Introduction to Unix

Frank G. Fiamingo

Linda DeBula

Linda Condron

University Technology Services

The Ohio State University

September 23, 1998

© 1996-1998 University Technology Services, The Ohio State University, Baker Systems Engineering Building, 1971 Neil Avenue, Columbus, OH 43210.

All rights reserved. Redistribution and use, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Neither the name of the University nor the names of its contributors may be used to endorse or promote products or services derived from this document without specific prior written permission.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. THIS PUBLICATION MAY INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

UNIX is a registered trademark of The Open Group, AT&T is a trademark of American Telephone and Telegraph, Inc.

This publication is provided “as is” without warranty of any kind. This publication may include technical inaccuracies or typographical errors.

Copyright and URLs revised September 1998

The authors’ email addresses are:

Frank Fiamingo	fiamingo.1@osu.edu
Linda DeBula	debula.2@osu.edu
Linda Condron	condron.1@osu.edu

This document can be obtained via:

http://wks.uts.ohio-state.edu/unix_course/unix.html
or
ftp://wks.uts.ohio-state.edu/unix_course/unix_book.ps

Table of Contents

1 History of Unix 7
2 Unix Structure 9
2.1 The Operating System 9
2.2 The File System 11
2.3 Unix Directories, Files and Inodes 12
2.4 Unix Programs 13
3 Getting Started 14
3.1 Logging in 14
3.1.1 Terminal Type 14
3.1.2 Passwords 15
3.1.3 Exiting 15
3.1.4 Identity 16
3.2 Unix Command Line Structure 16
3.3 Control Keys 17
3.4 stty - terminal control 17
3.5 Getting Help 19
3.6 Directory Navigation and Control 20
3.6.1 pwd - print working directory 21
3.6.2 cd - change directory 21
3.6.3 mkdir - make a directory 22
3.6.4 rmdir - remove directory 22
3.6.5 ls - list directory contents 23
3.7 File Maintenance Commands 25
3.7.1 cp - copy a file 26
3.7.2 mv - move a file 26
3.7.3 rm - remove a file 27
3.7.4 File Permissions 27
3.7.5 chmod - change file permissions 28
3.7.6 chown - change ownership 29
3.7.7 chgrp - change group 29
3.8 Display Commands 30
3.8.1 echo - echo a statement 30

3.8.2 cat - concatenate a file 31
3.8.3 more, less, and pg - page through a file 31
3.8.4 head - display the start of a file 32
3.8.5 tail - display the end of a file 32
4 System Resources & Printing 33
4.1 System Resources 33
4.1.1 df - summarize disk block and file usage 34
4.1.2 du - report disk space in use 34
4.1.3 ps - show status of active processes 35
4.1.4 kill - terminate a process 36
4.1.5 who - list current users 37
4.1.6 whereis - report program locations 37
4.1.7 which - report the command found 38
4.1.8 hostname/uname - name of machine 38
4.1.9 script - record your screen I/O 38
4.1.10 date - current date and time 40
4.2 Print Commands 41
4.2.1 lp/lpr - submit a print job 41
4.2.2 lpstat/lpq - check the status of a print job 42
4.2.3 cancel/lprm - cancel a print job 42
4.2.4 pr - prepare files for printing 43
5 Shells 45
5.1 Built-in Commands 46
5.1.1 Sh 46
5.1.2 Csh 47
5.2 Environment Variables 48
5.3 The Bourne Shell, sh 49
5.4 The C Shell, csh 50
5.5 Job Control 51
5.6 History 52
5.7 Changing your Shell 54
6 Special Unix Features 55
6.1 File Descriptors 55
6.2 File Redirection 55
6.2.1 Csh 56
6.2.2 Sh 57
6.3 Other Special Command Symbols 58
6.4 Wild Cards 58
7 Text Processing 59
7.1 Regular Expression Syntax 59
7.2 Text Processing Commands 61
7.2.1 grep 61
7.2.2 sed 65

7.2.3 awk, nawk, gawk 67
8 Other Useful Commands 70
8.1 Working With Files 70
8.1.1 cmp - compare file contents 71
8.1.2 diff - differences in files 72
8.1.3 cut - select parts of a line 73
8.1.4 paste - merge files 74
8.1.5 touch - create a file 76
8.1.6 wc - count words in a file 77
8.1.7 ln - link to another file 78
8.1.8 sort - sort file contents 79
8.1.9 tee - copy command output 82
8.1.10 uniq - remove duplicate lines 84
8.1.11 strings - find ASCII strings 85
8.1.12 file - file type 86
8.1.13 tr - translate characters 86
8.1.14 find - find files 89
8.2 File Archiving, Compression and Conversion 91
8.2.1 File Compression 91
8.2.2 tar - archive files 93
8.2.3 uuencode/uudecode - encode a file 94
8.2.4 dd - block copy and convert 95
8.2.5 od - octal dump of a file 96
8.3 Remote Connections 98
8.3.1 TELNET and FTP - remote login and file transfer protocols 98
8.3.2 finger - get information about users 100
8.3.3 Remote commands 101
9 Shell Programming 103
9.1 Shell Scripts 103
9.2 Setting Parameter Values 103
9.3 Quoting 104
9.4 Variables 105
9.5 Parameter Substitution 107
9.6 Here Document 109
9.7 Interactive Input 110
9.7.1 Sh 110
9.7.2 Csh 110
9.8 Functions 111
9.9 Control Commands 113
9.9.1 Conditional if 113
9.9.1.1 Sh 113
9.9.1.2 Csh 114
9.9.2 Conditional switch and case 115
9.9.2.1 Sh 115

9.9.2.2 Csh 116
9.9.3 for and foreach 117
9.9.3.1 Sh 117
9.9.3.2 Csh 117
9.9.4 while 118
9.9.4.1 Sh 118
9.9.4.2 Csh 119
9.9.5 until 119
9.9.6 test 120
9.9.7 C Shell Logical and Relational Operators 122
10 Editors 123
10.1 Configuring Your vi Session 124
10.2 Configuring Your emacs Session 125
10.3 vi Quick Reference Guide 126
10.4 emacs Quick Reference Guide 127
11 Unix Command Summary 128
11.1 Unix Commands 128
12 A Short Unix Bibliography 131
12.1 Highly Recommended 131
12.2 Assorted Others 131

CHAPTER 1 History of Unix

1965 Bell Laboratories joins with MIT and General Electric in the development effort for the new operating system, Multics, which would provide multi-user, multi-processor, and multi-level (hierarchical) file system, among its many forward-looking features.

1969 AT&T was unhappy with the progress and drops out of the Multics project. Some of the Bell Labs programmers who had worked on this project, Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy designed and implemented the first version of the Unix File System on a PDP-7 along with a few utilities. It was given the name UNIX by Brian Kernighan as a pun on Multics.

1970, Jan 1 time zero for UNIX

1971 The system now runs on a PDP-11, with 16Kbytes of memory, including 8Kbytes for user programs and a 512Kbyte disk.

Its first real use is as a text processing tool for the patent department at Bell Labs. That utilization justified further research and development by the programming group. UNIX caught on among programmers because it was designed with these features:

- programmers environment
- simple user interface
- simple utilities that can be combined to perform powerful functions
- hierarchical file system
- simple interface to devices consistent with file format
- multi-user, multi-process system
- architecture independent and transparent to the user.

1973 Unix is re-written mostly in C, a new language developed by Dennis Ritchie. Being written in this high-level language greatly decreased the effort needed to port it to new machines.

1974 Thompson and Ritchie publish a paper in the Communications of the ACM describing the new Unix OS. This generates enthusiasm in the Academic community which sees a potentially great teaching tool for studying programming systems development. Since AT&T is prevented from marketing the product due to the 1956 Consent Decree they license it to Universities for educational purposes and to commercial entities.

1977 There are now about 500 Unix sites world-wide.

1980 BSD 4.1 (Berkeley Software Development)

1983 SunOS, BSD 4.2, SysV

1984 There are now about 100,000 Unix sites running on many different hardware platforms, of vastly different capabilities.

1988 AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4). This would later be developed into UnixWare and Solaris 2.

1993 Novell buys UNIX from AT&T

1994 Novell gives the name "**UNIX**" to X/OPEN

1995 Santa Cruz Operations buys UnixWare from Novell. Santa Cruz Operations and Hewlett-Packard announce that they will jointly develop a 64-bit version of Unix.

1996 International Data Corporation forecasts that in 1997 there will be 3 million Unix systems shipped world-wide.

CHAPTER 2 Unix Structure

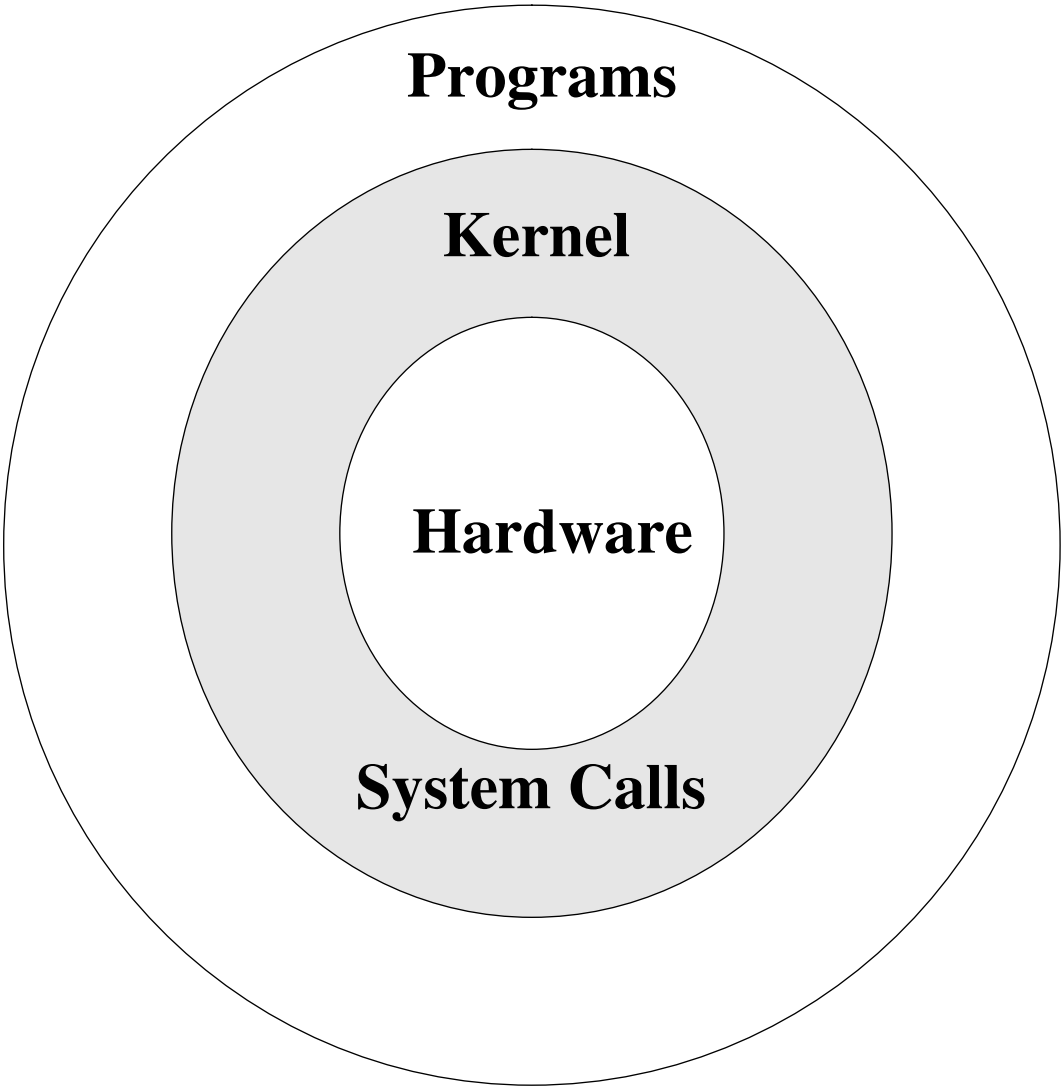
2.1 The Operating System

Unix is a layered operating system. The innermost layer is the hardware that provides the services for the OS. The operating system, referred to in Unix as the **kernel**, interacts directly with the hardware and provides the services to the user programs. These user programs don't need to know anything about the hardware. They just need to know how to interact with the kernel and it's up to the kernel to provide the desired service. One of the big appeals of Unix to programmers has been that most well written user programs are independent of the underlying hardware, making them readily portable to new systems.

User programs interact with the kernel through a set of standard **system calls**. These system calls request services to be provided by the kernel. Such services would include accessing a file: open close, read, write, link, or execute a file; starting or updating accounting records; changing ownership of a file or directory; changing to a new directory; creating, suspending, or killing a process; enabling access to hardware devices; and setting limits on system resources.

Unix is a **multi-user**, **multi-tasking** operating system. You can have many users logged into a system simultaneously, each running many programs. It's the kernel's job to keep each process and user separate and to regulate access to system hardware, including cpu, memory, disk and other I/O devices.

FIGURE 2.1 **Unix System Structure**

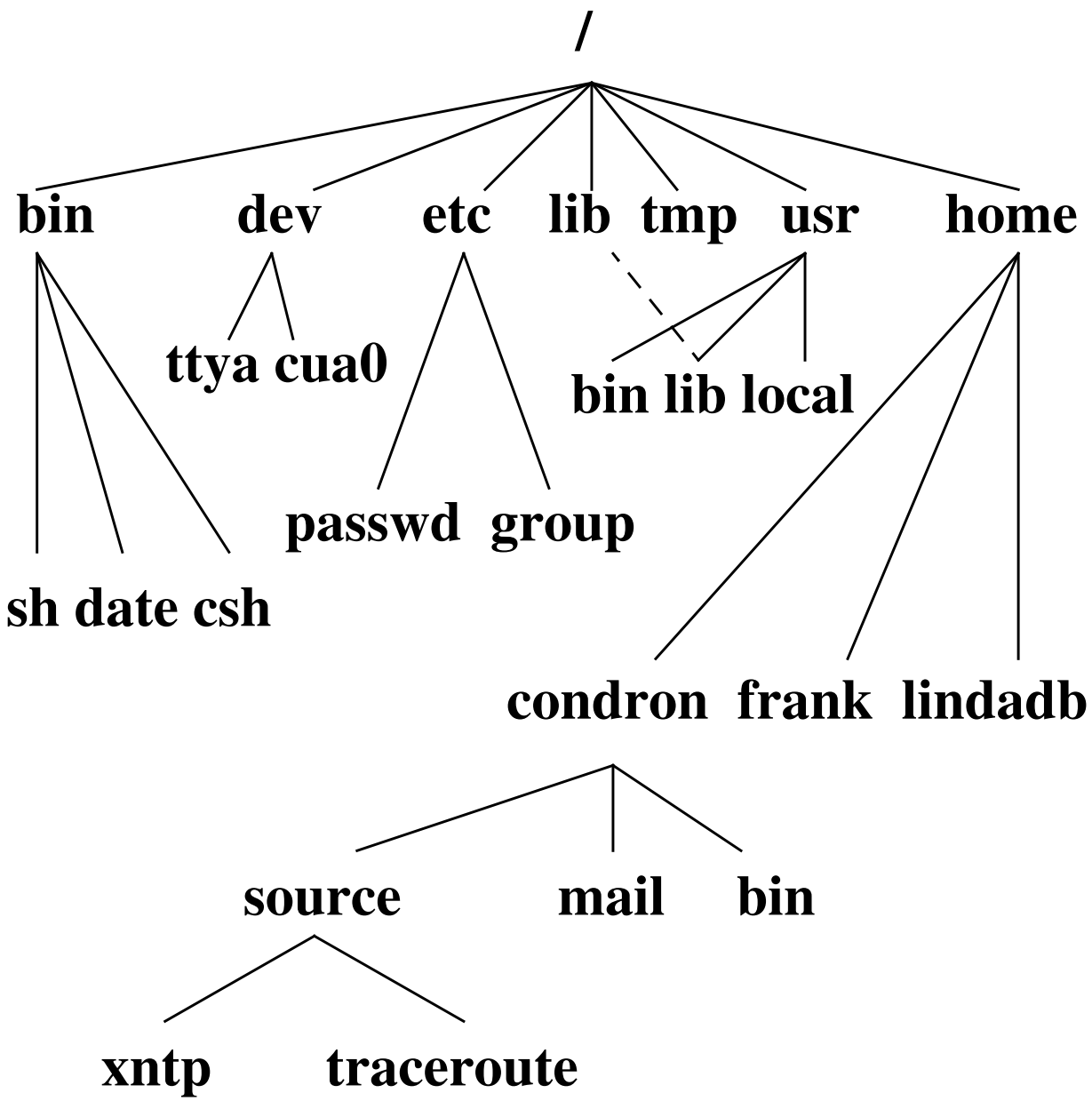


 © 1998 University Technology Services, The Ohio State University

2.2 The File System

The Unix file system looks like an inverted tree structure. You start with the **root** directory, denoted by **/**, at the top and work down through sub-directories underneath it.

FIGURE 2.2 **Unix File Structure**



Each node is either a **file** or a **directory** of files, where the latter can contain other files and directories. You specify a file or directory by its **path name**, either the full, or absolute, path name or the one relative to a location. The full path name starts with the root, /, and follows the branches of the file system, each separated by /, until you reach the desired file, e.g.:

```
/home/condron/source/xntp
```

A relative path name specifies the path relative to another, usually the current working directory that you are at. Two special directory entries should be introduced now:

.	the current directory
..	the parent of the current directory

So if I'm at /home/frank and wish to specify the path above in a relative fashion I could use:

```
../condron/source/xntp
```

This indicates that I should first go up one directory level, then come down through the **condron** directory, followed by the **source** directory and then to **xntp**.

2.3 Unix Directories, Files and Inodes

Every **directory** and **file** is listed in its parent directory. In the case of the root directory, that parent is itself. A directory is a file that contains a table listing the files contained within it, giving file names to the **inode** numbers in the list. An inode is a special file designed to be read by the kernel to learn the information about each file. It specifies the permissions on the file, ownership, date of creation and of last access and change, and the physical location of the data blocks on the disk containing the file.

The system does not require any particular structure for the data in the file itself. The file can be ASCII or binary or a combination, and may represent text data, a shell script, compiled object code for a program, directory table, junk, or anything you would like.

There's no header, trailer, label information or **EOF** character as part of the file.

2.4 Unix Programs

A **program**, or **command**, interacts with the kernel to provide the environment and perform the functions called for by the user. A program can be: an executable shell file, known as a shell script; a built-in shell command; or a source compiled, object code file.

The **shell** is a command line interpreter. The user interacts with the kernel through the shell. You can write ASCII (text) scripts to be acted upon by a shell.

System programs are usually binary, having been compiled from C source code. These are located in places like **/bin**, **/usr/bin**, **/usr/local/bin**, **/usr/ucb**, etc. They provide the functions that you normally think of when you think of Unix. Some of these are ***sh***, ***csh***, ***date***, ***who***, ***more***, and there are many others.

 © 1998 University Technology Services, The Ohio State University

CHAPTER 3 Getting Started

3.1 Logging in

After connecting with a Unix system, a user is prompted for a **login** username, then a **password**. The login username is the user's unique name on the system. The password is a changeable code known only to the user. At the **login** prompt, the user should enter the username; at the **password** prompt, the current password should be typed.

Note: **Unix is case sensitive.** Therefore, the **login** and **password** should be typed exactly as issued; the login, at least, will normally be in lower case.

3.1.1 Terminal Type

Most systems are set up so the user is by default prompted for a terminal type, which should be set to match the terminal in use before proceeding. Most computers work if you choose "**vt100**". Users connecting using a Sun workstation may want to use "**sun**"; those using an X-Terminal may want to use "**xterms**" or "**xterm**".

The terminal type indicates to the Unix system how to interact with the session just opened.

Should you need to reset the terminal type, enter the command:

setenv TERM ***<term type>*** - if using the C-shell (see Chapter 4.)

(On some systems, e.g. MAGNUS, it's also necessary to type "***unsetenv TERMCAP***".)

-or-

TERM=<***term type***>; export TERM - if using the Bourne shell (see Chapter 4.)

where <***term type***> is the terminal type, such as **vt100**, that you would like set.

 © 1998 University Technology Services, The Ohio State University

3.1.2 Passwords

When your account is issued, you will be given an initial password. It is important for system and personal security that the password for your account be changed to something of your choosing. The command for changing a password is "***passwd***". You will be asked both for your old password and to type your new selected password twice. If you mistype your old password or do not type your new password the same way twice, the system will indicate that the password has not been changed.

Some system administrators have installed programs that check for appropriateness of password (is it cryptic enough for reasonable system security). A password change may be rejected by this program.

When choosing a password, it is important that it be something that could not be guessed -- either by somebody unknown to you trying to break in, or by an acquaintance who knows you. Suggestions for choosing and using a password follow:

Don't	use a word (or words) in any language
	use a proper name
	use information that can be found in your wallet
	use information commonly known about you (car license, pet name, etc)
	use control characters. Some systems can't handle them
	write your password anywhere
	ever give your password to *anybody*
Do	use a mixture of character types (alphabetic, numeric, special)
	use a mixture of upper case and lower case
	use at least 6 characters
	choose a password you can remember
	change your password often
	make sure nobody is looking over your shoulder when you are entering your password

3.1.3 Exiting

^D - indicates end of data stream; can log a user off. The latter is disabled on many systems

^C - interrupt

logout - leave the system

exit - leave the shell

3.1.4 Identity

The system identifies you by the user and group numbers (**userid** and **groupid**, respectively) assigned to you by your system administrator. You don't normally need to know your userid or groupid as the system translates username ↔ userid, and groupname ↔ groupid automatically. You probably already know your username; it's the name you logon with. The groupname is not as obvious, and indeed, you may belong to more than one group. Your primary group is the one associated with your username in the password database file, as set up by your system administrator. Similarly, there is a group database file where the system administrator can assign you rights to additional groups on the system.

In the examples below *%* is your shell prompt; you don't type this in.

You can determine your userid and the list of groups you belong to with the ***id*** and ***groups*** commands. On some systems ***id*** displays your user and primary group information, e.g.:

```
% id
    uid=1101(frank) gid=10(staff)
```

on other systems it also displays information for any additional groups you belong to:

```
% id
    uid=1101(frank) gid=10(staff) groups=10(staff),5(operator),14(sysadmin),110(uts)
```

The ***groups*** command displays the group information for all the groups you belong to, e.g.:

```
% groups
    staff sysadmin uts operator
```

3.2 Unix Command Line Structure

A **command** is a program that tells the Unix system to do something. It has the form:

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

where an **argument** indicates on what the command is to perform its action, usually a file or series of files. An option modifies the command, changing the way it performs.

Commands are case sensitive. ***command*** and ***Command*** are not the same.

Options are generally preceded by a hyphen (**-**), and for most commands, more than one option can be strung together, in the form:

command -[option][option][option]

e.g.:

ls -alR

will perform a long list on all files in the current directory and recursively perform the list through all sub-directories.

For most commands you can separate the options, preceding each with a hyphen, e.g.:

command -option1 -option2 -option3

as in:

ls -a -l -R

Some commands have options that require parameters. Options requiring parameters are usually specified separately, e.g.:

lpr -Pprinter3 -# 2 file

will send 2 copies of file to printer3.

These are the standard conventions for commands. However, not all Unix commands will follow the standard. Some don't require the hyphen before options and some won't let you group options together, i.e. they may require that each option be preceded by a hyphen and separated by whitespace from other options and arguments.

Options and syntax for a command are listed in the **man page** for the command.

3.3 Control Keys

Control keys are used to perform special functions on the command line or within an editor. You type these by holding down the **Control** key and some other **key** simultaneously. This is usually represented as **^K**ey. **Control-S** would be written as **^S**. With control keys upper and lower case are the same, so **^S** is the same as **^s**. This particular example is a **stop** signal and tells the terminal to stop accepting input. It will remain that way until you type a **start** signal, **^Q**.

Control-U is normally the "**line-kill**" signal for your terminal. When typed it erases the entire input line.

In the ***vi*** editor you can type a control key into your text file by first typing **^V** followed by the control character desired, so to type **^H** into a document type **^V^H**.

3.4 stty - terminal control

stty reports or sets terminal control options. The "**tty**" is an abbreviation that harks back to the days of teletypewriters, which were associated with transmission of telegraph messages, and which were models for early computer terminals.

For new users, the most important use of the ***stty*** command is setting the erase function to the appropriate key on their terminal. For systems programmers or shell script writers, the ***stty*** command provides an invaluable tool for configuring many aspects of I/O control for a given device, including the following:

- erase and line-kill characters
- data transmission speed
- parity checking on data transmission
- hardware flow control
- newline (NL) versus carriage return plus linefeed (CR-LF)

- interpreting tab characters
- edited versus raw input
- mapping of upper case to lower case

This command is very system specific, so consult the **man pages** for the details of the ***stty*** command on your system.

Syntax

stty [options]

Options

(none)	report the terminal settings
all (or **-a**)	report on all options
echoe	echo ERASE as BS-space-BS
dec	set modes suitable for Digital Equipment Corporation operating systems (which distinguishes between ERASE and BACKSPACE) (Not available on all systems)
kill	set the LINE-KILL character
erase	set the ERASE character
intr	set the INTERRUPT character

Examples

You can display and change your terminal control settings with the ***stty*** command. To display all (**-a**) of the current line settings:

```
% stty -a
    speed 38400 baud, 24 rows, 80 columns
    parenb -parodd cs7 -cstopb -hupcl cread -clocal -crtscts
    -ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr icrnl -iuclc
    ixon -ixany -ixoff imaxbel
    isig iexten icanon -xcase echo echoe echok -echonl -noflsh -tostop
    echoctl -echoprt echoke
    opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel
    erase  kill   werase rprnt  flush  lnext  susp   intr   quit   stop   eof
    ^H     ^U     ^W     ^R     ^O     ^V     ^Z/^Y  ^C     ^\     ^S/^Q  ^D
```

You can change settings using ***stty***, e.g., to change the erase character from **^?** (the delete key) to **^H**:

```
% stty erase ^H
```

This will set the terminal options for the current session only. To have this done for you automatically each time you login, it can be inserted into the **.login** or **.profile** file that we'll look at later.

3.5 Getting Help

The Unix manual, usually called **man pages**, is available on-line to explain the usage of the Unix system and commands. To use a man page, type the command "***man***" at the system prompt followed by the command for which you need information.

Syntax

man [options] command_name

Common Options

-k keyword	list command synopsis line for all keyword matches
-M path	path to man pages
-a	show all matching man pages (SVR4)

Examples

You can use ***man*** to provide a one line synopsis of any commands that contain the keyword that you want to search on with the "**-k**" option, e.g. to search on the keyword **password**, type:

```
% man -k password
   passwd (5)        - password file
   passwd (1)        - change password information
```

The number in parentheses indicates the section of the man pages where these references were found. You can then access the man page (by default it will give you the lower numbered entry, but you can use a command line option to specify a different one) with:

```
% man passwd
PASSWD(1)          USER COMMANDS          PASSWD(1)
NAME
   passwd    - change password  information
SYNOPSIS
   passwd [ -e login_shell ] [ username ]
DESCRIPTION
   passwd changes (or sets) a user's password.
   passwd prompts twice for the new password, without displaying
   it.  This is to allow for the possibility of typing mistakes.
   Only the user and the super-user can change the user's password.
OPTIONS
   -e   Change the user's login shell.
```

Here we've paraphrased and truncated the output for space and copyright concerns.

3.6 Directory Navigation and Control

The Unix file system is set up like a tree branching out from the root. The the **root** directory of the system is symbolized by the forward slash (/). System and user directories are organized under the **root**. The user does not have a root directory in Unix; users generally log into their own **home** directory. Users can then create other directories under their **home**. The following table summarizes some directory navigation commands.

TABLE 3.1 Navigation and Directory Control Commands

Command/Syntax	What it will do
cd [*directory*]	change directory
ls [options] [*directory* or *file*]	list *directory* contents or *file* permissions
mkdir [options] *directory*	make a *directory*
pwd	print working (current) directory
rmdir [options] *directory*	remove a *directory*

If you're familiar with DOS the following table comparing similar commands might help to provide the proper reference frame.

TABLE 3.2 Unix vs DOS Navigation and Directory Control Commands

Command	Unix	DOS
list directory contents	ls	dir
make directory	mkdir	md & mkdir
change directory	cd	cd & chdir
delete (remove) directory	rmdir	rd & rmdir
return to user's home directory	cd	cd\
location in path (present working directory)	pwd	cd

 © 1998 University Technology Services, The Ohio State University

3.6.1 pwd - print working directory

At any time you can determine where you are in the file system hierarchy with the ***pwd***, print working directory, command, e.g.:

```
% pwd
    /home/frank/src
```

3.6.2 cd - change directory

You can change to a new directory with the ***cd***, change directory, command. ***cd*** will accept both absolute and relative path names.

Syntax

cd [directory]

Examples

cd (also ***chdir*** in some shells)	change directory
cd	changes to user's home directory
cd /	changes directory to the system's root
cd ..	goes up one directory level
cd ../..	goes up two directory levels
cd /full/path/name/from/root	changes directory to absolute path named (note the leading slash)
cd path/from/current/location	changes directory to path relative to current location (no leading slash)
cd ~username/directory	changes directory to the named username's indicated directory (Note: the ~ is not valid in the Bourne shell; see Chapter 5.)

3.6.3 mkdir - make a directory

You extend your home hierarchy by making sub-directories underneath it. This is done with the ***mkdir***, make directory, command. Again, you specify either the full or relative path of the directory:

Syntax

mkdir [options] directory

Common Options

-p	create the intermediate (parent) directories, as needed
-m mode	access permissions (SVR4). (We'll look at modes later in this Chapter).

Examples

```
% mkdir /home/frank/data
```

or, if your present working directory is /home/frank the following would be equivalent:

```
% mkdir data
```

3.6.4 rmdir - remove directory

A directory needs to be empty before you can remove it. If it's not, you need to remove the files first. Also, you can't remove a directory if it is your present working directory; you must first change out of it.

Syntax

rmdir directory

Examples

To remove the empty directory /home/frank/data while in /home/frank use:

```
% rmdir data
```

or

```
% rmdir /home/frank/data
```

3.6.5 ls - list directory contents

The command to list your directories and files is ***ls***. With options it can provide information about the size, type of file, permissions, dates of file creation, change and access.

Syntax

ls [options] [argument]

Common Options

When no argument is used, the listing will be of the current directory. There are many very useful options for the ls command. A listing of many of them follows. When using the command, string the desired options together preceded by "**-**".

-a	lists all files, including those beginning with a dot (**.**).
-d	lists only names of directories, not the files in the directory
-F	indicates type of entry with a trailing symbol:
	directories /
	sockets =
	symbolic links @
	executables *
-g	displays Unix group assigned to the file, requires the -l option (BSD only)
	-or- on an SVR4 machine, e.g. Solaris, this option has the opposite effect
-L	if the file is a symbolic link, lists the information for the file or directory the link references, not the information for the link itself
-l	long listing: lists the mode, link information, owner, size, last modification (time). If the file is a symbolic link, an arrow (-->) precedes the pathname of the linked-to file.

The **mode field** is given by the **-l** option and consists of 10 characters. The first character is one of the following:

CHARACTER	IF ENTRY IS A
d	directory
-	plain file
b	block-type special file
c	character-type special file
l	symbolic link
s	socket

The next 9 characters are in 3 sets of 3 characters each. They indicate the **file access permissions**: the first 3 characters refer to the permissions for the **user**, the next three for the users in the Unix **group** assigned to the file, and the last 3 to the permissions for **other** users on the system. Designations are as follows:

r	read permission
w	write permission
x	execute permission
-	no permission

There are a few less commonly used permission designations for special circumstances. These are explained in the man page for ***ls***.

Examples

To list the files in a directory:

```
% ls
   demofiles       frank           linda
```

To list all files in a directory, including the hidden (dot) files try:

```
% ls -a
   .       .cshrc      .history    .plan       .rhosts         frank
   ..      .emacs      .login      .profile    demofiles       linda
```

To get a long listing:

```
% ls -al
   total 24
   drwxr-sr-x   5 workshop acs          512 Jun  7 11:12 .
   drwxr-xr-x   6 root     sys          512 May 29 09:59 ..
   -rwxr-xr-x   1 workshop acs          532 May 20 15:31 .cshrc
   -rw-------   1 workshop acs          525 May 20 21:29 .emacs
   -rw-------   1 workshop acs          622 May 24 12:13 .history
   -rwxr-xr-x   1 workshop acs          238 May 14 09:44 .login
   -rw-r--r--   1 workshop acs          273 May 22 23:53 .plan
   -rwxr-xr-x   1 workshop acs          413 May 14 09:36 .profile
   -rw-------   1 workshop acs           49 May 20 20:23 .rhosts
   drwx------   3 workshop acs          512 May 24 11:18 demofiles
   drwx------   2 workshop acs          512 May 21 10:48 frank
   drwx------   3 workshop acs          512 May 24 10:59 linda
```

 © 1998 University Technology Services, The Ohio State University

3.7 File Maintenance Commands

To create, copy, remove and change permissions on files you can use the following commands.

TABLE 3.3 File Maintenance Commands

Command/Syntax	What it will do
chgrp [options] *group file*	change the group of the file
chmod [options] *file*	change file or directory access permissions
chown [options] *owner file*	change the ownership of a file; can only be done by the superuser
cp [options] *file1 file2*	copy *file1* into *file2*; *file2* shouldn't already exist. This command creates or overwrites *file2*.
mv [options] *file1 file2*	move *file1* into *file2*
rm [options] *file*	remove (delete) a file or directory (**-r** recursively deletes the directory and its contents) (**-i** prompts before removing files)

If you're familiar with DOS the following table comparing similar commands might help to provide the proper reference frame.

TABLE 3.4 Unix vs DOS File Maintenance Commands

Command	Unix	DOS
copy file	cp	copy
move file	mv	move (not supported on all versions of DOS)
rename file	mv	rename & ren
delete (remove) file	rm	erase & del
display file to screen		
entire file	cat	type
one page at a time	more, less, pg	type/p (not supported on all versions of DOS)

 © 1998 University Technology Services, The Ohio State University

3.7.1 cp - copy a file

Copy the contents of one file to another with the ***cp*** command.

Syntax

cp [options] old_filename new_filename

Common Options

-i	interactive (prompt and wait for confirmation before proceeding)
-r	recursively copy a directory

Examples

```
% cp old_filename new_filename
```

You now have two copies of the file, each with identical contents. They are completely independent of each other and you can edit and modify either as needed. They each have their own inode, data blocks, and directory table entries.

3.7.2 mv - move a file

Rename a file with the move command, ***mv***.

Syntax

mv [options] old_filename new_filename

Common Options

-i	interactive (prompt and wait for confirmation before proceeding)
-f	don't prompt, even when copying over an existing target file (overrides **-i**)

Examples

```
% mv old_filename new_filename
```

You now have a file called **new_filename** and the file **old_filename** is gone. Actually all you've done is to update the directory table entry to give the file a new name. The contents of the file remain where they were.

3.7.3 rm - remove a file

Remove a file with the ***rm***, remove, command.

Syntax

rm [options] filename

Common Options

-i	interactive (prompt and wait for confirmation before proceeding)
-r	recursively remove a directory, first removing the files and subdirectories beneath it
-f	don't prompt for confirmation (overrides -i)

Examples

% rm old_filename

A listing of the directory will now show that the file no longer exists. Actually, all you've done is to remove the directory table entry and mark the inode as unused. The file contents are still on the disk, but the system now has no way of identifying those data blocks with a file name. There is no command to "**unremove**" a file that has been removed in this way. For this reason many novice users alias their remove command to be "***rm -i***", where the **-i** option prompts them to answer yes or no before the file is removed. Such aliases are normally placed in the **.cshrc** file for the C shell; see Chapter 5)

3.7.4 File Permissions

Each file, directory, and executable has permissions set for who can **read**, write, and/or **execute** it. To find the permissions assigned to a file, the ***ls*** command with the **-l** option should be used. Also, using the **-g** option with "***ls -l***" will help when it is necessary to know the group for which the permissions are set (BSD only).

When using the "***ls -lg***" command on a file (***ls -l*** on SysV), the output will appear as follows:

-rwxr-x--- user unixgroup size Month nn hh:mm filename

The area above designated by letters and dashes (**-rwxr-x---**) is the area showing the file type and permissions as defined in the previous Section. Therefore, a permission string, for example, of **-rwxr-x---** allows the **user** (owner) of the file to read, write, and execute it; those in the **unixgroup** of the file can read and execute it; **others** cannot access it at all.

3.7.5 chmod - change file permissions

The command to change permissions on an item (file, directory, etc) is ***chmod*** (change mode). The syntax involves using the command with three digits (representing the **user** (owner, **u**) permissions, the **group** (**g**) permissions, and **other** (**o**) user's permissions) followed by the argument (which may be a file name or list of files and directories). Or by using symbolic representation for the permissions and who they apply to.

Each of the permission types is represented by either a numeric equivalent:

read=4, write=2, execute=1

or a single letter:

read=r, write=w, execute=x

A permission of **4** or **r** would specify **read** permissions. If the permissions desired are read and write, the 4 (representing read) and the 2 (representing write) are added together to make a permission of 6. Therefore, a permission setting of 6 would allow read and write permissions.

Alternatively, you could use symbolic notation which uses the one letter representation for who and for the permissions and an operator, where the operator can be:

+	add permissions
-	remove permissions
=	set permissions

So to set read and write for the owner we could use "**u=rw**" in symbolic notation.

Syntax

chmod nnn [argument list]	numeric mode
chmod [who]op[perm] [argument list]	symbolic mode

where **nnn** are the three numbers representing **user**, **group**, and **other** permissions, **who** is any of **u**, **g**, **o**, or **a** (all) and **perm** is any of **r**, **w**, **x**. In symbolic notation you can separate permission specifications by commas, as shown in the example below.

Common Options

-f	force (no error message is generated if the change is unsuccessful)
-R	recursively descend through the directory structure and change the modes

Examples

If the permission desired for file1 is **user**: read, write, execute, **group**: read, execute, **other**: read, execute, the command to use would be

chmod 755 file1 or ***chmod u=rwx,go=rx file1***

Reminder: When giving permissions to **group** and **other** to use a file, it is necessary to allow at least execute permission to the directories for the path in which the file is located. The easiest way to do this is to be in the directory for which permissions need to be granted:

chmod 711 . or ***chmod u=rw,+x .*** or ***chmod u=rwx,go=x .***

where the dot (**.**) indicates **this directory**.

3.7.6 chown - change ownership

Ownership of a file can be changed with the ***chown*** command. On most versions of Unix this can only be done by the super-user, i.e. a normal user can't give away ownership of their files. ***chown*** is used as below, where **#** represents the shell prompt for the super-user:

Syntax

chown [options] user[:group] file (SVR4)

chown [options] user[.group] file (BSD)

Common Options

-R	recursively descend through the directory structure
-f	force, and don't report any errors

Examples

```
# chown new_owner file
```

3.7.7 chgrp - change group

Anyone can change the group of files they own, to another group they belong to, with the ***chgrp*** command.

Syntax

chgrp [options] group file

Common Options

-R	recursively descend through the directory structure
-f	force, and don't report any errors

Examples

```
% chgrp new_group file
```

3.8 Display Commands

There are a number of commands you can use to **display** or **view** a file. Some of these are editors which we will look at later. Here we will illustrate some of the commands normally used to display a file.

TABLE 3.5 Display Commands

Command/Syntax	What it will do
cat [options] *file*	concatenate (list) a file
echo [text string]	echo the text string to stdout
head [-number] *file*	display the first 10 (or number of) lines of a file
more (or ***less*** or ***pg***) [options] *file*	page through a text file
tail [options] *file*	display the last few lines (or parts) of a file

3.8.1 echo - echo a statement

The ***echo*** command is used to repeat, or echo, the argument you give it back to the standard output device. It normally ends with a line-feed, but you can specify an option to prevent this.

Syntax

echo [string]

Common Options

-n	don't print <new-line> (BSD, shell built-in)
\c	don't print <new-line> (SVR4)
\0n	where **n** is the 8-bit ASCII character code (SVR4)
\t	tab (SVR4)
\f	form-feed (SVR4)
\n	new-line (SVR4)
\v	vertical tab (SVR4)

Examples

```
% echo Hello Class        or        echo "Hello Class"
```

To prevent the line feed:

```
% echo -n Hello Class     or        echo "Hello Class \c"
```

where the style to use in the last example depends on the ***echo*** command in use.

The \x options must be within pairs of single or double quotes, with or without other string characters.

3.8.2 cat - concatenate a file

Display the contents of a file with the concatenate command, ***cat***.

Syntax

cat [options] [file]

Common Options

-n	precede each line with a line number
-v	display non-printing characters, except tabs, new-lines, and form-feeds
-e	display $ at the end of each line (prior to new-line) (when used with **-v** option)

Examples

% cat filename

You can list a series of files on the command line, and ***cat*** will concatenate them, starting each in turn, immediately after completing the previous one, e.g.:

% cat file1 file2 file3

3.8.3 more, less, and pg - page through a file

more, ***less***, and ***pg*** let you page through the contents of a file one screenful at a time. These may not all be available on your Unix system. They allow you to back up through the previous pages and search for words, etc.

Syntax

more [options] [+/pattern] [filename]

less [options] [+/pattern] [filename]

pg [options] [+/pattern] [filename]

Options

more	**less**	**pg**	**Action**
-c	**-c**	**-c**	clear display before displaying
	-i		ignore case
-w	default	default	don't exit at end of input, but prompt and wait
-lines		**-lines**	# of lines/screenful
+/pattern	**+/pattern**	**+/pattern**	search for the pattern

 © 1998 University Technology Services, The Ohio State University

Internal Controls

more	displays (one screen at a time) the file requested
<space bar>	to view next screen
<return> or ***<CR>***	to view one more line
q	to quit viewing the file
h	help
b	go back up one screenful
/word	search for **word** in the remainder of the file
	See the **man page** for additional options
less	similar to ***more***; see the **man page** for options
pg	the SVR4 equivalent of ***more*** (page)

3.8.4 head - display the start of a file

head displays the head, or start, of the file.

Syntax

head [options] file

Common Options

-n number	number of lines to display, counting from the top of the file
-number	same as above

Examples

By default ***head*** displays the first 10 lines. You can display more with the "**-n numbe**r", or "**-number**" options, e.g., to display the first 40 lines:

% head -40 filename or head -n 40 filename

3.8.5 tail - display the end of a file

tail displays the tail, or end, of the file.

Syntax

tail [options] file

Common Options

-number	number of lines to display, counting from the bottom of the file

Examples

The default is to display the last 10 lines, but you can specify different line or byte numbers, or a different starting point within the file. To display the last 30 lines of a file use the **-number** style:

% tail -30 filename

CHAPTER 4

System Resources & Printing

4.1 System Resources

Commands to report or manage system resources.

TABLE 4.1 System Resource Commands

Command/Syntax	What it will do
chsh (***passwd -e/-s***) *username login_shell*	change the user's login shell (often only by the superuser)
date [options]	report the current date and time
df [options] [resource]	report the summary of disk blocks and inodes free and in use
du [options] [*directory* or *file*]	report amount of disk space in use+
hostname/uname	display or set (super-user only) the name of the current machine
kill [options] [-SIGNAL] [pid#] [%job]	send a signal to the process with the process id number (pid#) or job control number (%n). The default signal is to kill the process.
man [options] *command*	show the manual (**man**) page for a command
passwd [options]	set or change your password
ps [options]	show status of active processes
script file	saves everything that appears on the screen to file until ***exit*** is executed
stty [options]	set or display terminal control options
whereis [options] *command*	report the binary, source, and man page locations for the command named
which *command*	reports the path to the command or the shell alias in use
who or ***w***	report who is logged in and what processes are running

 © 1998 University Technology Services, The Ohio State University

4.1.1 df - summarize disk block and file usage

df is used to report the number of disk blocks and inodes used and free for each file system. The output format and valid options are very specific to the OS and program version in use.

Syntax

df [options] [resource]

Common Options

-l	local file systems only (SVR4)
-k	report in kilobytes (SVR4)

Examples

```
{unix prompt 1} df
   Filesystem            kbytes    used   avail capacity  Mounted on
   /dev/sd0a              20895   19224       0   102%    /
   /dev/sd0h             319055  131293  155857    46%    /usr
   /dev/sd1g             637726  348809  225145    61%    /usr/local
   /dev/sd1a             240111  165489   50611    77%
   /home/guardian
   peri:/usr/local/backup
                        1952573  976558  780758    56%
   /usr/local/backup
   peri:/home/peri        726884  391189  263007    60%    /home/peri
   peri:/usr/spool/mail   192383    1081  172064     1%
   /var/spool/mail
   peri:/acs/peri/2       723934  521604  129937    80%    /acs/peri/2
```

4.1.2 du - report disk space in use

du reports the amount of disk space in use for the files or directories you specify.

Syntax

du [options] [directory or file]

Common Options

-a	display disk usage for each file, not just subdirectories
-s	display a summary total only
-k	report in kilobytes (SVR4)

 © 1998 University Technology Services, The Ohio State University

Examples

```
{unix prompt 3} du
    1       ./.elm
    1       ./Mail
    1       ./News
    20      ./uc
    86      .

{unix prompt 4} du -a uc
    7       uc/unixgrep.txt
    5       uc/editors.txt
    1       uc/.emacs
    1       uc/.exrc
    4       uc/telnet.ftp
    1       uc/uniq.tee.txt
    20      uc
```

4.1.3 ps - show status of active processes

ps is used to report on processes currently running on the system. The output format and valid options are very specific to the OS and program version in use.

Syntax

ps [options]

Common Options

BSD	SVR4	
-a	**-e**	all processes, all users
-e		environment/everything
-g		process group leaders as well
-l	**-l**	long format
-u	**-u user**	user oriented report
-x	**-e**	even processes not executed from terminals
	-f	full listing
-w		report first 132 characters per line

note -- Because the ***ps*** command is highly system-specific, it is recommended that you consult the **man pages** of your system for details of options and interpretation of ***ps*** output.

Examples

```
{unix prompt 5} ps
     PID TT STAT  TIME COMMAND
   15549 p0 IW    0:00 -tcsh (tcsh)
   15588 p0 IW    0:00 man nice
   15594 p0 IW    0:00 sh -c less /tmp/man15588
   15595 p0 IW    0:00 less /tmp/man15588
   15486 p1 S     0:00 -tcsh (tcsh)
   15599 p1 T     0:00 emacs unixgrep.txt
   15600 p1 R     0:00 ps
```

4.1.4 kill - terminate a process

kill sends a signal to a process, usually to terminate it.

Syntax

kill [-signal] process-id

Common Options

-l displays the available kill signals:

Examples

{unix prompt 9} kill -l

HUP INT QUIT ILL TRAP IOT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM URG STOP TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH LOST USR1 USR2

The **-KILL** signal, also specified as **-9** (because it is 9th on the above list), is the most commonly used ***kill*** signal. Once seen, it can't be ignored by the program whereas the other signals can.

{unix prompt 10} kill -9 15599

[1] + Killed emacs unixgrep.txt

4.1.5 who - list current users

who reports who is logged in at the present time.

Syntax

who [am i]

Examples

```
beauty condron>who
   wmtell   ttyp1   Apr 21 20:15   (apple.acs.ohio-s)
   fbwalk   ttyp2   Apr 21 23:21   (worf.acs.ohio-st)
   stwang   ttyp3   Apr 21 23:22   (127.99.25.8)
   david    ttyp4   Apr 21 22:27   (slip1-61.acs.ohi)
   tgardner ttyp5   Apr 21 23:07   (picard.acs.ohio-)
   awallace ttyp6   Apr 21 23:00   (ts31-4.homenet.o)
   gtl27    ttyp7   Apr 21 23:24   (data.acs.ohio-st)
   ccchang  ttyp8   Apr 21 23:32   (slip3-10.acs.ohi)
   condron  ttypc   Apr 21 23:38   (lcondron-mac.acs)
   dgildman ttype   Apr 21 22:30   (slip3-36.acs.ohi)
   fcbetz   ttyq2   Apr 21 21:12   (ts24-10.homenet.)

beauty condron>who am i
   beauty!condron  ttypc   Apr 21 23:38    (lcondron-mac.acs)
```

4.1.6 whereis - report program locations

whereis reports the filenames of source, binary, and manual page files associated with command(s).

Syntax

whereis [options] command(s)

Common Options

-b	report binary files only
-m	report manual sections only
-s	report source files only

Examples

brigadier: condron [69]> whereis Mail
Mail: /usr/ucb/Mail /usr/lib/Mail.help /usr/lib/Mail.rc /usr/man/man1/Mail.1

```
brigadier: condron [70]> whereis -b Mail
    Mail: /usr/ucb/Mail /usr/lib/Mail.help /usr/lib/Mail.rc

brigadier: condron [71]> whereis -m Mail
    Mail: /usr/man/man1/Mail.1
```

4.1.7 which - report the command found

which will report the name of the file that is be executed when the command is invoked. This will be the full path name or the alias that's found first in your path.

Syntax

which command(s)

example--

```
brigadier: condron [73]> which Mail
    /usr/ucb/Mail
```

4.1.8 hostname/uname - name of machine

hostname (***uname -n*** on SysV) reports the host name of the machine the user is logged into, e.g.:

```
brigadier: condron [91]> hostname
    brigadier
```

uname has additional options to print information about system hardware type and software version.

4.1.9 script - record your screen I/O

script creates a script of your session input and output. Using the ***script*** command, you can capture all the data transmission from and to your terminal screen until you ***exit*** the script program. This can be useful during the programming-and-debugging process, to document the combination of things you have tried, or to get a printed copy of it all for later perusal.

Syntax

script [-a] [file] <. . .> exit

Common Options

-a	append the output to file

typescript is the name of the default file used by ***script***.

You must remember to type ***exit*** to end your script session and close your typescript file.

Examples

```
beauty condron>script
   Script started, file is typescript
beauty condron>ps
     PID TT STAT   TIME COMMAND
   23323 p8 S      0:00 -h -i (tcsh)
   23327 p8 R      0:00 ps
   18706 pa S      0:00 -tcsh (tcsh)
   23315 pa T      0:00 emacs
   23321 pa S      0:00 script
   23322 pa S      0:00 script
    3400 pb I      0:00 -tcsh (tcsh)
beauty condron>kill -9 23315
beauty condron>date
   Mon Apr 22 22:29:44 EDT 1996
beauty condron>exit
   exit
   Script done, file is typescript
   [1]  + Killed                 emacs

beauty condron>cat typescript
   Script started on Mon Apr 22 22:28:36 1996
   beauty condron>ps
     PID TT STAT   TIME COMMAND
   23323 p8 S      0:00 -h -i (tcsh)
   23327 p8 R      0:00 ps
   18706 pa S      0:00 -tcsh (tcsh)
   23315 pa T      0:00 emacs
   23321 pa S      0:00 script
   23322 pa S      0:00 script
    3400 pb I      0:00 -tcsh (tcsh)
   beauty condron>kill -9 23315
   beauty condron>date
   Mon Apr 22 22:29:44 EDT 1996
   beauty condron>exit
   exit

   script done on Mon Apr 22 22:30:02 1996
   beauty condron>
```

4.1.10 date - current date and time

date displays the current data and time. A superuser can set the date and time.

Syntax

date [options] [+format]

Common Options

-u	use Universal Time (or Greenwich Mean Time)
+format	specify the output format
%a	weekday abbreviation, Sun to Sat
%h	month abbreviation, Jan to Dec
%j	day of year, 001 to 366
%n	<new-line>
%t	<TAB>
%y	last 2 digits of year, 00 to 99
%D	MM/DD/YY date
%H	hour, 00 to 23
%M	minute, 00 to 59
%S	second, 00 to 59
%T	HH:MM:SS time

Examples

```
beauty condron>date
    Mon Jun 10 09:01:05 EDT 1996

beauty condron>date -u
    Mon Jun 10 13:01:33 GMT 1996

beauty condron>date +%a%t%D
    Mon             06/10/96

beauty condron>date '+%y:%j'
    96:162
```

4.2 Print Commands

TABLE 4.2 Printing Commands

Command/Syntax	What it will do
lpq (***lpstat***) [options]	show the status of print jobs
lpr (***lp***) [options] *file*	print to defined printer
lprm (***cancel***) [options]	remove a print job from the print queue
pr [options] *[file]*	filter the file and print it on the terminal

The print commands allow us to print files to standard output (***pr***) or to a line printer (***lp/lpr***) while filtering the output. The **BSD** and **SysV** printer commands use different names and different options to produce the same results: ***lpr***, ***lprm***, and ***lpq*** vs ***lp***, ***cancel***, and ***lpstat*** for the BSD and SysV submit, cancel, and check the status of a print job, respectively.

4.2.1 lp/lpr - submit a print job

lp and ***lpr*** submit the specified file, or standard input, to the printer daemon to be printed. Each job is given a unique request-id that can be used to follow or cancel the job while it's in the queue.

Syntax

lp [options] filename
lpr [options] filename

Common Options

lp	lpr	function
-n number	**-#**number	number of copies
-t title	**-T**title	title for job
-d destination	**-P**printer	printer name
-c	(default)	copy file to queue before printing
(default)	**-s**	don't copy file to queue before printing
-o option		additional options, e.g. nobanner

Files beginning with the string "**%!**" are assumed to contain PostScript commands.

Examples

To print the file ssh.ps:

```
% lp ssh.ps
request id is lp-153 (1 file(s))
```

This submits the job to the queue for the default printer, **lp**, with the request-id lp-153.

4.2.2 lpstat/lpq - check the status of a print job

You can check the status of your print job with lpstat or lpq.

Syntax

lpstat [options]

lpq [options] [job#] [username]

Common Options

lpstat	lpq	function
-d	(defaults to lp)	list system default destination
-s		summarize print status
-t		print all status information
-u [login-ID-list]		user list
-v		list printers known to the system
-p printer_dest	**-P**printer_dest	list status of printer, printer_dest

Examples

```
% lpstat
   lp-153        frank       208068     Apr 29 15:14 on lp
```

4.2.3 cancel/lprm - cancel a print job

Any user can cancel only heir own print jobs.

Syntax

cancel [request-ID] [printer]

lprm [options] [job#] [username]

Common Options

cancel	lprm	function
	-Pprinter	specify printer
	-	all jobs for user
-u [login-ID-list]		user list

Examples

To cancel the job submitted above:

```
% cancel lp-153
```

 © 1998 University Technology Services, The Ohio State University

4.2.4 pr - prepare files for printing

pr prints header and trailer information surrounding the formatted file. You can specify the number of pages, lines per page, columns, line spacing, page width, etc. to print, along with header and trailer information and how to treat **<tab>** characters.

Syntax

pr [options] file

Common Options

+page_number	start printing with page page_number of the formatted input file
-column	number of columns
-a	modify -column option to fill columns in round-robin order
-d	double spacing
-e[char][gap]	tab spacing
-h header_string	header for each page
-l lines	lines per page
-t	don't print the header and trailer on each page
-w width	width of page

Examples

The file containing the list of P. G. Wodehouse's Lord Emsworth books could be printed, at 14 lines per page (including 5 header and 5 (empty) trailer lines) below, where the **-e** option specifies the **<tab>** conversion style:

```
% pr -l 14 -e42 wodehouse


Apr 29 11:11 1996  wodehouse_emsworth_books Page 1


Something Fresh [1915]                  Uncle Dynamite [1948]
Leave it to Psmith [1923]               Pigs Have Wings [1952]
Summer Lightning [1929]                 Cocktail Time [1958]
Heavy Weather [1933]                    Service with a Smile [1961]
```

```
Apr 29 11:11 1996  wodehouse_emsworth_books Page 2


Blandings Castle and Elsewhere [1935]     Galahad at Blandings [1965]
Uncle Fred in the Springtime [1939]       A Pelican at Blandings [1969]
Full Moon [1947]                          Sunset at Blandings [1977]
```

CHAPTER 5 Shells

The shell sits between you and the operating system, acting as a command interpreter. It reads your terminal input and translates the commands into actions taken by the system. The shell is analogous to ***command.com*** in DOS. When you log into the system you are given a default shell. When the shell starts up it reads its startup files and may set environment variables, command search paths, and command aliases, and executes any commands specified in these files.

The original shell was the Bourne shell, ***sh***. Every Unix platform will either have the Bourne shell, or a Bourne compatible shell available. It has very good features for controlling input and output, but is not well suited for the interactive user. To meet the latter need the C shell, ***csh***, was written and is now found on most, but not all, Unix systems. It uses C type syntax, the language Unix is written in, but has a more awkward input/output implementation. It has job control, so that you can reattach a job running in the background to the foreground. It also provides a history feature which allows you to modify and repeat previously executed commands.

The default prompt for the Bourne shell is **$** (or **#**, for the root user). The default prompt for the C shell is **%**.

Numerous other shells are available from the network. Almost all of them are based on either ***sh*** or ***csh*** with extensions to provide job control to ***sh***, allow in-line editing of commands, page through previously executed commands, provide command name completion and custom prompt, etc. Some of the more well known of these may be on your favorite Unix system: the Korn shell, ***ksh***, by David Korn and the Bourne Again SHell, ***bash***, from the Free Software Foundations GNU project, both based on ***sh***, the T-C shell, ***tcsh***, and the extended C shell, ***cshe***, both based on ***csh***. Below we will describe some of the features of ***sh*** and ***csh*** so that you can get started.

5.1 Built-in Commands

The shells have a number of **built-in**, or native commands. These commands are executed directly in the shell and don't have to call another program to be run. These built-in commands are different for the different shells.

5.1.1 Sh

For the Bourne shell some of the more commonly used built-in commands are:

:	null command
.	source (read and execute) commands from a file
case	case conditional loop
cd	change the working directory (default is $HOME)
echo	write a string to standard output
eval	evaluate the given arguments and feed the result back to the shell
exec	execute the given command, replacing the current shell
exit	exit the current shell
export	share the specified environment variable with subsequent shells
for	for conditional loop
if	if conditional loop
pwd	print the current working directory
read	read a line of input from stdin
set	set variables for the shell
test	evaluate an expression as true or false
trap	trap for a typed signal and execute commands
umask	set a default file permission mask for new files
unset	unset shell variables
wait	wait for a specified process to terminate
while	while conditional loop

 © 1998 University Technology Services, The Ohio State University

5.1.2 Csh

For the C shell the more commonly used built-in functions are:

alias	assign a name to a function
bg	put a job into the background
cd	change the current working directory
echo	write a string to stdout
eval	evaluate the given arguments and feed the result back to the shell
exec	execute the given command, replacing the current shell
exit	exit the current shell
fg	bring a job to the foreground
foreach	for conditional loop
glob	do filename expansion on the list, but no "\" escapes are honored
history	print the command history of the shell
if	if conditional loop
jobs	list or control active jobs
kill	kill the specified process
limit	set limits on system resources
logout	terminate the login shell
nice ***command***	lower the scheduling priority of the process, ***command***
nohup ***command***	do not terminate ***command*** when the shell exits
popd	pop the directory stack and return to that directory
pushd	change to the new directory specified and add the current one to the directory stack
rehash	recreate the hash table of paths to executable files
repeat	repeat a command the specified number of times
set	set a shell variable
setenv	set an environment variable for this and subsequent shells
source	source (read and execute) commands from a file
stop	stop the specified background job
switch	switch conditional loop
umask	set a default file permission mask for new files
unalias	remove the specified alias name
unset	unset shell variables
unsetenv	unset shell environment variables
wait	wait for all background processes to terminate
while	while conditional loop

5.2 Environment Variables

Environmental variables are used to provide information to the programs you use. You can have both **global environment** and **local shell variables**. Global environment variables are set by your login shell and new programs and shells inherit the environment of their parent shell. Local shell variables are used only by that shell and are not passed on to other processes. A child process cannot pass a variable back to its parent process.

The current environment variables are displayed with the "***env***" or "***printenv***" commands. Some common ones are:

- **DISPLAY** The graphical display to use, e.g. nyssa:0.0
- **EDITOR** The path to your default editor, e.g. /usr/bin/vi
- **GROUP** Your login group, e.g. staff
- **HOME** Path to your home directory, e.g. /home/frank
- **HOST** The hostname of your system, e.g. nyssa
- **IFS** Internal field separators, usually any white space (defaults to tab, space and <newline>)
- **LOGNAME** The name you login with, e.g. frank
- **PATH** Paths to be searched for commands, e.g. /usr/bin:/usr/ucb:/usr/local/bin
- **PS1** The primary prompt string, Bourne shell only (defaults to $)
- **PS2** The secondary prompt string, Bourne shell only (defaults to >)
- **SHELL** The login shell you're using, e.g. /usr/bin/csh
- **TERM** Your terminal type, e.g. xterm
- **USER** Your username, e.g. frank

Many environment variables will be set automatically when you login. You can modify them or define others with entries in your startup files or at anytime within the shell. Some variables you might want to change are **PATH** and **DISPLAY**. The **PATH** variable specifies the directories to be automatically searched for the command you specify. Examples of this are in the shell startup scripts below.

You set a **global environment variable** with a command similar to the following for the C shell:

```
% setenv NAME value
```

and for Bourne shell:

```
$ NAME=value; export NAME
```

You can list your global environmental variables with the ***env*** or ***printenv*** commands. You unset them with the ***unsetenv*** (C shell) or ***unset*** (Bourne shell) commands.

To set a **local shell variable** use the ***set*** command with the syntax below for C shell. Without options ***set*** displays all the local variables.

```
% set name=value
```

For the Bourne shell set the variable with the syntax:

```
$ name=value
```

The current value of the variable is accessed via the "**$name**", or "**${name}**", notation.

5.3 The Bourne Shell, sh

Sh uses the startup file **.profile** in your home directory. There may also be a system-wide startup file, e.g. **/etc/profile**. If so, the system-wide one will be sourced (executed) before your local one.

A simple **.profile** could be the following:

```
PATH=/usr/bin:/usr/ucb:/usr/local/bin:.        # set the PATH
export PATH                                    # so that PATH is available to subshells
# Set a prompt
PS1="{`hostname` `whoami`} "                   # set the prompt,  default is "$"
# functions
ls() { /bin/ls -sbF "$@";}
ll() { ls -al "$@";}
# Set the terminal type
stty erase ^H                                  # set Control-H to be the erase key
eval `tset -Q -s -m ':?xterm'`                 # prompt for the terminal type, assume xterm
#
umask 077
```

Whenever a **#** symbol is encountered the remainder of that line is treated as a comment. In the **PATH** variable each directory is separated by a colon (**:**) and the dot (**.**) specifies that the current directory is in your path. If the latter is not set it's a simple matter to execute a program in the current directory by typing:

./program_name

It's actually a good idea not to have dot (**.**) in your path, as you may inadvertently execute a program you didn't intend to when you ***cd*** to different directories.

A variable set in **.profile** is set only in the login shell unless you "***export***" it or **source .profile** from another shell. In the above example **PATH** is exported to any subshells. You can source a file with the built-in "**.**" command of ***sh***, i.e.:

. ./.profile

You can make your own functions. In the above example the function ***ll*** results in an "l***s -al***" being done on the specified files or directories.

With ***stty*** the erase character is set to Control-H (^H), which is usually the Backspace key.

The ***tset*** command prompts for the terminal type, and assumes "**xterm**" if we just hit <CR>. This command is run with the shell built-in, ***eval***, which takes the result from the tset command and uses it as an argument for the shell. In this case the "**-s**" option to tset sets the **TERM** and **TERMCAP** variables and exports them.

The last line in the example runs the ***umask*** command with the option such that any files or directories you create will not have read/write/execute permission for **group** and **other**.

For further information about ***sh*** type "***man sh***" at the shell prompt.

5.4 The C Shell, csh

Csh uses the startup files **.cshrc** and **.login**. Some versions use a system-wide startup file, e.g. **/etc/csh.login**. Your **.login** file is sourced (executed) only when you login. Your **.cshrc** file is sourced every time you start a ***csh***, including when you login. It has many similar features to **.profile**, but a different style of doing things. Here we use the ***set*** or ***setenv*** commands to initialize a variable, where ***set*** is used for this shell and ***setenv*** for this and any subshells. The environment variables: **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **user**, **term**, and **path** variables of the ***csh***. So ***setenv*** doesn't need to be done for these. The C shell uses the symbol, **~**, to indicate the user's home directory in a path, as in **~/.cshrc**, or to specify another user's login directory, as in **~username/.cshrc**.

Predefined variables used by the C shell include:

- **argv** The list of arguments of the current shell
- **cwd** The current working directory
- **history** Sets the size of the history list to save
- **home** The home directory of the user; starts with $HOME
- **ignoreeof** When set ignore EOF (^D) from terminals
-

 © 1998 University Technology Services, The Ohio State University