root	0000-Admin(0000)	term/10	20	Fri 10:19	italy
userb	bourne shell user	term/12	1:01	Fri 10:25	

Example 3

\$ finger -q

Login	TTY	When
wjnr	*console	08:10
root	*term/15	08:11
root	term/10	10:19
userb	term/12	10:25

Example 4

\$ finger -l

Login name:	userb	(messages off)	In real life:	bourne user
Directory:	/home/userb		Shell:	/sbin/sh
On since	Oct 8 08:47:08	on term/18		

Sending Messages

write - Write a message to another user's terminal

Description

The **write** command is used to send messages to an active user's terminal, or to engage in two-way communication with another user. This capability is dependent on the **mesg** setting of the receiving terminal. The status of receiving terminals can be determined using the **finger** command.

write transmits one line at a time. To signal the other user when to reply, a designation, like **o** (over) is used to indicate the end of a message. Similarly, a designation, like **oo** (over and out), indicates the end of the communication.

To establish connection between the sending and receiving terminals, each user must enter the **write** command and the user's name to send the first message. A device name can be used in addition to the user name if the user is logged on other terminals. Thereafter, each user enters only the message and an indicator to reply (**o**) or to end the connection (**oo**). To terminate the connection and to return to the shell prompt, each user presses **<^d>**, which displays **<EOT>** (end of transmission) on the other terminal.

Argument

term/n Designates the terminal name; where *n* is the name of the device name associated with the terminal.

Examples

In the example on the next page, **user1** initiates the communication. Notice the use of **o** to signal the end of the user's message, and the **oo** to terminate the communication.

wall - write to all users

Reference

- *UNIX System V, Release 4 User's Reference Manual, write(1)*

Sending Messages

write user [terminal]

user1	user2
<pre>(send) \$ write user2 Are you attending the meeting this afternoon? o (message) Message from user2 . . . Yes -- Should we bring our status reports? o (reply) No, just the documentation we discussed earlier. Don't forget 1:30 sharp! See you there. oo (message) OK -- Later. oo <^d> <EOT> \$</pre>	<pre>(receive) Message from user1 . . . Are you attending the meeting this afternoon? o (reply) \$ write user1 Yes -- Should we bring our status reports? o (message) No, just the documentation we discussed earlier. Don't forget 1:30 sharp! See you there. oo (reply) OK -- Later. oo <^d> <EOT> \$</pre>

Mailing Messages

mail - Send and read mail messages

Description

Mail messages are sent to the users mail file, */var/mail/logname*. Mail is checked by */etc/profile* at login.

Send Mail

To send mail, enter the **mail** command and one or more user names. Enter the text of the message, pressing <RETURN> at the end of each line. End the message by entering <^d> on a separate line. The message is sent to the user's mail file. If **mail** is interrupted during message input, the contents of the message is placed in the *dead.letter* file in the current directory. If user errors occur during message input, it is returned to the sender with an appropriate description of the error.

To send mail to a user on another UNIX system, enter the user name as *system_name!logname*. Sending mail to a remote system requires the configuration of the BNU utilities described in Appendix D. A list of system names can be displayed with the **uname** command.

Options

-t Adds *To:* to header when mail is read

Read Mail

The system displays *you have mail* for new messages. To display the contents of the mail file, enter the **mail** command and press <RETURN>. A descriptive header displays the lognames of the sender and recipient(s); the date/time the message was sent; the subject, the type, and length of the message. The most recent message is displayed first. For long messages, press <Hold Screen> or <^s> and <^q> to pause and resume the screen display. Press <RETURN> at the ? prompt for the next message.

A number of commands are available within the **mail** program. For example, messages can be deleted, undeleted, or saved to the default *mbox* file or to the named file. At the mail prompt, enter a ? to display the command summary. To exit mail, enter **q** to save any changes and quit, or press **x** to discard any changes and quit.

Options

-p Display all messages without prompting for each
 -r Display messages in reverse order
 -e Check for presence of new mail;
 messages are not displayed
 -Fuser Forward incoming mail to named user(s)

Reference

- *UNIX System V, Release 4 User's Reference Manual, mail(1)*

Mailing Messages

`mail [-option] user(s)`

- **Send mail**

`$ mail user`

There is an unscheduled staff meeting
this afternoon at 1330 in room 302.

`<^d>`

`$ mail user3 user4 < memo`

- **Read mail**

`$ you have mail`

`$ mail`

From eva Fri Feb 14 01:16 EST 1992

To: mike

Subject: project review

Content-Type: text

Content-Length: 46

Received your recommendations -- many thanks!

`?d`

↑
to delete mail item

Mailing Messages

mailx - Enhanced mail utility to send and read mail

Description

The procedures to send and to read mail are the same as those used in **mail**. **mailx** provides extra features unavailable in **mail**. Further, **mail**'s environment can be customized. The system administrator may designate mail commands or variables in the system file `/etc/mail/mailx.rc` for all users. A user may also create a file called `.mailrc` in the home directory containing other mail variables or commands.

Send

mailx automatically prompts for a subject. **mailx** also offers several commands, called *tilde escapes* to use during message entry. These commands allow you to edit messages during message input and before the message is mailed.

Mail can be sent to users on other systems by designating the user name as `system_name!logname`. Again, sending mail to a remote system requires the configuration of the BNU utilities, described later in this module. A list of system names can be displayed with the **uname** command.

Procedures to send mail using **mailx** are described on the following pages.

Read

The message *you have mail* is displayed for new messages. **mailx** offers a variety of commands to manage mail messages.

Procedures to read mail using **mailx** are described on the following pages.

Reference

- *UNIX System V, Release 4 User's Reference Manual, mailx(1)*

Mailing Messages

```
mailx [-options] user(s)
```

- **Enhanced mail – enhanced features**
- **Incoming messages can be manipulated**
- **Tilde escape commands available in input mode**
- **Mail environment can be modified**

mailx - Sending Messages

The command format to send messages using **mailx** is illustrated on the next page. The subject prompt is automatically displayed when **mailx** is invoked. It can be designated also at the command line using the **-s** option followed by the subject, as in Example 1. Enter the text of the message and press <RETURN> at the end of each line. To send the message, press <^d> or **~.** at the beginning of a line by itself. If **mail** is interrupted during message input (for example, by pressing <^c> or <~q>), the contents of the message is placed in the *dead.letter* file in the sender's home directory.

A variety of **mailx** commands called *tilde escapes* provide additional features during message input. These special **mailx** sequences are entered at the beginning of a line by itself. Frequently used input commands (tilde escapes) are described below. A summary of tilde escapes can be displayed using the **~?** command.

Tilde Escapes (Input Commands)

-e or -v	Invoke an editor to correct and to manipulate message text; default editor for -e is ed and vi for -v ; can be modified in the <i>.profile</i> or the mailx startup file called <i>.mailrc</i> .
-p	Preview or print message up to the point -p executed
-r file	Read in the contents of a file message
-h	Display all header information which can be modified
-t user(s)	Add user name(s) to the To: list
-c user(s)	Add user name(s) to carbon copy (Cc) list
-b user(s)	Add user name(s) to blind copy (Bc) list
~! command	Escape to shell to execute UNIX command
~q	Interrupt (exit) mail; any message input is saved in the <i>dead.letter</i> file in the home directory
~x	Interrupt (exit) mail; message input is not saved
~.	Standard exit ends and sends message

Options

-s subject Indicate subject at the command line

Example

The example on the next page shows sending a message. Notice the use of the **-r** command to read in the contents of a file. The status line below the **-r** command automatically displays status information about the file (name, line count, and character count). The **~.** command ends the message input and sends the message.

Reference

- *UNIX System V, Release 4 User's Reference Manual, mailx(1)*

mailx - Sending Messages

mailx [-option] user(s)

Example

```
$ mailx user1 -s meeting
```

I am enclosing the following notes
as input to the meeting scheduled
for this afternoon.

```
~r admin/notes
```

```
"admin/notes" 14 872
```

Regret missing this meeting. Kindly
send a summary of the salient points.

```
~.
```

```
EOT
```

mailx - Reading Messages

Incoming mail is stored by default in the mailbox file, */var/mail/logname*, for each user. As messages are read, they are marked to be moved to a secondary storage file called *mbox* in the user home directory.

The message *you have mail* is displayed for new messages. To read mail, enter **mailx** and press <RETURN>. The **mailx** version number and a help reminder message are displayed. The next line prints the name of the user's mail file and a count of mail messages and their status, followed by a listing of messages and descriptive information. The > symbol appears next to the current message. An N (new) or U (unread) designates the status of each message. This is followed by descriptive information displaying the sender's logname, the date and time each message was sent, the length of each message in lines and characters, and the message subject. The ? mail prompt is automatically displayed. Messages can be displayed by pressing <RETURN> or entering a mail command.

mailx provides a variety of commands to manage incoming mail. A list of commands can be displayed by entering a question mark (?) at the mail prompt.

There are two standard methods to exit **mailx**. The **q** command moves messages that have been read to the user's *mbox* file. All other mail commands take effect. The **x** command, on the other hand, exits mail leaving all messages intact in the mailbox file and all changes are ignored.

Options

- e Check for the presence of incoming mail
does not display messages
- f file Read messages from alternate mail file
- H Display header summary information only
- N Do not display header summary information

Example

The example on the next page accesses **mailx** to read messages. The status information of current messages is displayed automatically when **mailx** is invoked. The figure contains a partial listing of available commands to manage mail messages.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **mailx(1)**

mailx – Reading Messages

mailx [-options] user(s)

Example

```
$ mailx
```

```
mailx version 4.0  Type ? for help.
```

```
"/var/mail/userb": 1 message 1 new
```

```
>N 1 userb          Tue Aug 27 10:51    10/340
```

```
? ?
```

alias,group user ...	declare alias for user names
alternates user	declare alternate names for your login
cd, chdir [directory]	chdir to directory or home if none given
!command	shell escape
copy [msglist] file	save messages to file without marking as saved
delete [msglist]	delete messages
discard, ignore header	discard header field when

[msglist] is optional and specifies messages by number, author, subject or type. The default is the current message.

```
? h
```

```
>N 1 userb          Tue Aug 27 10:51    10/340
```

```
? q
```

```
Held 1 message in /var/mail/userb
```

Forwarding Mail

notify – Receive notification and forward incoming mail

Description

This command notifies a user when new mail arrives. The **mesg** setting enables or disables display of the notification message. **notify** is also used to forward mail to another file. If an alternative file is not designated with the **-m** option when notification is activated, new mail is forwarded to *.mailfile* in the user's home directory.

Notification is activated by designating the **-y** option when the command is executed. The **-n** option deactivates mail notification. **notify** without arguments displays the current state.

Forwarding is disabled using the mail command **mail -F ""**. The **" "** offsets the null argument.

Options

-y	Activate notification
-n	Deactivate notification
-m file	Forward new mail to named file

Examples

The first example shows that notification is not active. In the second example, notification is activated using the **-y** option. This is confirmed by displaying the current setting. Notice the message indicating the location where new mail is forwarded. This message is contained in the user's mail file (*/var/mail/logname*) and is displayed when the **mail** command is executed. (The backslash at the end of the first line of output was inserted for illustration purposes to refer to a command line continuation.) The command structure in the **mail** output illustrates the "Forward to | command" facility of **mail** to implement notifications. The **%R** and **%S** refer to keywords allowed within the piped-to command structure that will be substituted before the command line is executed; for example **%R** refers to returning the path name to the originator of the message and **%S** refers to the value of the subject header line if one is present. Refer to the reference documentation on the **mail** command for more complete description of this facility.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **notify(1)**, **mail(1)**

Forwarding Mail

`notify [-options]`

Examples

```
$ notify
/usr/bin/notify: Mail notification not active
$ notify -y
$ notify
/usr/bin/notify: Mail notification active
/usr/bin/notify: New mail messages will go to '/home/user2/.mailfile'
$ mail
Your mail is being forwarded to | /usr/lib/mail/notify -m \
/home/user2/.mailfile -o %R -s %S
```

Sending Automatic Mail Reply

vacation - Send automatic mail reply and forward incoming mail

Description

vacation forwards and automatically responds to incoming mail. When new messages arrive, **mail** checks the recipient's mailbox (*/var/mail/logname*) for a message that mail is forwarded. This message is also displayed when **vacation** is executed. New mail is forwarded to *.mailfile* in the user's home directory unless another file name is designated using the **-m** option. If problems are encountered forwarding to the user's *.mailfile*, incoming mail can be forwarded to another user using the **-F** option instead of returning mail to the sender.

A default reply stored in the */usr/share/lib/mail/std_vac_msg* file is mailed to the user sending the mail. Alternatively, the user can create a reply using an editor specifying this alternative reply file name with the **-M** option of the command.

The *.maillog* file in the user's home directory records users who have already seen the reply message. An alternative log file name can be designated using the **-l** option.

Forwarding is disabled using the mail option **mail -F ""**. The "" offsets the null argument.

Options

-F user	Designate alternate recipient of incoming mail
-l log_file	Designate log file of users having received the reply message (default is <i>.maillog</i> in the home directory)
-m mail_file	Designate alternative mail file (default is <i>.mailfile</i> in the home directory)
-M reply_file	Designate alternative reply file

Examples

The example on the next page shows the message that mail is being forwarded when **vacation** is executed; again, this message resides in */var/mail/logname*. A sample default reply is also shown.

Reference

- *UNIX System V, Release 4 User's Reference Manual, vacation(1)*

Sending Automatic Mail Reply

vacation [-options]

Example

```
$ vacation  
Forwarding to | /usr/lib/mail/vacation2 -o %R
```

Sample Default Reply

```
From user2 Mon Oct 7 15:59 1991  
Subject: AUTOANSWERED!!!  
Content-Type: text  
Content-Length: 238
```

I am on vacation. I will read (and answer if necessary) your e-mail message when I return.

This message was generated automatically and you will receive it only once, although all the messages you send me while I am away WILL be saved.

Summary

- **Communication is the exchange of information.**
- **Data communication is the transmission of electronic information over distance.**
- **A data communication system consist of hardware and software.**
- **A communications network consists of interconnected devices forming a pattern called a *topology*.**
- **Data transmission is managed by software.**
- **Organizations developed to establish industry standard communication protocols. Open Systems Interconnection (OSI) is the industry standard communication model of standard protocols.**
- **There are basically two types of communication: local (same system) and remote (other system).**
- **Local UNIX communication commands are:**
 - **news** displays system news.
 - **mesg** permits or denies messages sent to a terminal.
 - **write** is used to send messages to an active terminal, or to engage in two-way communication with another user (terminal).
 - **finger** displays user information about local and remote users
 - **mail** allows messages to be sent to another user on a local or a remote system.
 - **mailx** is an enhanced mail utility to send and to read mail on local or remote system.
 - **notify** informs a user when incoming mail arrives. It is also used to forward a user's mail to a default or designated location.
 - **vacation** sends an automatic mail reply and forwards mail.

Practical Exercise

Perform the following activities at your terminal.

1. Display the number of current news items. Display their names.
2. View a specific news item.

Your instructor will assign you in pairs for the following activity using the **write** command. Decide who will initiate communication using **write**. Write several messages to one another before terminating communication.

Perform the following activities using the electronic mail utilities.

3. Mail yourself a reminder message.
 - a. View your mail when you are prompted that you have incoming mail.
 - b. Display the list of **mail** commands.
 - c. Exit **mail** discarding any changes.
4. Mail yourself another message using **mailx**. Use tilde escapes during your message input.
 - a. View your mail when you are prompted.
 - b. Display the list of available **mailx** commands.
 - c. Exit **mailx** discarding any changes.
5. Use **mail** or **mailx** to send mail to two other users. Read your incoming mail when prompted. Use any available commands and **exit**.

A

Terminal Setup

Terminal Setup

Terminals and other peripheral devices are configured in the UNIX environment, usually by the system administrator. Many terminals also allow terminal operating features to be changed. Usually, the terminals are already set properly by the system administrator. Some terminal features are "safe" to adjust, such as the cursor type, and do not result in problems. Other more critical features, such as communication settings, should be set carefully to be compatible with the host system. It is always advisable to first consult the appropriate terminal documentation, and/or the local system administrator.

The Unisys-supported UVT-1224 and TO-300 terminals provide separate menu systems through which the user identifies control parameters, such as screen scroll rate, screen background, cursor type, keyboard nationality, and communication speed and parity. Each feature is set to a predetermined value (factory default setting) until the user selects a different setting. Modified settings are considered temporary (effective until the terminal is powered off) unless a save operation is performed. Factory default settings are not generally compatible with the UNIX host system configuration. Importantly, communication parameters must match the terminal configuration on the host system. Application programs may require unique communication parameter settings.

When a Unisys-supported PC is used to communicate with a UNIX system, a communication software product establishes the appropriate communication parameters with the host system.

This appendix provides a general description of the UVT-1224 and TO-300 terminal setup menus and standard terminal keyboard control keys.

Reference

- *Video Terminal UVT-1224 Operating and Programming Guide* (UP 12956)
- *TO-300 Video Terminal Operations Guide* (UP 15935)

Terminal Setup

- **Terminal operating features can be changed**
- **Communication parameters must match terminal configuration on UNIX host system**
- **Setup screens are available for the UVT-1224 and TO-300 terminals**
 - Terminal features are preset to factory default settings
 - Modified settings are temporary until the terminal is powered off
 - New settings can be saved

UVT1224 Setup Menus

The setup key displays the main menu in the lower area of the screen, overlaying the original display temporarily. Each of the seven subordinate setup screens has a standard display. Each screen contains a screen title and the name of the terminal at the top of the screen. The greater portion of the menu contains subordinate screen selections, parameter (feature), and action fields. The last line displays status information. Subordinate screen selections are located on the top row of the main menu. An action field defines an action, such as *save* or *exit*. Parameter fields define options for each feature and display the current setting.

Setup Keys

The following keys are used in the setup screens:

Setup	Display or remove setup screens
Cursor keys (arrow)	Move cursor to desired field
Enter	Display next feature or execute selected action

To access the desired screen, move the cursor to the screen selection and press <ENTER>. The first two fields of each subordinate screen are the same. *To Next Setup* displays the next screen. *To Directory* displays the main setup screen.

To change a feature setting, move the cursor to the desired parameter field. Press <ENTER> repeatedly until the desired option is displayed as the current feature.

To execute an action, move the cursor to the desired action field and press <ENTER>.

Note: Communication parameters must match the host system. The *Default* action field sets all configuration parameters to the factory default, which may not be compatible with the host system.

Summary

The procedure listed below changes setup parameters.

1. Access the setup menu.
2. Select the desired screen, feature, or action. Make the desired changes.
3. Save the changes using the *Save* action located on the main screen.
4. Exit the setup menu using the *Exit* action (main screen), or by pressing the <SETUP> key.

Reference

- *Video Terminal UVT-1224 Operating and Programming Guide*, Section 3, Setup Screens

UVT-1224 Setup Menus

- **Standard screen display includes**
 - Screen title and terminal name
 - Fields to select subordinate screens, actions, or features
 - Status line

- **Keys**
 - Setup
 - Cursor (arrow) keys
 - Enter

- **Procedure to change terminal settings**
 1. Access setup menu
 2. Select desired screen, feature or action
 3. Save changes
 4. Exit setup menu

Standard UVT-1224 Control Keys

Although different keyboard models are available for this terminal, the table on the next page lists standard control keys.

Standard UVT-1224 Control Keys

Key	Description
Arrow keys	Repositions the cursor
Break	Transmits interrupt/cancel message to host; enabled/disabled through Setup; also ^c
BS (F12)	Erases the character to left of cursor; same as stty erase character
Caps Lock	Locks alphabetic keys in uppercase; can be adjusted through Setup to lock symbols as well
Compose	Generates characters not available on the keyboard, but a part of the terminal character set
Ctrl	Used in combination with other keys to perform functions
Hold Screen	Pauses screen scroll; toggles to resume scroll; also <^s> and <^q>, respectively
Print Screen	Prints screen display; printer must be directly attached to terminal; feature must be configured in Setup
Return	Loads and executes command
Setup	Menu of adjustable parameters controlling terminal operation and communication
Shift	Selects uppercase alphanumeric characters
Status	Displays 25th line on screen providing status information of the terminal or messages from the host system; enable/disable through Setup

TO-300 Setup Menus

The main setup screen is displayed by pressing the setup key, F3. Other key combinations can also be used depending on the keyboard type. Refer to the TO-300 documentation for information.

The setup menu for this terminal also consists of eight screens. Unlike the UVT-1224 screens, TO-300 setup displays full screens. Each setup screen displays the terminal name, screen title, and a version number at the top of the screen, followed by a double-column display of parameters and their current settings. The lower left area of the screen displays status prompts and help windows. The lower right portion of the screen displays a list of function keys (F1 through F9) to access the other setup screens and to exit the setup menu.

Setup Help

Two help windows are available providing general information on how to use the setup screens. Press <h> to display the first help window listing the keys to select setup parameters, to move between columns of parameters, or to select other setup screens.

Up/down arrow	Select parameter
Left/right arrow	Select parameter option
Tab	Select column listing parameters
F1 through F8	Select screen

The second help window lists the keys that perform actions, such as saving the current settings. This screen is accessed by pressing any key from the first help screen.

E Exit Setup	R Restore Saved Values
D Default Setup	P Select Host Port
S Save Current Values	F1-F8 Select Screen

Note: Communication features must match the host system. The *Default Setup* action sets all configuration parameters to the factory default, which may not be compatible with the host system.

Summary

The procedure listed below changes setup parameters.

1. Access the setup menu.
2. Select the screen, or parameter to change. Make the desired changes.
3. Press <s> to save setup parameters.
4. Press <e> or <F9> to exit setup screens.

Reference

- *TO-300 Video Terminal Operations Guide* (UP 15935)

TO-300 Setup Menus

- **Standard screen display includes**
 - Terminal name, screen title, and version number
 - Double-column listing of parameters and current settings
 - Help windows
 - List of function keys to access other screens

- **Help**
 - First window lists keys to select setup parameters or other screens
 - Second window lists action keys

- **Procedure to change terminal settings**
 1. Access setup menu
 2. Select desired screen, action, or feature
 3. Save changes
 4. Exit setup menu

Standard TO-300 Control Keys

Although different keyboard models are available for this terminal, the table on the next page lists standard control keys.

Standard TO-300 Control Keys

Key	Description
Arrow keys	Repositions the cursor
Break	Sends interrupt/cancel message to host; enabled/disabled through Setup; also ^c
BS (F12)	Erases the character to left of cursor; same as stty erase character
Caps Lock	Locks alphabetic keys in uppercase; can be adjusted through Setup to lock symbols as well
Ctrl	Used in combination with other keys to perform functions
Hold Screen	Pauses screen scroll; toggles to resume scroll; also <^s> and <^q>, respectively
Print Screen	Prints screen display; printer must be directly attached to terminal; feature must be configured in Setup
Return	Loads and executes command
Send Block	Places terminal in block mode for local editing; host system software must support this feature
Setup	Menu of adjustable parameters controlling terminal operation and communication
Shift	Selects uppercase alphanumeric characters

PC as a UNIX Terminal

To use a PC as a terminal in the UNIX environment, some type of communication software is used to configure communication parameters allowing the PC to communicate with the UNIX system. The configuration and setup of the PC, as well as overall system device management, is a key responsibility of the local system administrator, and generally not a concern of the casual user.

PC as a UNIX Terminal

- **Communication software used to establish communication between PC and UNIX system**
- **Generally, configured and set up by local system administrator**

Standard PC Control Keys

Although different keyboard models are available for this terminal, the table on the next page lists standard control keys.

Standard PC Control Keys

Key	Description
Alt	Used in conjunction with other keys to perform functions
Arrow keys	Reposition the cursor
Backspace	Erases the character to left of cursor
Break	Sends interrupt/cancel message to host; enabled/disabled with MS-DOS Break command
Caps Lock	Locks alphabetic keys in uppercase
Ctrl	Used in combination with other keys to perform functions
Enter	Loads and executes command, or <RETURN>
PrtSc	Prints screen display; printer must be directly attached to terminal; Shift-PrtSc to print current screen and Ctrl-PrtSc for continuous screen prints
<i>(ON OFF)</i> } → ^s and ^q	Pauses and resumes screen scroll
Shift	Selects uppercase alphanumeric characters

B

**Standard Directories
and Files**

Standard UNIX Directories and Files

Some of the directories and files that make up the UNIX file hierarchy are delivered with the system. Standard directories and files are described in this section.

Standard Directories

- /bin* Symbolic link to the */usr/bin* directory
- This directory is created to provide compatibility with previous versions of UNIX.
- /etc* Contains the system administration directories and files, including boot, configuration, startup, shutdown, and default login directories and files
- The files within this directory structure are usually ASCII files and are usually specific to the computer model on which the operating system runs. Many files can be reconfigured for the particular computer.
- /dev* Contains the special I/O files
- These are the block and character files required to transmit and to receive data to or from the various input and output devices, including hard disks, diskettes, terminals, printers, and magnetic tape. Many of the files are grouped according to device type in special directories.
- /hinu* Contains hardware configuration files
- The */hinu* directory is the mount point for a file system unique to Unisys. It is this file system that contains the hardware inventory files.
- gethinu*
- /home* Default directory for user login (home) directories
- /install* Used by the **sysadm** command to mount utility packages for installation and removal of software
- /lib* Symbolic link to the */usr/lib* directory
- This directory is created to provide compatibility with previous versions of UNIX.
- /mls* Multi-Level Security directory that contains the commands and utilities associated with audit trailing

- /proc* Mount point for another file system type
- Files in this directory represent the current processes in memory. The names of the file is the PID number of the process.
- /sbin* Contains the ^{process id} programs required by *init* for system booting or for recovery from a system failure
- Also contains the single-user administrative commands.
- /stand* Mount point for the boot file system (bfs)
- Files in this directory are used only at boot time. Some of these files are also found in the */etc/initprog* and */etc/default* directories.
- /tmp* Repository for temporary files
- ^{open}
^{rather} It should never be used for long-term storage because the entire directory is removed whenever the operating system is started.
- /usr* Contains user-oriented system and application files that remain static, unchanged
- Files in this directory include user commands and library files, like *man*.
- /var* Contains system files that change with system use
- Files in this directory include news bulletins, user mail, and print and at/batch spool files.

Standard Files

<i>/etc/cron.d/at.allow</i> <i>Schedule</i>	User access file for at and batch commands
<i>/etc/group</i>	Group control file
<i>/etc/inittab</i>	Executes processes defined at various system activity levels
<i>/etc/issue</i>	Pre-login message
<i>/etc/motd</i>	Post-login message-of-the-day
<i>/etc/passwd</i>	User control file
<i>/etc/profile</i>	System-wide environment file
<i>/etc/shadow</i>	Login password control file (<i>has encrypted passwords</i>)
<i>/etc/ttydefs</i>	Contains terminal line setting information
<i>/usr/lib/saf/sac</i>	Service Access Controller starts and administers port monitors
<i>/usr/lib/saf/ttymon</i>	Port monitor for terminal ports
<i>/usr/lib/terminfo</i>	Terminal capability database often used by screen-oriented applications (like vi) and some UNIX commands (like ls , more)

C

Command Summary

Command List

cal	Display online calendar (p. 2-23)
cancel	Remove print request (p. 5-35)
cat	Concatenate and print files (p. 5-16)
cd	Change current directory (p. 5-10)
chgrp	Change the group ownership of a file (p. 6-28)
chmod	Change permission mode of a file or directory (p. 6-12)
chown	Change file ownership (p. 6-20)
cp	Copy files (p. 5-20)
date	Display system date and time (p. 2-9/23)
disable	Place printer offline (p. 5-34)
ed	UNIX line editor (p. 4-4)
enable	Place printer online (p. 5-36)
exit	Log out (p. 2-18)
file	Determine file type (p. 5-14)
find	Locate files matching designated conditions and execute the specified action (p. 5-38)
finger	Display information about local and remote users (p. 7-12)
fmt	Text formatter (p. 5-26)

grep	Search file(s) for designated pattern (p. 5-40)
groups	Display group membership of user (p. 6-26)
id	Display user logname, group name, user and group Ids (p. 6-24)
ln	Link files; create multiple pointers (references) to named file (p. 5-24)
logname	Display user log name (p. 2-25)
lp	Request LP print service (p. 5-28)
lpstat	Display status of LP print service (p. 5-30)
ls	List contents of directory (p. 5-8)
mail	Send or read mail to users (p. 7-16)
mailx	Enhanced mail service (p. 7-18)
man	Access to online reference manuals (p. 2-16)
mandex	Menu-driven indexing system to online reference manuals (p. 2-18)
mesg	Permit or deny messages (p. 7-10)
mkdir	Make directories (p. 5-6)
mv	Move files (p. 5-22)
newgrp	Change user's current group (p. 6-30)
news	Display system news items (p. 7-8)
notify	Notify user of the arrival of new mail (p. 7-24)

passwd	Change login password and password attributes (p. 2-14)
pg	Display contents of file(s) a screen at a time (p. 5-18)
pr	Format file(s) for printing (p. 5-26)
pwd	Print working directory (p. 5-4)
rm	Remove file(s) (p. 5-46)
rmdir	Remove directories (p. 5-12)
sort	Order contents of file and display result on the standard output (p. 5-42)
stty	Set or display terminal options (p. 2-23)
tty	Display device file name associated with terminal (p. 2-23)
umask	Change default permission mode (p. 6-18)
vacation	Automatically respond to incoming mail messages (p. 7-26)
vi	Screen-oriented editor (p. 4-32)
who	Display status of users currently logged in (p. 2-23)
write	Two-way communication with another user (active terminal) (p. 7-14)

D

**Basic Networking
Utilities**

Basic Networking Utilities (BNU) Overview

The Basic Network Utilities (BNU) is a collection of programs and files that allow users on one UNIX system to communicate with other UNIX systems. It was formerly called UUCP, which represents UNIX-to-UNIX Communications Package. UUCP system was developed in 1976 by AT&T. The current version was made available in 1983 with UNIX System V Release 3. Commands and files have been modified or added to facilitate UUCP administration and to provide support for more advanced communication capability. This version is also referred to as HoneyDanBer UUCP, derived from the authors' names.

BNU provides a form of communications called *store-and-forward*. If the remote system is unavailable, the communication is stored until the computer is available. This is suitable for non-real-time communications, like file transfer, but inappropriate for interactive communications.

Each system on a BNU network has files that describe the other systems that are linked to it. These files also describe the types of links available for the other systems. BNU is set up and maintained by the system administrator.

Because BNU is provided with the basic UNIX operating system, there is no additional cost for software. Importantly, the UNIX kernel does not require changes.

- Basic networking functions:
 - Display system names
 - Transfer files
 - Display file transfer status
 - Call remote UNIX system
 - Execute commands on remote system

Basic Networking Utilities

- **Provides UNIX-to-UNIX communications**
- **Formerly UUCP**
- **Uses store-and-forward communications**
- **Low startup costs**
- **No kernel changes required**

Displaying System Name

uname - Display the local system name

Description

This command is useful to display identifying information about the local system. Without arguments, **uname** displays the node name by which the system is identified in a communications network. Most of the options, except **-a**, display selected information.

Options

-a	All information
-m	Machine (hardware) name
-n	Node name
-r	Release of operating system
-s	Name of operating system
-v	Version of operating system

Examples

The first example displays all identifying information about the system.

unix	usa	4.0	2	i386
⋮	⋮	⋮	⋮	⋮
System name	Node name	Release	Version	Machine name

In the second example, the **-m** option is used to display the hardware name. The last example uses the combined options **-sn** to display the system and node names.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **uname(1)**

Displaying System Name

```
uname [-options]
```

Example 1

```
$ uname -a  
unix usa 4.0 2 i386 386/AT
```

Example 2

```
$ uname -m  
i386
```

Example 3

```
$ uname -sn  
unix usa
```

Displaying Network System Names

uname - Display names of all BNU (uucp) systems

Description

This command displays the names of all other systems recognized by the Basic Networking Utilities (BNU), also referred to as UNIX-to-UNIX COPY (uucp). These system names are entered in a uucp */etc/uucp/Systems* file maintained by the system administrator.

Options

-l Display the local system name

Example

The example on the next page shows a sample listing of system names. The -l option is used to display the local system name.

Reference

UNIX System V, Release 4 User's Reference Manual, **uname(1)**

Displaying Network System Names

```
uuname [-option]
```

Example

```
$ uuname  
mart  
wopper  
tower
```

```
$ uuname -l  
wjnr
```

Transferring Files

uucp – Copy file(s) to another UNIX system

Description

The **uucp** command copies specified source files to the designated destination file. Either source or destination file can be on the local or remote system. However, they cannot both be on the local system. The path name of a file located on a remote system must contain all the intervening system names, separated by exclamation points (!), as in *system_name1/pathname* or *system_name1!system_name2!system_name3!/pathname*.

Special characters used in file names will be expanded on the destination system.

File path names can be one of the following types.

- Full path name
- Relative path name
- Path name preceded by *-logname* refers to the home directory of the remote logname
- Path name preceded by *~/destination* refers to an existing remote file or directory. The destination is appended to the */var/spool/uucppublic* directory. Add a */* to the destination to ensure the destination is a directory; if it does not exist, it is created.

Options

-C	Copy local file to spool directory for transfer
-j	Display uucp job number to track job status or to terminate job
-m	Notify requester of job completion by mail
-nologname	Notify recipient that file was sent
-r	Queue job for transfer, do not send

Examples

The first example copies the local *myfile* to *user2* on the second remote system, notifying the requester of the job completion. The second example copies the *locfile* to an existing directory in the *uucppublic* directory on the next remote system, notifying the recipient that the file was sent. The last example queues *file2* on the second remote system to the *user1* directory in the *uucppublic* directory of the first remote system, requesting the job status number.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **uucp(1)**

Transferring Files

```
uucp [-options] source_file destination_file
```

Examples

```
$ uucp -m myfile sys1!sys2!/var/spool/uucppublic/user2
```

```
$ uucp -nremuser locfile sys1!~/dir1
```

```
$ uucp -jr sys1!sys2!~/file2 sys1!~/user1/
```

Transferring Files

uuto - Copy source files to public directory

uupick - Accept or reject files transferred to user

Description

These commands are used together to transfer one or more files to a user on another UNIX system. The destination is entered in the format *system1/system2/user*. **uuto** sends a copy of the source file(s) to the */var/spool/uucppublic/PUBDIR* directory (specifically, its *receive/logname* subdirectory) on the other system and notifies the designated user by mail. The sender can request notification of the complete transfer using the **-m** option.

When the user is notified, the **uupick** command is used to search *PUBDIR* for files destined for the user. When a file is identified, the file name is displayed. The user then selects one of several **uupick** commands to dispose of the file. The **-s** option directs **uupick** to search *PUBDIR* only for files sent from the named system.

uupick commands include

*	Display command summary
<RETURN>	Go to next entry
d	Delete entry
m [directory]	Move entry to current or named directory
a [directory]	Move all entries to current or named directory
p	Print content of file
q or <^d>	Quit
!command	Execute named UNIX command

Options

-m	uuto : Notify sender that file transfer is complete
-ssystem	uupick : Search only for files sent by named system

Examples

The first example copies the *employees* file for *userX* to the *PUBDIR* on the *sys4* remote system. In the second example, the **-s** option of the **uupick** command is used to find all files from *sys2*.

Reference

- *UNIX System V, Release 4 User's Reference Manual*, **uuto(1)**, **uupick(1)**

Transferring Files

```
uuto [-option] source destination
```

```
uupick [-option] user
```

Example 1

```
$ uuto -m employees sys4!userX
```

Example 2

```
$ uupick -ssys2
```

Checking Job Status

uustat – Display or cancel uucp job requests

Description

The basic functions of the **uustat** command are to display the status of job requests and to cancel job requests scheduled by the user with **uucp** or **uuto**. The output of **uustat** (without options) displays the following information:

- Job number
- Date and time job was queued
- S (send) or R (receive) designation
- Name of destination system
- Sender's logname
- Size (in bytes) transferred
- File name

More frequently used options of the **uustat** command are listed below. Options **-a** and **-q** lists local and remote jobs queued by **uucp** and **uuto**. The **-k** option cancels the named job Id. The **-r** option is used to update the modification time of a job so that it is not removed by the cleanup daemon.

Options

- | | |
|-------------------------|---|
| -a | List all jobs queued |
| -q | List the jobs queued for each remote system |
| -k <i>job_Id</i> | Cancel designated job in queue |
| -r | Rejuvenate job queue date |

Reference

- *UNIX System V, Release 4 User's Reference Manual, uustat(1)*

Checking Job Status

```
uustat [-option]
```

- Report status of queued jobs
- Cancel named job

Calling Another UNIX System

cu - Connect to a remote system to transfer files or to execute commands on the remote system

Description

The **cu** command provides login capabilities on another system to perform file transfer or to execute commands on the remote system while maintaining the local UNIX session.

The *destination* components of the **cu** command may be the telephone number of a remote system, the name of a remote system, or a LAN address.

cu has two operating phases: *connect* and *converse*. The connect phase uses BNU configuration files to establish connection with the remote system. The remote system displays a login prompt signaling the end of the connection phase. At this point, the user interacts with the remote shell using **cu** commands. The **~.** command terminates the remote connection and returns you to the local system.

signoff

Options

- d Display diagnostic activity
- l*line* Specify device name used to connect;
 destination may be omitted in the command line
- s*speed* Specify transmission speed

Examples

The first example establishes remote connection to the *sys01* system. The second example specifies the device (terminal) name as the destination.

Reference

- *UNIX System V Release 4 User's Reference Manual, cu(1)*

Calling Another UNIX System

`cu [-option] destination`

Example 1

```
$ cu sys01
```

Example 2

```
$ cu -l/dev/term/10
```

Remote Command Execution

uux - Execute commands on remote systems

Description

The **uux** command executes a UNIX command on a remote system. The commands that can be executed are restricted by the `/etc/uucp/Permissions` file. It is useful, however to print information on a remote system that has the desired printer capability.

The *command-string* is a shell command line, including such things as input-output redirection, or command pipelines. The difference between the *command-string* and a regular shell command is the use of system names to qualify the command name and/or the file names.

File names can be one of the following:

- A full path name
- A relative path name
- A path name preceded by *~logname*. This designation is replaced by the user's home directory.

Only the first command of a pipeline can have a system name precede it, all other commands are executed on the system specified for the first command.

Options

- n Do not send notification to the user
- mfile Report status of transfer in named file

Examples

In the example, the local *testfile* is printed on the remote printer on the *host* system.

Reference

- *UNIX System V, Release 4 User's Reference Manual, uux(1)*

Remote Command Execution

```
uux [-option] command_string
```

Example

```
$ pr textfile | uux host!lp
```

BNU Command Summary

- Remote UNIX communication commands include:
 - **uname** displays the local system name.
 - **uuname** displays names of all BNU (uucp) systems.
 - **uucp** copies files to another system.
 - **uuto** copies files to a remote public directory.
 - **uupick** accepts or rejects file transferred to a remote public directory with **uuto**.
 - **uustat** displays or cancels uucp job requests.
 - **cu** allows connection to a remote system to transfer files or to execute commands on the remote system.
 - **uux** executes UNIX system commands on remote systems.

