Basic Introduction to UNIX/linux

Claude Cantin (claude.cantin@nrc.ca)
http://www.nrc.ca/imsb/rcsg

Research Computing Support Group
Information Management Services Branch
National Research Council

April 16, 2006

This page intentionally left blank.

Claude Cantin
National Research Council of Canada

History of printing:

Date	Copies
March 2003	200
March 2001	200
June 1999	200
November 1997	200
July 1996	200
November 1995	150
March 1995	150
February 1994	150
October 1993	100
August 1993	75
February 1993	75
November 1992	35
September 1992	40
February 1992	50
December 1991	50
April 1991	50
September 1990	40
January 1990	40

Table 1: Printings.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This course is intended for people not familiar with the UNIX/linux operating system, but familiar with other computer systems such as MS Windows, DOS or VMS. It is meant as an introduction for beginners to help them understand concepts behind the UNIX/linux operating system. Intermediate users may find the course useful as a refresher.

Up to 2003, most of the command examples used throughout the text were performed using a Silicon Graphics O^2, running IRIX 6.5. But since September 2003, the hands-on portion of the course is done using the linux (SuSE 8.2, then 9.0) operating system, which means most commands are now done with linux. SGI systems running IRIX, Sun systems running Solaris, Hewlett-Packards running HP/UX, IBM RS/6000s running AIX and most PCs (and other architectures) running linux use most of the commands described in this manual. They use the same basic commands, although some of the options used may vary slightly between the different architectures.

In specific cases, the book uses commands based on linux. The distribution used was SuSE version 7.3 and newer.

NOTES:

- This book refers to various UNIX derivatives running on "workstations".

  The author's definition of "workstation" includes systems such as the Sun Microsystems SPARCstation family, the Silicon Graphics Personal IRIS, Indigo, Indigo^2, Power Series, Challenge, Power Challenge, Onyx, Power Onyx, Indy, O^2, Octane, Origin and Altix families, the IBM

RS/6000 series, the HP 9000 model 700 and 800 families, the Compaq AXP families (systems running True UNIX), and 500+ MHz PCs running one of the linux distributions.

- Although most sections refer to UNIX in general, some refer to a specific architecture. Others may refer to NRC-specific topics. Those sections are generally clearly indicated.

- This book also refers to various linux distributions, notably SuSE 7.x and 8.x, and Red Hat 7.x and 8.x.

1.1 History

The UNIX story begins in 1969, when AT&T Bell Labs dropped out of a joint project between themselves, General Electric, and the Massachusetts Institute of Technology. Multics, the project in question, was an experimental operating system on the GE 645 (thanks to Tom Van Vleck from http://www.multicians.org for correcting details from pre-2002 versions of the course notes).

Ken Thompson and Dennis Ritchie, both from Bell Labs, had been exposed to the Multics project. They ported a game, called 'Space Travel', from the GE 645 running Multics, onto a PDP-7. To help them do the porting, Thompson wrote a "simple file system and some utilities for the PDP-7" [8, p. 3]. This was the birth of UNIX, in 1970 ("UNIX" was meant to be a pun on "Multics").

Late in 1970, a PDP-11/20 was purchased, and UNIX became an official Bell Labs project. The first UNIX edition "was documented in a manual authored by Thompson and Ritchie dated November 1971" [1, p. 2]. Most of the ideas found in today's UNIX systems were incorporated in this edition.

The second edition, 1972, incorporated the *piping* concept.

In 1973, UNIX was re-written in C, by Thompson and Ritchie. Note that Ritchie developed the *C* language (derived from the *B* language by Thompson) at approximately the same time.

UNIX was first distributed in May 1975 for a nominal fee. This was UNIX Version 6.

In 1979, a more portable version of UNIX (Version 7) was released for general use; from here, three major UNIX versions emerged: BSD (Berkeley System Distribution), XENIX, and AT&T's System V.

1.1.1 BSD: Berkeley System Distribution

The University of California at Berkeley acquired Version 4 of UNIX in 1974. In 1975, during a sabbatical leave from Bell Labs, Ken Thompson went to Berkeley and helped install UNIX Version 6 on a PDP-11/70. "The same year, two graduate students also arrived at Berkeley: Bill Joy and Chuck Haley" [8, p. 5] : they were to play an important role in the BSD developments.

Joy wrote the **vi** editor, and the **C shell**. Bill Joy is also one of the co-founders of Sun Microsystems.

In 1978, 2BSD was released.

In 1979, a combination of improved 2BSD and UNIX Version 7 was released as 3BSD.

DARPA (the Defence Advanced Research Project Agency) funded the development of 4BSD and 4.1BSD. The Fast File System was included in 4.2BSD which was released in 1983.

Finally, 4.3BSD was released in 1987.

Berkeley's development of its BSD UNIX ended with the release of its last major version, 4.4BSD, in 1992. Then, late in 1994, it released 4.4BSD*lite*.

Many UNIX variants are based on that last version, written to run better on PC's based on the Intel '486 and Pentium CPUs, namely `FreeBSD`, `NetBSD`, `BSD/OS` and `OpenBSD`. New versions of these versions became available in 1995.

1.1.2 XENIX

XENIX is directly based on UNIX Version 7. Its first version, 2.3, was released in 1980 by Microsoft for use on microcomputers (IBM PCs or clones).

XENIX 3.0, released in 1983, included new features from 4.1BSD and from AT&T's System III.

Finally, the latest version, XENIX 5.0 was released in 1985, and conforms to the System V Interface Definition.

Xenix is not used anymore.

1.1.3 Linux

`linux` is a version of UNIX (some would say a "UNIX clone"), based on the original kernel work by Linus Torvalds in Helsinki, Finland. It runs mostly on the x86 and Pentium family of micro-processors, but has been ported to many other architectures.

The original kernel work by Linus Torvalds was done in the early 1990s. Full linux distributions first appeared circa 1993. They were (still are) based on Torvalds' kernel, but consisted mostly of software/applications openly available for anyone wishing to use it.

Different distributions of `linux` include `Red Hat`, `Slackware`, `Caldera`, `Debian`, `SuSE`, `Mandrake` and many others.

Although `linux` may be downloaded at no charge (for free), it has a copyright. It follows the GNU General Pulic License (GPL), in that it is freely available, but if sold, a copy of the source code must be provided.

Details on GNU's GPL is available at `http://www.gnu.org/copyleft/gpl.html`.

Today (2003), linux is the fastest growing segment of any UNIX operating system variant.

1.1.4 System V (formerly known as "AT&T" System V)

Because of a "1956 consent decree, AT&T was limited in the businesses that they could engage in and the patents they could licence..." [8, p. 4]. This consent decree delayed the introduction of AT&T's first commercial release by a few years. This first official release finally occurred in 1982, and was called System III. This version did not include "important Berkeley innovations, such as the C shell and screen editing capabilities" [8, p. 6].

The next version, System V, followed in 1983. This version did include utilities such as **vi** and **curses** from the BSD.

System V Release 2 was released in 1984, and System V Release 3 in 1987.

Finally, System V Release 4 was officially announced on Nov 1, 1989, and was released in 1990. This version combines System V Release 3, 4.3BSD, XENIX, SunOS and new features in one operating system.

Figure 1.1 offers a graphical view of the evolution of the UNIX operating system.

1.2 History (continued)

In 1988, two major competing UNIX groups emerged: the OSF (Open Software Foundation) and UI (UNIX International). OSF was lead by IBM,

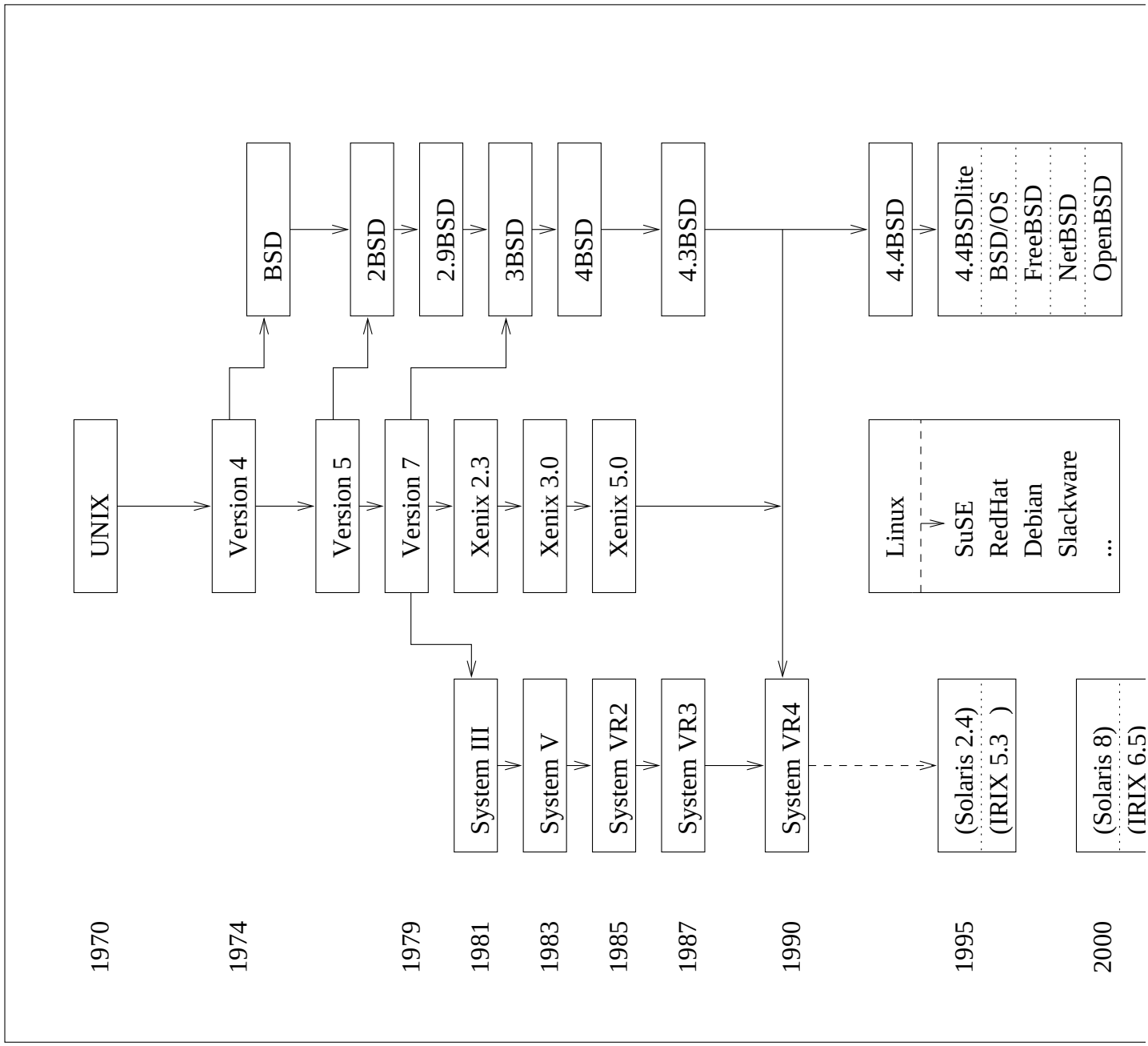

Figure 1.1: History of UNIX

Digital, and HP, whereas UI was lead by AT&T and Sun Microsystems.

The OSF was first formed to counter AT&T's alliance with Sun Microsystems to license and enhance the UNIX Operating system. OSF wanted to write their own version of UNIX, based on IBM's AIX.

UNIX International was formed shortly after to promote AT&T's System V UNIX system. The "special" alliance between AT&T and Sun Microsystems was reduced to partnerships between all UI members.

In January 1990, UI and its members were applauding the release of UNIX System V release 4 (SVR4, or V.4), which combines SVR3, BSD, XENIX, and SunOS.

Early in 1993, Sun was shipping Solaris 2.1, completely based on SVR4. Silicon Graphics starting shipping IRIX 5.0 in the spring or 1993 and IRIX 5.2 for all their systems in early 1994.

IRIX 6.0, a 64bit version, was shipping for the high-end systems in in the fall of 1994. IRIX 6.2 is SGI's "all-platform" newest release, released in the Spring of 1996.

In January 1993, Novell purchased USL thus taking control of UNIX. In September 1993, Novell re-organised and formed the UNIX Systems Group (USG), which took control of UNIX.

In late 1995, Novell sold the control of UNIX (including UnixWare, System V Release 4.2 code) to the Santa Cruz Operation (SCO). In 2001, Caldera Systems (a linux vendor) purchased SCO's Server Software Division as well as SCO's Professional Services Division and became `Caldera International`. What was left of SCO became `Tarantella`, now mostly a "professional services" company.

In 2002, `Caldera International, Inc.` changed its name to `The SCO Group`.

And in 2003, The SCO Group initiated a series of law suits against several UNIX/linux vendors, claiming UNIX copyrighted code had made its way into linux. Counter suits soon evolved.

This will likely take years to be resolved (to be followed!).

Most workstation vendors' current 64 bit Operating Systems: SGI with IRIX 6.5, Compaq (formerly Digital) with Tru64 UNIX (Compaq was acquired by HP in 2002), HP with HP/UX 11, Sun with Solaris 9.

1.3 The UNIX Operating System

The UNIX system is mainly composed of three different parts: the **kernel**, the **file system**, and the **shell**.

> [The **kernel**] is that part of the system which manages the resources of whatever computer system it lives on, to keep track of the disks, tapes, printers, terminals, communication lines and any other devices.
>
> The **file system** is the organising structure for data. The file system is perhaps the most important part of the UNIX operating system. The file system goes beyond being a simple repository for data, and provides the means of organizing the layout of the data storage in complex ways.
>
> The **shell** is the command interpreter. Although the shell is just a utility program, and is not properly a part of the system, it is the part that the user sees. The shell listens to your terminal and translates your requests into actions on the part of the kernel and the many utility programs. [7, p. 3]

file systems are discussed in chapters 2 and 3 of this manual, and **shells** are discussed in chapter 4.

One can imagine the UNIX system as a series of three concentric circles, with the inner circle representing the **kernel**, the second circle representing the programming **shell**, and the last one representing application programs. Figure 1.2 illustrates that concept.

The **shell** communicates to the **kernel**, and vice versa. The application programs can communicate directly with both the **shell** and the **kernel**.

1.3.1 Multi-tasking, Time Sharing

UNIX is a multi-tasking operating system, which means that a number of programs can run at the same time. Those programs (called **processes**) can communicate with each other.

For example, a C program could be compiling as mail is being read or a file is being edited.

Processes that "wake-up" occasionally, and/or regularly, are called **daemons**. Daemons are used to synchronise disks, send and receive mail, print documents, and so on.

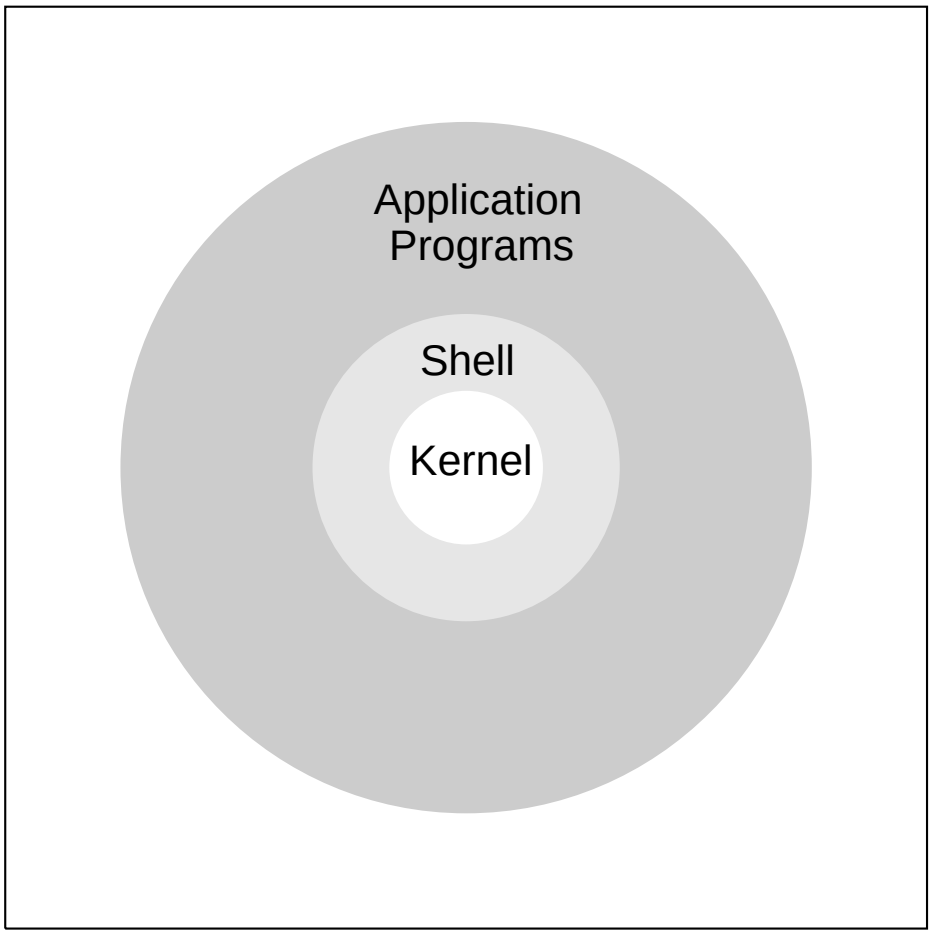

Figure 1.2: UNIX Systems

1.3.2 Multi-user

UNIX is also multi-user: two, three, or more users are able to use the same processor to execute their programs.

1.3.3 Network Capabilities

Today's UNIX workstations come with TCP/IP and ethernet connections. The same is true for PCs. At NRC the ethernet network connects dozens of Suns, Silicon Graphics, VAXes, IBM RS/6000s, HPs, and most PCs together.

From any of those nodes, it is simple to logon to a remote machine, send mail to a user on those machines, or transfer files to or from those nodes.

More on communications in a later chapter.

1.3.4 Portability

Traditionally, most operating systems were written in Assembler, for a specific architecture. It was therefore VERY painful – if at all possible – to 'port' the operating system to other architectures.

UNIX, on the other hand, is mostly written in the C language. This alone allows UNIX to be portable to many architectures. Today, UNIX/linux runs on more architectures than any other operating system in the world. Examples of such architectures/processors are the Motorola 680X0-based workstations, the 80X86 machines, the RISC based architectures (SPARC, MIPS, 88000), VAXes, IBM mainframes, Amdahl, Cray, and many more.

And this does not include all the different architectures linux runs on.

Table 1.1 lists different UNIX versions vendors use.

1.3.5 Flexibility

UNIX is also a very flexible operating system, both for system administrators and users. Program names can be changed. `alias`es can be defined. Arguments to programs can also be changed. New programs can be built, and put in the user's own `bin` directory, thus allowing further customisation of the system.

UNIX variant	Vendor
SunOS	Sun Microsystems
Solaris	Sun Microsystems
IRIX	Silicon Graphics Inc.
AIX	IBM
HP-UX	Hewlett Packard
OSF/1	HP (Compaq/Digital)
Digital UNIX	HP (Digital)
A/UX	Apple Computers
SCO	Santa Cruz Operation
Destiny	AT&T
XENIX	Microsoft
UTS	Amdahl
UniCos	Cray
UXP	Fujitsu
Linux	Public domain
FreeBSD	Public domain
NetBSD	Public domain

Table 1.1: UNIX variants on the market.

1.3.6 Software Available

Thousands of application packages [12, p. 2] are available for the UNIX/linux system.

In addition to the commercial packages, many programs are written and made available in the public domain. Examples of such packages are the X Window system, written at the Massachusetts Institute of Technology, the TEX system produced at Stanford, and many other utilities/applications written by individuals and organisations, for the benefit of the "Open Source Community". For many, it is their way of thanking the "Internet Community" for the vast amount of resources available.

Linux is a perfect examples of the incredible amount of software available, at no cost to individuals. In fact, most open source, freeware, or public domain packages used in the scientific world are developed and maintained on a linux platform. The same can be said of any open source, freeware, or public domain packages developed on any UNIX variant: linux is the

platform of choice to develop/write software.

1.3.7 Virtual Memory

The UNIX operating system has virtual memory, or swap space, which means it can run programs bigger than the amount of RAM the computer actually has! The amount of virtual memory is decided upon by the system administrator.

1.3.8 Case Sensitivity

The most common mistake for beginners involves the use of mixed case in UNIX commands: UNIX is case sensitive, i.e., "a" is different from "A".

1.4 UNIX Philosophy

UNIX's philosophy is the same as the C language's: it assumes users know what they are doing!!!

1.5 Exercises

1. What is the operating system that covers the widest range of architectures? Why?
2. What are some of the advantages of running UNIX?
3. What is UNIX's main philosophy?
4. Name some of the reasons people would install and use linux on their desktops or servers.

Chapter 2

File System

The UNIX File System manages and controls access to files and directories. It keeps track of opened and closed files, and manages files on the hard disk.

2.1 File and Directory Names

2.1.1 Length

The maximum length of directory and file names varies from system to system. Some versions (such as HP's HP/UX) may restrict the names to 14 characters, whereas most allow much longer file and path names.

For example, Sun and SGI workstations allows file names up to 255 characters long, and `path` names to be as long as 1023 characters (a `path` name is the full length of the file name, beginning at the top – root – of the file system).

2.1.2 Conventions

Naming files is very simple: any ASCII character can be used. It is, however, recommended that no metacharacters (*, {, }, `[`, `]`, ?, `$`, \, `~`, >, <, |, `&`) be used. It is also recommended to simply use letters, digits, underscore (_), hyphen (`-`) and the dot (`.`).

By convention, files ending with `.c` are `C` language files. File names ending with `.p`, `.f`, `.s` are respectively `PASCAL`, `FORTRAN`, and `assembler` program files. Header (include) files usually end with `.h`.

File names beginning with a . (dot) are *hidden* files: they do not appear when files are listed. Those files are typically configuration files. Usually, their name consists of the package they represent followed by the `rc` (run command) string. `.mailrc` and `.newsrc` are two examples.

Any file name may contain several dots and/or underscores. These are NOT separators.

2.2 Structure of Directories, Files

The filing structure of the UNIX operating system is hierarchical. It is a tree structured system, completely open (assuming necessary permissions) to every user on the system, with everything emerging from `/` (`root`) at the top.

Figure 2.1 illustrates this concept.

Every directory has a parent, and—possibly—one or more children. Those children can in turn be parents.

A directory is a special type of file. A file and a directory of the same name within the same directory is therefore impossible.

NOTE that a file is a linear sequence of characters, including line feeds (`\n`); there is no specific file structure.

Everything in UNIX is considered a file; a directory is a special kind of file, and so is the keyboard (`/dev/kbd`) and the system's console (`/dev/console`).

2.3 Permissions/File Access Modes

A `permission` defines the *ownership* of a file and the *access* of all users to that file (or directory).

Each file and directory has permissions associated with it. Those permissions are made up of three groups of three characters: `rwx rwx rwx` (`r`ead, `w`rite and e`x`ecute). The first group represents the owner permissions on the file, the second group represents the group permissions on the file, and the third group represents the world permissions on the file.

The `owner` is the user owning the file.

Each user has an internal `user number` (UID), and an internal `group number` (GID), set by the system administrator. Everyone with the same

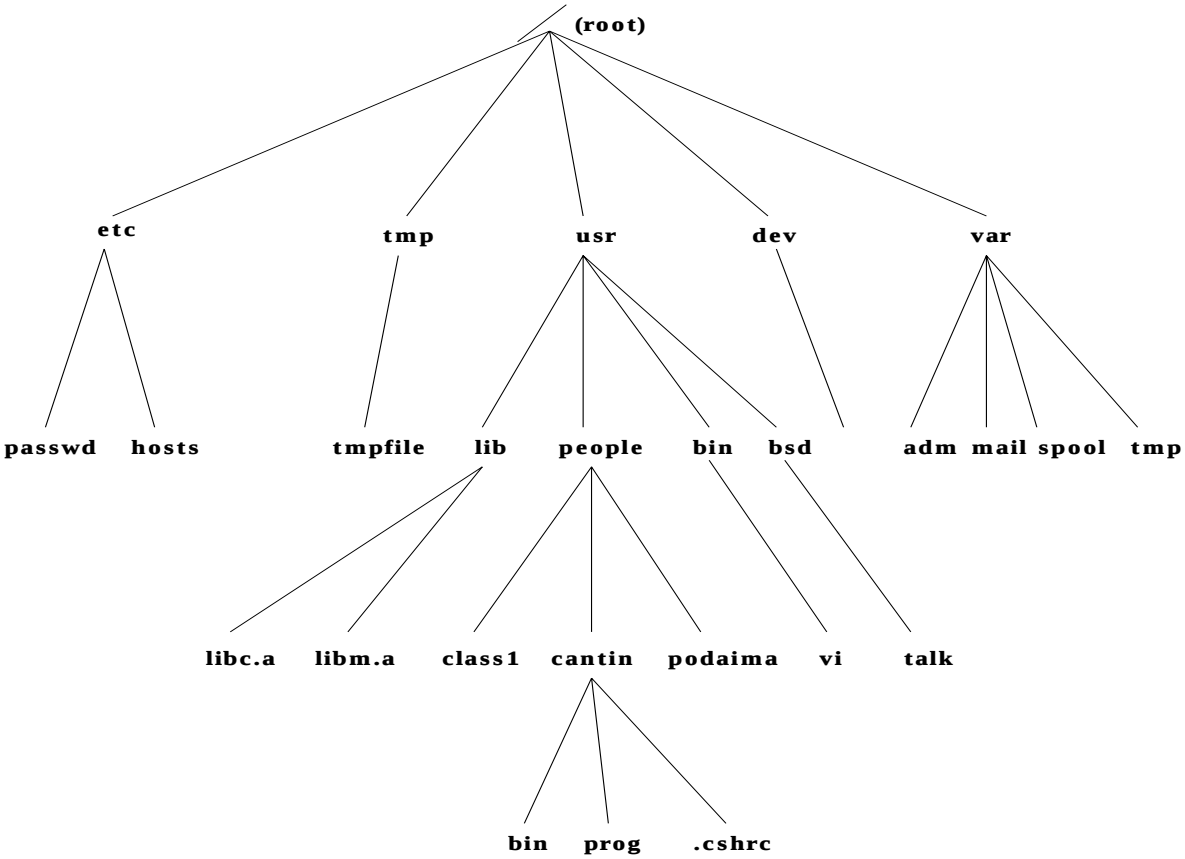

Figure 2.1: Structure of File System

`group number` is said to be in the same `group`. Group permissions apply to everyone else with the same `group number`.

`world` is everyone else.

For example, the file `phone.numbers` may have permissions set as `rwxrwxr--`. This would mean that the `owner` of the file can read the file, write to it, and execute it. People in his/her `group` have the same privileges. Everyone else can only read the file; they cannot change, or execute it.

The permissions of a file/directory can be changed with the `chmod` command.

2.3.1 chmod: Change Mode (Permissions)

The `chmod` command allows a user to change the permissions of a file/directory. To use `chmod`, the user must be the owner of the file.

The syntax of the command is:

chmod [**-R**] *mode filename(s)*

-R is an option.

-R (Recursively) will cause all files and directories within (underneath) the file/directory whose permissions are being changed to take those permissions.

mode may be specified as three octal values (one for each of the three sets of permissions): if any of the permission bits `r`,`w`, or `x` is set, the corresponding permission is enabled: give it a 1. If not, give it a 0. Then, for each of the three permission groups, interpret the three binary numbers as an octal number.

Another way to explain it is to give different weighing factors to the different permissions: 4 to "r", 2 to "w", and 1 to "x". If the permission is set, add the weighing factor. Otherwise do not add anything to the group value.

Using the above example of the file `phone.numbers`, which had the `rwxrwxr--` permissions, it would translate to `111 111 100`, or `774`, or 4+2+1 4+2+1 4+0+0:

```
rwx rwx r--
111 111 100
7   7   4
```

If write permission for `world` (sometimes called `others`) is added, *mode* is changed to `776` (or `111 111 110`, or 4+2+1 4+2+1 4+2+0).

mode may also be specified symbolically as `+r` which would add read permissions to everyone, or as `-w` which would take away write permissions from everyone (all groups).

filename(s) may be one or more filenames, and/or one or more directories, separated by blank spaces.

For example, to change the permissions on file `phone.numbers`, from `rwxrwxr--` to `rwxrwxrw-`, the command

chmod 776 phone.numbers

or

chmod +w phone.numbers

or (because we are changing it only in the `others` group)

chmod go+w phone.numbers

which means "group others add write", could be used.

The general syntax, using the conventional method is

chmod [ugo]+|-[rwx] *filename(s)*

Where

u : permission for user/owner.

g : permission for group.

o : permission for others.

r : read permission.

w : write permission.

x : execute permission.

Any combination of `ugo`, `rwx` may be used. If none is used, then all three are assumed. One exception: if `chmod -` is used, the result depends on the value of *umask* (`umask` sets the default permissions on files; usually 022 at NRC – see man page for details).

It is also acceptable to put a combination of "`[ugo]+|-[rwx]`", as long as they are separated by a comma (`,`), as in

chmod ug+x,o-x *filename*

.

Typing the command

chmod --help (linux)
or
chmod (generic UNIX)

by itself will display the `usage` of the command.

Refer to the UNIX manual of your machine for more options on the command.

2.4 Exercises

1. My friend just created a file in his/her directory. Can I go and change permissions on it? Why, or why not ?

2. I just created a new file in my directory, but I would like it to have `rwx` permissions for me, and `r` permissions for everyone else. What command can I use to change those permissions?

3. I have a file with permissions `rwxrwxr-x`. I want it to have `rwxr-xr-x` permissions. What commands (I want two) can be used to change the permissions on it?

4. Take the same file. I want to remove all execute permissions for everyone, except myself. Again, which command can be used to achieve my goal?

Chapter 3

Tour of the File system

3.1 /: Root

`/` by itself is the `root` or the top, the beginning of the file system. "`super user`" is usually the only user allowed to write to that directory. Each directory contains files (and directories) related to one group of subjects.

Some of those directories are discussed below. For more information on any of them, please see [3, pp. 17-48].

3.2 `/bin` and `/usr/bin`: Commands

The `/bin` and `/usr/bin` directories contain public commands. Those commands may be in *binary*, or in *shell* format.

Binary format commands are those written in C, C++, FORTRAN, or any other compiler language. The shell format commands are interpreted by the *C* or *Bourne* shells, whichever executes them (more on *shells* later).

When UNIX was first written, `/bin` and `/usr/bin` physically resided on two different disks: `/bin` being on a smaller faster (more expensive) disk, and `/usr/bin` on a bigger slower disk. Now, `/bin` is a *symbolic link* to `/usr/bin`: they are essentially the same directory.

3.3 /dev: Devices

In UNIX, every output device is referred to as a file. For writing to the `console` (the monitor), the file to write to is `/dev/console`. To write to tape, the file is `/dev/nrst0` (on a Sun SPARCstation 2). Even the keyboard writes to a file (`/dev/kbd`) which is read by the operating system.

3.4 /etc and /usr/etc: Management

The `/etc` and `/usr/etc` directories are the "management" directories. `/etc` and `/usr/etc` contain programs used by the system administrator to administer and configure the system to the needs of the users.

One of the most important files of the `/etc` directory is the `passwd` (pronounced "password") file. The `passwd` file contains basic information about each user's account, including the logon and, in some cases, the password (encrypted) of every user (in the other cases, the encrypted passwords are in a file called `/etc/shadow` which only the super user has access).

Again, the commands are in two different directories simply for historical reasons.

3.5 Home Directories

The `HOME` directory is the directory in which the user lands when logging into the system. Different vendors use different conventions for their `HOME` directory.

Vendor	`HOME` directory
HP	`/users/`*login*
IBM	`/u/`*login*
Linux	`/home/`*login*
Silicon Graphics	`/usr/people/`*login*
SUN (SunOS)	`/home/`*machine_name*`/`*login*
SUN (Solaris)	`/users/`*login*

Table 3.1: `HOME` directories

login refers to the user's account name whereas *machine_name* refers to the name of the system where the account resides.

3.6 /lib and /usr/lib: Libraries

"These areas were originally intended for libraries of compiled subroutines, to be searched during the linking phase following the compilation of a program" [3, p. 27]. Those areas now also contain programs not directly available to users but used by other programs, such as compilers. They also contain databases (fonts, conversion units, etc.) used by programs.

IRIX 5.2 (from SGI) has 5 files in /lib. /usr/lib has a few thousand. On other systems, like Suns, /lib *points* to /usr/lib. Hence, both directories share the same information.

3.7 /tmp: Temporary Directories

/tmp is a necessary "temporary" directory. It is available to all users to read from and write into. But, along with /var/tmp and /usr/tmp, it is mostly used by compilers and by UNIX as temporary storage for intermediate files.

3.8 /var: Directories

The philosophy behind the /var directory structure is that it contains directories that vary in size and that tend to grow if not maintained properly.

3.8.1 /var/adm: Administration

Administration information is kept in this directory. Logins—who logged on when and for how long—are also kept here. Messages output to the console, system crashes and accounting information for each user are kept in specific files that grow continuously.

The system administrator can monitor which commands are most often used by referring to database files from this directory. He/she may find out how much CPU time a specific account uses, even the commands used by specific users (if the necessary configurations have been made).

The system administrator may or may not choose to keep accounting and administration data.

3.8.2 `/var/mail`: System Mailboxes

`/var/mail` contains every user's system mailbox. `/var/mail/`*`user`* contains *user*'s incoming mail, and any mail not moved out of the mailbox.

3.8.3 `/var/spool`: Spooling Areas

Mail waiting to be sent, and files waiting to be printed are stored in this area of the file system.

`/var/spool/lpd` contains files waiting to be printed on the printer.

`/var/spool/mqueue` contains mail messages waiting to be sent out, or to be received in the local mailboxes.

3.8.4 `/var/tmp`: Temporary Directory

Is a temporary directory for users to use at their leisure. It is also used by programs as temporary storage. Its content is guaranteed only for the duration of the session.

3.9 `/usr`: Directories

`/usr` contains a series of directories containing specific, more permanent information.

What follows is a description of a number of directories within the `/usr` directory.

3.9.1 `/usr/bsd`: Berkeley Binaries

This directory contains some of the binaries "borrowed" from the UNIX BSD version.

3.9.2 `/usr/demos`: Demonstration

Demonstration programs are kept in this directory. Some systems don't have it; others don't make it available to users.

Sun's version of that directory is `/usr/demo` (no "s").

3.9.3 `/usr/include`: Include Files

Many C programs have `#include <file>` statements. This directory contains the files referenced by those statements.

Many of the files contain information on data structure for reading specific UNIX files, or for getting particular information from the operating system (such as who is logged on, what is the type of a specific file, etc.). Other files simply assign a name to an integer value: an example is the EOF string, defined as (-1) in file `/usr/include/stdio.h`.

3.9.4 `/usr/local`: Local Programs

Most sites have their own set of local utilities, programs and applications. To distinguish those from the system ones, they are placed in a directory separate from the standard operating system: `/usr/local`.

By following that convention, operating system upgrades do not affect "public domain" and "homegrown" software.

3.9.5 `/usr/sbin`: More Binaries

This directory contains an ever increasing number of binary, or executable, files.

3.9.6 `/usr/share`: Shareable Directories

Those are typically directories that can be shared with other systems, allowing a saving of disk space.

The information in those directories are typically independent of the model of the workstation (for the same vendor).

`/usr/share/lib/dict`: Dictionary

The file `/usr/share/lib/dict/words` contains over twenty-four thousand (24,000) words, in readable ASCII format. It is generally used by speller utilities.

In linux, that file is `/usr/share/dict/words`.

`/usr/share/lib/Insight`: Documentation

On Silicon Graphics, the on-line documentation (other than `man pages`) are store in this directory.

`/usr/share/doc`: Linux Documentation

On linux systems, `/usr/share/doc` contains a wealth of documentation. This is where most of the linux documentation is kept.

`/usr/share/lib/spell`: Speller

The files `hlista`, `hlistb`, and `hstop` contain a list of "American-only words, British-only words, and 'stop-words' (words which the program may think acceptable, but which are not) for the `spell` command" [3, p. 39]. These three files are hashed (i.e., we can't read them).

`/usr/share/lib/terminfo`: Terminal Database

This directory contains the description of terminals, ranging from the `ansi` to the `vt100`.

`/usr/share/man` and `/usr/share/catman`: Manuals

On-line documentation resides in `/usr/share/man` and `/usr/share/catman`. The manual pages are kept in several different directories, depending on what type of information the "`man page`" (for manual page) holds. Typically, section 1 holds documentation for common commands, section 2 contains information concerning system calls, and so on.

The manual pages are read using the `man` command.

The `/usr/share/catman` directory contains the pre-formatted man pages.

3.9.7 `/usr/tmp`: Temporary

This is not a real directory. Instead, it is a `symbolic link` to `/var/tmp`.

3.10 Exercises

1. You want to see which demonstration programs you have on your system. In which directory are you more likely to find these programs?
2. Where would the most used commands be ?
3. Where would the dictionary be on your system?
4. Where would your home directory be?

Chapter 4

Shells

A *shell* is a program that interprets and runs the commands typed at the console by the user. The *shell* sends requests to the kernel, which executes them.

UNIX comes with various shells, the most common being the `C Shell`, the `Bourne Shell` and the `Korn Shell`. IRIX adds the `Tenex Shell`. linux includes both the `Tenex Shell` and the `Bourne Again Shell`. Each `shell` can be programmed. Programs containing `shell commands`, are called `C shell script`s, `Bourne shell script`s, `Korn shell script`s, etc. They each have their advantages. "The general consensus of views from users familiar with both (`C` and `Bourne`) shells appears to be that the C shell is superior for interactive work, because of its process control features, while the Bourne shell has more powerful language constructs, and so is better for use in shell scripts" [3, p. 72].

The `Tenex` shell, also known as the `T Shell`, is a superset of the `C` shell, in that it includes all what the `C` shell has to offer, plus additional functionalities. Written in the late 80s, it is in the public domain.

The `Korn Shell`, also written in the late 80s, incorporates features and functionality of both the `C` and `Bourne` shells, while retaining the speed of the `Bourne` shell (with which it is also upward compatible). The same might be true of the `Bourne Again` shell as it combines the best of the `Korn Shell` and the `Tenex Shell`.

Typical Linux installations use the `bash` shell (Bourne Again SHell) as its default shell. `bash` contains the same feature as the `Bourne` and `Korn` shells, while retaining the flexibility and the features of the `C` and `Tenex` shells.

At the Research Computing Support Group, we tend to use the `T` shell

as the default shell on IRIX, and use `bash` (the `Bourne Again SHell`) with linux.

This chapter will first describe the features common to the five shells, such as input/output redirection, and metacharacters (wildcards). It will then go on to explain main differences between the various shells introduced.

4.1 Input/Output Redirection

Most UNIX commands expect input to come from a file(s), and produce output to another file(s). "One of these files is the *Standard Input*, and is the place from which a program expects to read its input. Another is called the *Standard Output*, and it is the file to which the program writes its results. The third file is the *Diagnostic Output* (also called *Standard Error*), and it is a file to which the program writes any error responses" [7, p. 85].

Table 4.1 shows a summary on how input/output redirection may be used.

Generally, the standard input is taken to be the keyboard (input is typed by the user), the standard output is the terminal screen, as is the standard error. They are known as file descriptor 0, 1 and 2 respectively.

4.1.1 <: Input Redirection

For commands expecting the input to come from the keyboard, < can be used to get the input from a file instead of having to type it in. A simple example is

cat < file

This will cause `file` to be displayed on the screen (the standard output). Note that it so happens that `cat` can take a filename as an input parameter, so

cat file

would result in the exact same response.

A more typical example is the use of a Fortran or C program (say `prog`) which takes data from the keyboard. If the user knows in advance what the data will be, the program could be run this way:

prog < data

where `data` contains the data to be read by the program.

4.1.2 >, >>: Standard Output

Just as < is used as the standard input, > is used as the standard output redirection.

cat file

usually displays the results on the terminal screen.

By concatenating > `outfile` to the command, as in

cat file > outfile

the output will be redirected to `outfile`. If `outfile` did not exist, it is created. If it does exist, the previous contents are lost, and replaced with the new output!!

This problem can be avoided by using >>:

cat file >> outfile

where `file` will be appended to `outfile`.

4.1.3 2>, >&: Standard Diagnostic (Error) Output

The redirection of the standard diagnostic output is not quite as simple as for input and output redirections. It is actually different in the C shell and in the Bourne shell.

In the C shell, >`&` is used to redirect both the output AND the diagnostic message(s) to a file. Therefore, using the previous example,

cat file >>& outfile

would cause both the output and the error messages (diagnostic output) to be appended to `outfile`.

The standard output and error may be sent to two different files by using

(cat file > outfile) >& errfile

In the Bourne, Korn and Bash shells, it is possible to separate the diagnostic output from the standard output:

cat file >> outfile 2> errfile

will cause the output of `cat` to append to `outfile`, and the error messages to go to `errfile`. Note that the redirection is `2>`.

In order for the error and output to go to the same place, the following is used:

cat file >> outfile 2>&1

(the &1 really means the first descriptor, which happens to be the new standard output). If the error messages are to be ignored:

cat file >> outfile 2> /dev/null

`/dev/null` is a special file which acts like an *infinite sink*.

4.2 |: Pipes

The concept of using a command's output for another command's input is called `piping`. A `pipe` eliminates the need for temporary files: instead, the output of one command goes directly to the input of the next.

A pipe is not only faster than creating temporary files, it is faster than using redirection: when using redirection, the system has to wait until all data has been redirected to begin the next command, whereas with piping, the next command is started as soon as data is available to it.

As a `pipe` example, let's find out how many users are logged onto the system:

who | wc -l

where `who` is the command to display all users on the system (one line per user), and `wc -l` is the command to count how many lines the input file contains (i.e. the number of users).

Without the `pipe` command, the following would have to be used:

who > /tmp/tempfile
wc -l < /tmp/tempfile

Note that the sign for `pipe` is a vertical bar |.

4.3 Pipes and Redirections

When `piping` a command, only the standard output is sent through the pipe. To send the standard error as well, we use

cmd1 |& cmd2

in the C and Tenex shells. In the Bourne/Korn/Bash shells,

cmd1 2>&1 | cmd2

Redirection	`C/Tenex` shell	`Bourne/Korn/Bash` shell
standard input	*cmd* < *infile*	*cmd* < *infile*
standard output	*cmd* > *outfile*	*cmd* > *outfile*
standard error only	(*cmd* > *outfile*) >& *errfile*	*cmd* 2> *errfile*
standard output and error	*cmd* >& *outfile*	*cmd* > *outfile* 2>&1
pipe	*cmd1* | *cmd2*	*cmd1* | *cmd2*
std output and error to pipe	*cmd1* |& *cmd2*	*cmd1* 2>&1 | *cmd2*

Table 4.1: Redirections and Pipes.

4.4 `&`: Background Processing

UNIX is a `multitasking` operating system. Sure, but how does one run a number of programs? One answer: use the `background` process facility!

When typing command names and waiting for the output, the user is said to be in `foreground` mode: the user has to wait for the program/command to finish executing before typing the next one. But if the user types an ampersand (`&`) at the end of the command line, that particular command is executed in the `background`, and the command prompt immediately reappears, along with a `process id number`, called `pid`, as in:

```
prompt> latex bigfile.tex > tex.out &
[1] 1198
```

`latex bigfile.tex` displays messages onto the standard output. But, in this case, it has been redirected to `tex.out`. 1198 is the `pid` of the job.

Upon completion, the message

```
[1]    Done          latex bigfile.tex > tex.out
```

will be displayed on the screen.

NOTE: if the terminal used is connected to a serial port on the UNIX system, then all `background` processes should be finished before logging out.

If, at logout time, there are still jobs running in `background` mode, then this could potentially hang the serial port the terminal is connected to.

In this case, `batch` (explained in a later chapter) should be used.

4.5 `Metacharacters` (Wildcards)

All shells use the concept of `metacharacters`. `Metacharacters` are special characters used to specify a set of file names.

4.5.1 ∗: Any Character(s)

An ∗ matches any combination of characters, including the `null` string. The ∗ could be used to erase all files ending with `.log` by using

rm ∗.log

where `rm` is the command to erase a file.

4.5.2 [,]: List of Characters

To express a list of characters, or numbers, `[` and `]` are used. The comma is used as a separator between the characters. If a hyphen is used, a range of characters is represented.

For example, all files ending with 1, 2, 3, 5, 6, 7, and 9 could be represented as

∗[**1-3,5-7,9**]

or all files ending with an upper case letter:

∗[**A-Z**]

4.5.3 ?: Any Single Character

The ? is used to represent any single character. ?? would represent any combination of two characters.

For example, all files beginning with `ar` and four characters long are represented by

ar??

4.5.4 {,}: Alternatives (except Korn Shell)

Consider a directory with most files beginning with `fil`. Two of those files are `filling`, and `filler`. An easy way to erase those files is to use:

rm fil{ler,ling}

This is equivalent to issuing:

rm filler
rm filling

The shell first interprets the command with the first entry within brackets, then with the second (then third, fourth...).

4.5.5 \: Escape

Any metacharacter can be represented as a normal ASCII character by preceding it with the \ (backslash).

For example,

rm fi*ler

would delete file name `fi*ler` NOT all file names beginning with `fi` and ending with `ler`.

4.6 Shell/Job Control

Usually, when a command is issued, it is automatically executed in "interactive" mode, and the prompt comes back only when that command has finished. In many instances, however, it may be needed that the prompt be restored prior to the command finishing.

This is especially true for GUI-based tools, like `xclock`, `netscape` or a graphing package like `xmgrace`.

Job control commands are available with most shells.

4.6.1 `jobs`: list background jobs

The command

```
jobs
```

will list the jobs initiated to be sent in the background. In other words, all jobs, within the current shell, ending with an ampersand (&).

For example, if two commands, `xclock &` and `mozilla &` had been issued, followed by `jobs`, the session would look similar to

```
prompt> xclock &
[1] 27207
prompt> mozilla &
[2] 27239
prompt> jobs
[1]-  Running                 xclock &
[2]+  Running                 mozilla &
prompt>
```

The `[1]` and `[2]` are referred to as *job numbers*. Other commands, namely `bg` and `fg` (seen shortly), may refer to them as `%1` or `%2` meaning job [1] and job [2] respectively.

4.6.2 `CTL-Z`: suspend a job

After issueing a command such as `xclock` (or most X-based utilities), the prompt is gone until the program is done. In the case of `xclock`, and many other X-based utilities, the tool might be on the desktop all day.

If `CTL-Z` is pressed (press and hold the CONTROL key, usually labeled `Ctrl`, then press the Z key) the currently running job in the shell will be *suspended*, which means it will be stopped (not killed, only stopped), and the command prompt will reappear.

`bg` (seen next) will resume execution of the command in background mode, while keeping the prompt active.

The best way to prevent this problem is to start all GUI-based programs in background mode, as is

```
xclock &
```

4.6.3 bg: send a job in the backgroup

Used as

`bg` [*%job number*]

When a job has been suspended with `CTL-Z`, it may be resumed with `bg`. This resumes execution of that job, from the point where it had been suspended.

If there are more than one suspended jobs, *%job number* is used to specify which job needs to be resumed.

4.6.4 fg: bring a job in the foreground

Used as

`fg` [*%job number*]

it is used to bring a specific job in the foreground, getting rid of the prompt.

If there is only one job, it not necessary to use the `job number`.

4.6.5 CTL-C: kill a job

In most instances, `CTL-C` will kill the currently running job. It must be running in foreground mode, which is usually the case when a job is not submitted with an ampersand (`&`).

4.6.6 Sample session

The following shows a short session where two jobs, `xclock` and `netscape`, are already running in the background.

`jobs` is first issued to confirm the two jobs are running.

`xclock` is then brought in the foreground.

`xclock` is then stopped with CTRL-Z (we cannot see the CONTROL-Z in the session, except that `[1]+ Stopped` is displayed, and the prompt now shows up).

`jobs` is issued again, to show which jobs are running. Notice that we also see suspended jobs.

`mozilla` is then brought in the foreground.

`mozilla` is then killed, with CTRL-C. The prompt comes back.

`jobs` shows that there is now only one job running.

```
prompt> jobs
[1]-  Running                 xclock &
[2]+  Running                 mozilla &
prompt> fg %1
xclock
                            <<<<<---- CTRL-Z was pressed here.
[1]+  Stopped                 xclock
prompt> jobs
[1]+  Stopped                 xclock
[2]-  Running                 mozilla &
prompt> fg %2
mozilla
                            <<<<<---- CTRL-C was pressed here.
prompt> jobs
[1]+  Stopped                 xclock
prompt>
```

4.7 C Shell

As previously mentioned, the `C shell` is usually used as an interactive shell, mostly because of its more flexible command line interpreter.

The default prompt for a C shell is the percent (%) sign.

It is invoked by the command `csh`.

4.7.1 `path`: Search Path

When executing a command typed on a line, UNIX searches for the name of that command in a series of directories. The order and the name of those directories is found in the `path` variable.

The `path` variable should be defined in one of the system-wide login files. This allows the system administrator to modify one file only when adding another directory to the `path` of all users of a system.

To print the contents of a variable, the `echo` command is used, followed by the variable whose contents are to be displayed, preceded by a dollar ($) sign. Hence, to display the contents of the `path` variable for the user `cantin`:

```
prompt> echo $path
.  /usr2/people/cantin/bin /usr/local/bin/tex
/usr/local/bin/imtools /usr/local/bin /usr/sbin
/usr/bsd /usr/bin /bin /usr/bin/X11
```

The first directory defined by `$path` is `.` (the current one) followed by `/usr2/people/cantin/bin`, then `/usr/local/bin/tex`, and so on until `/usr/bin/X11`. As soon as the program is found, it is executed.

When a UNIX command is used for this user, a search begins in the above directories in the order given.

To add to the path variable,

```
set path = ($path /new_path)
```

could be added to the user's own `.login` file.

Note that `$path` refers to the current `path` variable.

4.7.2 `history`

The `history` concept provides the user with the capability of calling and editing previously used commands, modifying others, listing the last commands used, and so on. Here are some of the commands using the `C` shell `history` concept:

- `!!`: recall the last command and execute it.
- `!`*num* where *num* is a number: recall and execute command number *num*.
- `^`*str1*`^`*str2*`^`: change *str1* into *str2* and execute command. The change is done on the last command.
- `!`*num*`:s/`*str1*`/`*str2*`/`: change *str1* into *str2* in command *num*.

- `history` recalls and list the last commands used.

If the user prompt does not include the history number, change the `prompt` variable in the `.cshrc` file to include the `history number` of the command by adding an exclamation mark in the definition as in

```
set prompt = "!  cantin>"
```

4.7.3 `alias`

The `alias` mechanism provides the user with the ability to rename a command, or change the default options of a command.

The list of `aliases` is usually defined in either the `.cshrc` file in the user's home directory and/or in `/etc/cshrc`.

alias [*command def*]

where *command* is the name of the new command, and *def* is how it is to be executed. **alias** typed on a line by itself will list all current aliases.

To create a new command called `erase` to work as the `rm` command:

alias erase rm

From that point on, typing **erase** will be the same as typing **rm**.

The `unalias` command cancels `alias`.

4.8 Tenex Shell

As mentioned, the `T` shell is a public domain shell. It contains all features of the `C` shell, as well as a few extra ones. Its default prompt is the greater-than-sign (>).

It is invoked using `tcsh`.

The main reason many users are using the `T` shell is that it allows the user to recall commands *and edit them* interactively on the command line, which is not very easily done in the `C` shell. This is done, as on VAXes, using the up and down cursor keys. It can also be done using `EMACS` control sequences such as

CTL-P, which recalls the last command executed. Another CTL-P would recall the command previous to that...
CTL-N, which recalls the next command in the history list.
CTL-F moves the cursor one position to the right.
CTL-B moves the cursor one position to the left.
CTL-E moves the cursor to the end of the line.
CTL-A moves the cursor to the beginning of the line.
CTL-H erases the character directly to the left of the cursor.
CTL-D erases the character directly above the cursor.
TAB (the escape key twice) is a *command and filename completion* (allows the system to provide the rest of the command or filename text if the first characters typed before $< TAB >$ are unique to that command/file name).

Other extra features include an auto-logout facility. The `man` page of the local installation will describe the facilities of the `T` shell (if present).

4.9 Bourne Shell

The default prompt for the `Bourne` shell is a dollar sign (`$`). Typing

sh

would start a `Bourne shell`.

4.9.1 PATH

In this shell, the `PATH` variable is in upper case characters. To display the variable:

$ echo $PATH
`.:/usr2/people/cantin/bin:/usr/local/bin/tex:/usr/local/bin/imtools:`
`/usr/local/bin:/usr/sbin:/usr/bsd:/usr/bin:/bin:/usr/bin/X11`

(The above is "one line").

To add to the `PATH` variable, make the following change:

```
PATH=$PATH:/new_path; export PATH
```

in your own `.profile` file.

In most cases, `PATH` is defined in `/etc/profile` by the system administrator.

4.9.2 history

This concept is not found in the `Bourne` shell.

4.9.3 alias

This concept is also not found in the `Bourne` shell.

4.10 Korn Shell

The `Korn` shell (`/bin/ksh`) was written by David Korn in the 1980's. It is a superset of the `Bourne` shell. As a result many of the commands in the `Korn` shell are similar to the `Bourne` shell. Its prompt is the dollar sign (`$`), the same as the Bourne shell.

The login file used by the `Korn` shell is the same as with the Bourne shell, `/etc/profile` and `.profile`.

4.10.1 `PATH`: Search Path

`PATH` is the variable which contains the order in which the shell will look to find the command(s) to be executed. Directories are separated with a colon (`:`).

4.10.2 history

The `Korn` shell has the ability to remember previously executed commands. It is possible to recall previously issued commands, and edit them *before* they get executed. Commands can be recalled and edited using two methods:

1. `emacs` editor style:
   CTL-P recalls the last command executed. Another CTL-P would recall the command previous to that...
   CTL-N recalls the next command in the history list.
   CTL-F moves the cursor one position to the right.
   CTL-B moves the cursor one position to the left.
   CTL-E moves the cursor to the end of the line.
   CTL-A moves the cursor to the beginning of the line.
   CTL-H erases the character directly to the left of the cursor.

CTL-D erases the character directly above the cursor.
ESC ESC (the escape key twice) is a *command completion.*

This style of command line editing is enabled with

set -o emacs

either in your `.profile` file, or manually during your logon session.

2. `vi` editor style: ESC k will bring the last command on the line, and put you in *vi mode* (actually, ESC puts you in vi mode, and `k` recalls the last issued command).

   `l` moves the cursor one position to the right.
   `h` moves the cursor one position to the left.
   `k` recalls the previous command.
   `j` recalls the next command.
   `0` moves the cursor to the beginning of the line.
   `$` moves the cursor to the end of the line.
   `fc` finds next character `c` on the line.
   `i` puts you in input mode.
   `a` brings the cursor forward one, then puts you in input mode.
   `A` brings the cursor to the end of the line, and puts you in input mode.
   ESC takes you out of input mode (or beeps if in `vi` mode).

   This style of command line editing is enabled with

   set -o vi

   either in your `.profile` file, or manually during your logon session.

The previous command can be recalled and automatically executed by typing

r

on a command line.

r [***string***]

will recall and execute the last command used, that began with *string.*

4.10.3 `aliases`

A command can be defined using the `alias` concept. An alias is defined by issuing a command in the form

`alias` *command=definition*

For example, to create a command called `erase` that would be functionally the same as `rm`,

alias erase="rm"

Note that it is always safe to put the definition of the command between quotes, as the definition could be a command with specific flags.

`aliases` can be issued on the command line, or put into the `.profile` login file.

The command `alias`, on its own, lists the defined aliases.

To disable an `alias`, use

unalias *command*

4.10.4 Other Miscellaneous Commands

`.` *filename*

executes each line from *filename* within the present shell, without restarting a new one.

`read "`*var?prompt*`"`

can be used to read variables from the keyboard.

`ulimit -a`

lists the limits imposed on users by the system.

Among these limits is `TMOUT` is the time value after which the present shell will be killed if no keys are touched for that many minutes.

4.11 Bash Shell

`bash` (Bourne-Again SHell) is a superset of the `Bourne` shell, but also includes the functionality of the `Tenex` and `Korn` shells. It is written by the Free Software Foundation group under the GNU project, and so is freely distributed with all Linux distributions.

In fact, it is the default shell on most Linux distributions.

4.11.1 `PATH`: Search Path

As with the `Bourne` and `Korn` shells, the variable containing the search path is the upper case `PATH`.

4.11.2 history

`bash` remembers history the same way the `Tenex` and `Korn` shells remember history. The arrow (cursor) keys may be used to navigate through the history list.

As with the `Korn` shell, both the `emacs` and `vi` editor styles may be used to navigate through the history list, using

```
set -o emacs
or
set -o vi
```

`set -o emacs` is, in most cases, the default.

`history` may also be recalled using the `C` shell functionality (`!!`, `!`*num*, etc.).

4.11.3 `aliases`

`aliases` are used the same way they are in the `Korn` shell:

`alias` *command=definition*

For example, to create a command called `erase` that would be functionally the same as `rm`,

alias erase="rm"

And, as with both the `Tenex` and `Korn` shell, aliases are removed with

unalias ***command***

4.12 Exercises

1. I have a program, called `names`, which prompts the user for a series of names and telephone numbers. But I also have a file containing all that information, in the right order. How can I run `names` without having to type all the information on the keyboard?

2. Take the same program `names`. Assume that the program usually reads all the information, then sorts the information by last name, and finally outputs the sorted information on the screen. But I want it in a file. How can I do that? What if I want to append it to a file called `sorted.names`?

3. I want to get rid of ALL my object files (they all end with `.o`). What one command can I use?

4. I want to remove all my files beginning with a number, but also end with an upper case character. How do I do it in one command?

5. I want to remove all four-character files beginning with `z`. What command can I use?

6. List all equivalent commands (no wildcards) to `rm my{fil,cap}e[1-4,7,9]`.

7. I have a file containing an asterisk (*). How can I remove it?

8. Assume I have a program that displays a list of dates on the screen. Let's call it `display.dates`. I then want to take that information, and count how many lines have been processed (I am not interested in seeing the intermediate output – all I want to know is how many lines were processed). What one-line command can I use?

9. I am working on an ASCII terminal, and I need to run a job that takes fifteen (15) minutes to execute. But I also have to write and send a piece of mail to my boss. I have no window system, and I need to leave in 15 minutes. How can I resolve my problem?

10. I'm in a `C` shell. How do I recall the last command I executed? How do I recall the last command that started with `la`? How do I change the `54` in my last command to `45`? How do I find out which `aliases` I am presently using?

11. You are not sure whether you are using the `C` or `Bourne` shell. How can you find out?

12. Why would one use SSH on their system?

Chapter 5

Basic Security

In computing and in any operating system, security is becoming more and more of an issue.

The following few sections will introduce basic security measures every user and system administrator should take.

5.1 File Permissions

File permissions have already been introduced in the `File System` chapter. USE them. If you have a file, or a directory which contains sensitive information, close it to everyone but yourself (using `chmod 700` *file*). If a file is to be run, but the contents should not be seen by any user, do not allow read permissions to anyone (use `chmod 700` *file*).

Many users completely close their home directory to other users, using the `chmod` command.

To system administrators: most files outside of `/tmp`, `/var/tmp` and the user areas should be write protected.

5.2 Passwords

- USE A PASSWORD.
- do not choose a password that exists in the dictionary file (`/usr/share/lib/dict/words`).
- change your password regularly.

- use a combination of letters and numbers, if possible.
- your password must be longer than five characters.
- do not write your password anywhere.
- do not share your password with everyone who asks for it.

5.3 Root Password

To system administrators:

- use the above rules.
- make sure every user uses a password.
- limit the number of people who know the root password, to at least one more than yourself, but not too many.
- use the `root` account only when performing system administration work.

5.4 SSH

SSH (Secure SHell) is a protocol used to provide secure connections between two hosts. The secure connection is provided by encrypting the entire session between the two systems.

Each system has a pair of *keys*: a *private* and a *public* key. The keys are created at the same time, so they do have a direct relationship between each other.

The *public* key is made available to anyone who requests it. It is used to encrypt messages to be sent to its original owner.

The *private* key is the only key able to decrypt messages produced by its own *public* key. Needless to say, the *private* key may only be seen/read by its own owner.

The result is that any system wanting to communicate with your system must have your *public* key. And your system must have the *public* key of any system you wish to contact.

This allows two systems to have private conversations, even if someone is eavesdropping on their conversation.

SSH is installed on all systems maintained by the Research Computing Support Group. In fact, as of May 2001, any session requiring a password with any UNIX system maintained by the Group will be required to be done using the SSH protocol. This includes file transfers and reading mail using remote clients.

5.5 TCP Wrappers

Traditionally, UNIX systems allow any host, from anywhere, to connect to its various services (such as `telnetd`, `ftpd`, `POP/IMAP`). Since the majority of NRC's systems are connected directly off the Internet, it means anyone may have access to the systems and their services.

To restrict who has access to which service, `TCP wrappers` are used to control which system has access to which service.

An access control file `/etc/hosts.allow` (or `/etc/hosts.deny`) contains a list of services, and a list of hosts/domains allowed (or denied) access to those services. Only hosts (and services) listed in `/etc/hosts.allow` are allowed execution by remote and local systems. A typical last line in the access crontrol file is

```
ALL : ALL : severity auth.crit : deny
```

which would deny access to all services/systems not mentioned prior to it.

`/etc/hosts.deny` would also typically have that same last line.

This allows us to greatly enhance our control on who has access to which service on which system.

5.6 Exercises

1. Why should I use a password on my account?
2. What should my password look like?
3. I am the system administrator for a few workstations. Should I give the `root` password to every user? Why, or why not? Should I always log in as `root`? Why, or why not?

Chapter 6

Commands I

UNIX was originally written for people using slow and clumsy teletype (hard-copy) terminals. The fewer the number of characters typed the better. For that reason, many UNIX commands are very short: two or three letters. The options, called *flags*, are also very short: usually one character.

The syntax for some commands will be described in this chapter along with some of the options, or flags. However, as this is only an introduction to UNIX, many flags for many commands will NOT be mentioned. It is the user's responsibility to verify which flags/options are available on their machine.

Short commands are usually not very mnemonic. But there are always tricks to help remember their meaning.

6.1 A Command is a File

A command is a file that can be executed simply by typing the file name. Most commands are in the `/usr/bin`, `/usr/bsd`, `/sbin`, `/usr/sbin`, or `/usr/local/bin` directories although they may be located anywhere in the directory structure.

A command is found by looking in the directories defined by the *path* environment variable (`path` or `PATH`, depending on which shell is used) and searching this structure.

Commands can be created by users. The only criteria is that the file (representing the command) be executable.

6.2 Syntax

The general UNIX syntax of a command is

command [-*flag(s)*] [*filename(s)*] (generic UNIX)
or
command [-*flag(s)*] [--*longflag(s)*] [*filename(s)*] (linux)

where:

- *command* is the filename representing the command.
- *flag(s)* are option(s) to the command. Most start with a minus (-) sign, followed by a number of single characters, each with a special meaning for the command.
- *longflag(s)* are more descriptive options to the command. Used almost exclusively with the linux distributions, they always start with two hyphens (--) and offer a better description of the option itself.
- *filename(s)* are files (input and/or output) to be used by the command.

6.3 Login Related Commands

This section will deal with how to log on to the UNIX system, and how to log out.

6.3.1 Logging On

Initially, a user needs to get a `userid` and, optionally, a password. The `userid` is typed in at the `login` prompt, which usually looks like:

```
login:
```

Following the logon, the password prompt will appear:

```
password:
```

When the password is typed, it will not be echoed back to the display terminal for security reasons.

The first time a user logs on, a password may not be required. In this case, the user will be logged on automatically.

Upon logging on, a number of special files will be executed. These will be discussed in the `Special Files` chapter in `Book II: Advanced Introduction (including Internet)`.

6.3.2 Changing Password

Upon logging onto UNIX for the first time, it is STRONGLY recommended that the user change his/her password, using the `passwd` command:

passwd

UNIX will ask for the old password, then the new one (twice), always with the echo turned off.

Some systems will provide restrictions on the password: IRIX will insist on a password of at least five characters and at least ones non-alphabetic character.

Also note that a maximum length of eight characters is used by the password algorithm: any password longer than eight characters will be truncated to eight.

6.3.3 Logging Out

Three commands can be used to log out of the UNIX session. These are: `logout`, `exit`, and `CTL-D`. Some systems may not accept the `CTL-D` or `exit`. They are roughly equivalent when logging out from the console.

6.4 Help

UNIX assumes the user knows what he/she is doing. For that reason, specific help files do not exist.

There is, however, a set of on-line manual pages which may or may not be on your particular system. The man-pages are accessed using the `man` command.

6.4.1 `man`: Manual Pages

man *man_page*
or
man -k *key_word*

where *man_page* is either a command, or a special file. `man` then displays documentation on the standard output, one screenful at a time. To see the next screenful, press the space bar (`man` works very similarly to the `more` command with a reduced functionality).

key_word is a keyword that is found in the description of a *man_page*. When

man -k *key_word*

is issued, all commands which will be listed contain in their description, the word *key_word*.

`man` pages are typically divided into eight sections; some Linux distributions may have a 9th one, specifically for kernel routines (very rarely used; its man pages even say section 9 is "obsolete"!).

The following table illustrates most man page sections on a Sun (mostly Berkeley based), on a Silicon Graphics (System V based) system, and on the SuSE 7.3 and Red-Hat 7.2 linux distribution:

Table content based on [4, p. 56].

6.4.2 Manuals on CDs

Since early 1993, most vendors include electronic versions of UNIX manuals with the OS release. Utilities to read them are included in the operating system.

The table below enumerates the vendor and the name of their on-line manual utility.

Most Linux distributions do not include such a tool. But the `/usr/share/doc` directory contains documentation on many of the packages, many of which is in HTML so can be read with any Web Browser.

The `Linux Documentation Project`, located on the web at

`http://www.linux.org/docs/`

Content	SUN	SGI	Linux
User commands.	1	1	1
Administrative commands.	1M	1M	8
System calls.	2	2	2
C routines.	3	3C	3
Fortran routines.	3	3F	3
Graphics routines.	N/A	3G	N/A
Streams routines.	3V	3S	3
X11 routines.	3	3X11	3
Audio routines.	N/A	3A	N/A
Special files.	4	7	4
Configuration files.	5	4	5
Games.	6	N/A	6
Demos.	6	6D	N/A
Maintenance.	8	8	8

Table 6.1: Sun, SGI and Linux man page sections.

offers generic linux documentation on-line.

The Research Computing Support Group also has linux documentation on their web site at

`http://www.nrc.ca/imsb/rcsg/linux/documentation/`

(access to that site may be limited to internal NRC only).

Vendor	Product Name	Command
Hewlett Packard	Laserrom	`lrom`
IBM	InfoExplorer	`info`
Silicon Graphics	Insight	`insight`
Sun Microsystems	AnswerBook	`answerbook`
Digital	Bookreader	`bookreader`

Table 6.2: Tools to read "CD" manual set.

6.5 File system Commands: Directories

Now that the UNIX tree-structured file system has been introduced, it would be useful to know how to find our way through the system and how to create files, delete them and rename them.

This section deals with commands directly related to directories: how to create or remove them, and how to move from one to another.

The next section will deal strictly with file manipulations.

6.5.1 `cd`: Change Directory

To move from one directory to another the `cd` command is used. Use as:

cd [*directory*]

directory may be an `absolute`, or a `relative` pathname. Or it may be nothing at all: if **cd** is used by itself, the user returns to his/her `home directory`.

An `absolute`—or full—pathname starts with `/`. In other words, the path taken to get to the directory is given from the top of the tree, through all the nodes, to the destination directory.

The `relative` pathname of the directory gives the route taken *from the current directory*, to the destination directory. Note that `.` (dot) means this current directory, `..` (dot dot) means the previous directory, and that `~` (tilde) means the user's home directory. For example

cd ../testdir

means go up one directory, then down into the `testdir` directory at that node.

cd (by itself) is equivalent to **cd ~**.

6.5.2 `mkdir`: Make Directory

To create a new directory, `mkdir` is used, as

mkdir [**-p**] *directory*

-p means create parent directories, if needed. *directory* may be an `absolute` or `relative` directory name.

The user must have write permission on the parent directory to create a new directory.

6.5.3 `rmdir`: Remove Directory

A directory can be removed with

rmdir *directory*

and again, *directory* may be an `absolute` or `relative` path name.

The directory being deleted must be empty (i.e., cannot have any child directories, or files), and the user must have write permission.

6.5.4 `pwd`: Print Working Directory

The command `pwd` is used to display the full pathname of the current directory; the display is always in the `absolute` format.

Example:

```
prompt> pwd
/usr/people/cantin/docs/unix
```

6.5.5 `cp`: Copy

To copy the contents of a directory into another directory, the following is used:

cp -r [**-ip**] *directory1 directory2*
or
cp -R [**-ip**] *directory1 directory2* (not linux)
or
cp -r [**-ipd**] *directory1 directory2* (linux only)
or
cp --recursive [**-ipd**] *directory1 directory2* (linux only)

where *directory1* and *directory2* are directories.

The contents of *directory1* will be copied into *directory2*. If *directory2* does not exist if will be created. If it exists, a copy of *directory1* will be created as a subdirectory of *directory2*. In both cases, a new directory will be created.

`-r` (recursive) is a recursive copy. When a directory is encountered, its files are also copied.

`-R` (recursive) is a recursive copy. When a directory is encountered, its files are also copied.

If a file/directory is a symbolic link, the link is copied, not the content of the file.

`-d` (preserve link) is used by linux. If the file/directory is a symblic link, it preserves that link.

`-i` (interactive) will prompt the user for confirmation if a file will be overwritten during the copy.

`-p` (preserve) will keep the time-stamps and permissions of the original files.

6.5.6 `mv`: Move

In UNIX, `mv` means move, or rename.

mv [**-i**] *directory1 directory2*

where *directory1* and *directory2* are directories.

This command will simply rename *directory1* to *directory2*. If *directory2* exists, the files of *directory1* will be moved into *directory2*.

`-i` will prompt the user for confirmation if a file is to be overwritten.

6.6 File system Commands: Files

By this time, the user should know how to find his/her way through the file system. The contents of the major directories should be understood.

The user should also be able to create new directories, and remove them.

This section is concerned with the creation, deletion, and manipulation of files. By the end of this section, the user should be able to identify all files within a directory, create new ones, remove others, display the contents of some, rename others, and more.

6.6.1 `ls`: Listing

The contents of a directory can be displayed using the `ls` command:

ls [**-alRC**] *filename...*

where *filename...* is one or more file or directory names.

`-a` list all files, including those beginning with "`.`" and "`..`".

`-a` may be replaced with `--all` on linux.

`-l` produce a long listing.

`-R` produce a recursive listing.

`-R` may be replaced with `--recursive` on linux.

`-C` force multi-column output.

Examples of the above flags follow.

Note: many more flags are available. For more information, see the documentation relating to your system or use `man ls`.

ls by itself lists the content of the current working directory, in alphabetical order:

```
prompt> ls

course.tex        course.toc        unixug
```

The `-a` (all) option, lists all files, including those beginning with a dot (.). The current directory is represented as . (dot) and the parent directory, by .. (dot dot):

`prompt>` **ls -a** (general UNIX)
or `prompt>` **ls --all** (linux)

```
.            ..           course.tex   course.toc   unixug
```

The `-l` (long) flag will produce a `long` listing:

```
prompt> ls -l

total 303
-rw-r--r--  1 cantin  saoscs  19347 Dec 11 10:12 course.tex
-rw-r--r--  1 cantin  saoscs   2520 Dec 11 09:53 course.toc
-rw-r--r--  1 cantin  saoscs   1978 Dec  7 09:14 unixug
```

The first line of the listing gives the total storage area taken by this directory, in kilobytes.

The first character of the first field indicates the type of file (`-` means an ASCII file, `d` a directory, `l` a symbolic link). The next nine characters indicate the permissions on the file. The next field is the number of links, i.e. the number of copies of that file on the file system (normally 1). The third field shows the `owner`, followed by the `group` name of the owner; the fifth, the size of the file (in bytes), the sixth the last modified time, and the last field the name of the file.

Any combination of flags is allowed:

```
prompt> ls -la

total 305
drwxr-xr-x  2 cantin   saoscs     512 Dec 11 10:14 .
drwxr--r--  9 cantin   saoscs     512 Dec  5 12:09 ..
-rw-r--r--  1 cantin   saoscs   19347 Dec 11 10:12 course.tex
-rw-r--r--  1 cantin   saoscs    2520 Dec 11 09:53 course.toc
-rw-r--r--  1 cantin   saoscs    1978 Dec  7 09:14 unixug
```

6.6.2 `cp`: Copy

Files can be copied onto other files using the `cp` command:

cp [**-ipR**] *filename1 filename2* (not linux)
or
cp [**-ipRd**] *filename1 filename2* (linux only)

where *filename1* and *filename2* are files, not directories.

`-i` (interactive) will prompt the user if *filename1* will overwrite *filename2*.

`-i` may be replaced by `--interactive` with linux.

`-p` (preserve) will keep the time-stamps and permissions of the original file.

`-R` (recursive) is a recursive copy. When a directory is encountered, its files are also copied.

If a file/directory is a symbolic link, the link is copied, not the content of the file.

`-R` may be replaced by `--recursive` with linux.

`-d` (preserve link) is used by linux. If the file/directory is a symblic link, it preserves that link.

filename1 will be copied onto *filename2*. If *filename2* does not exist, it will be created. If it exists, its contents will be overwritten!

A second method, involving directories, has been discussed in the previous section.

Files can also be copied using the following syntax:

cp [**-ip**] *filename... directory*

where *filename* may be one or more files. These files will be copied into *directory*.

6.6.3 `mv`: Move

mv [**-i**] *filename1 filename2*

means change the name of *filename1* to *filename2*. If *filename2* already exists, `mv` overwrites the old file, unless the `-i` (or `--interactive` with linux) option is used, in which case a confirmation from the user will be requested.

mv [**-i**] *filename... directory*

will move *filename...*, i.e., a number of files, into directory *directory*. If a filename already exists within *directory*, it will be overwritten unless the `-i` option was used, in which case a confirmation by the user will be needed.

Write permission on *directory* is essential.

6.6.4 `ln`: Link

ln [**-s**] *existing_file new_name*

allows you to either give a file a second name (no flags) or to create a new file *pointing* to the original file (with the `-s` option).

When used with no option, `ln` modifies the inode of *existing_file* adding a new name to it (an *inode* is the internal description of a file; it contains all the file's attributes including its name(s), access times, ownership, etc). From that point on, that file may be accessed using either names.

If one of the files is removed using one of its given names, only the file name is removed, not the file content. For the file content to be removed, all names listed in the inode must be removed.

A file may have many different names, all of which may be in different directories. But they must all be within the filesystem.

If one of the files is removed, only its name is removed from the list of names representing that file. Only when the last name in the list is removed, is the actual file (content) removed.

This is often refered to as a `hard link`.

When the `-s` option is used, a new file is created. Its content is a *symbolic link* pointing to *existing_file*.

new_name may be removed without any effect to *existing_file*. But if *existing_file* is removed, *new_file* now points to a non-existing location.

`ls -l` illustrates some differences between a hard link, and a symbolic link:

```
82 nickel,cantin> ls -l
total 320
-rw-r--r--    4 cantin   rcsg       78519 Aug  5  1992 bigcat.ps
lrwxr-xr-x    1 cantin   rcsg          21 May 20 14:23 cat.ps -> ../../imag
-rw-r--r--    4 cantin   rcsg       78519 Aug  5  1992 largecat.ps
lrwxr-xr-x    1 cantin   rcsg          35 May 20 14:31 lastcat.ps -> /usr2/
-rw-r--r--    4 cantin   rcsg       78519 Aug  5  1992 realcat.ps
lrwxr-xr-x    1 cantin   rcsg           9 May 20 14:24 wild.ps -> bigcat.ps
83 nickel,cantin>
```

The files werecreated with the following commands:

```
ln -s ../../images/tiger.ps cat.ps
ln ../../images/tiger.ps bigcat.ps
ln -s bigcat.ps wild.ps
ln bigcat.ps largecat.ps
ln  cantin/images/tiger.ps realcat.ps
ln -s  cantin/images/tiger.ps lastcat.ps
```

Note that all file created with the `ln` command, have the same creation date as the original `tiger.ps` file. This is because it IS the same file.

The second column in the `ls -l` output also shows a "4" for three of the files. They also have the same attributes (including modification time, size, permissions). If you were to perform the `ls -li` command in that directory, you would notice that all files do indeed have the same *inode number*. And that "4" actually represents how many different names are represented by that file.

If one of the file would be removed, "4" would become "3".

The files created with the `ln -s` command are all very small. That is because their content is the symbolic link seen on the last field of the `ls -l` command. Also note that the first character of the listing, for those files, is a "l".

That "l" means the file type is a symbolic link.

6.6.5 `touch`: Update

touch *filename...*

where *filename...* is one or more files.

If *filename...* does not exist `touch` creates it. Its contents will be the `null` string (i.e., no contents).

If *filename...* already exists, `touch` will update its time stamp to the current time.

6.6.6 `rm`: Remove

rm [**-rf**] *filename...*

where *filename...* is one or more files.

`-r` : "remove recursively". In other words, if the file to be removed is a directory, it will delete it along with all its files and/or sub-directories and their contents. Be careful with the `-r` option.

`-f` : do not output any error messages; do not prompt and remove even if file is write-protected (but file must still be owned by user).

6.6.7 `cat`: Concatenate

cat *filename...*

where *filename...* may be one or more files.

`cat` displays the content of *filename...* on the standard output. If more than one file is listed, they are displayed one after the other.

Output redirection may be used to copy the contents of files into another file. An example is

cat *.c > tot.cprog

This will copy the contents of all files ending with `.c`, into file `tot.cprog`. The order of the files within `tot.cprog` will be alphabetical.

`cat` can be used to append one file to another:

cat file2 >> file1

will append `file2` to `file1`.

6.6.8 `more`: Browser

more *filename...*

This command displays the named file(s) on the standard output, but one screenful at a time. At the end of each screenful, the percentage of the file already shown is displayed. Pressing the space bar displays the next screenful, while pressing the carriage return key moves forward only one line. A number followed by a carriage return displays this additional number of lines.

If more than one file is specified, a heading preceding each file is displayed, to identify the file that will follow:

```
prompt> more file1 file2
::::::::::::::
file1
::::::::::::::
this is the first line in file1
this is the second line in the file
::::::::::::::
file2
::::::::::::::
this is the beginning of file2
more lines here...
end of file2
```

To go back one screen, the "b" command is used. To stop (quit), press "q".

An alternative command to more is page. Their usage is identical.

6.6.9 head: Header

head [**-*n***] *filename...*

where the default for n is 10.

head copies the first n lines of each file to the standard output. If *filename...* lists more than one file,

==>*filename*<==

is displayed before each file:

```
prompt> head file1 file2
==> file1 <==
this is file1


==> file2 <==
this is file2
```

6.6.10 tail: Tail End

As head is used to display the heading of a file, tail is used to display the end of a file.

tail [+|-*n*[**lbc**]] *filename*

NOTE: only one filename may be used at once.

If `+` is used, the display starts *n* lines, characters, or units of blocks (depending which of `l`, `b`, or `c` is used) after the beginning of the file.

If `-` is used, the display starts at *n* lines, characters, or units of blocks from the end of the file.

The default is for `tail` to show the last ten lines of the file.

6.6.11 `wc`: Word Count

wc [**-lwc**] *filename...*

The default is to use all three flags:

`l` – lines.

`w` – words.

`c` – characters.

With linux the `l`, `w` and `c` flags may be replaced with the `lines`, `words` and `bytes` longflags, respectively.

Here is an example:

```
prompt> wc course.tex course.aux

    810    4137    27849 course.tex
     51     200     5095 course.aux
    861    4337    32944 total
```

The first field is the line count, the second field is the word count, the third field is the character count (size of the file) and the last field is the name of the file (only if more than one file is used).

6.6.12 `diff`: Difference

diff [**-bitw**] *old_file new_file*

displays the differences between *old_file* and *new_file* on the standard output.

`-b` ignores trailing blanks.

`-b` may be replaced with `--ignore-space-change` on linux.

`-i` ignores the case of the letters (upper is treated the same as lower case).

`-i` may be replaced with `--ignore-case` on linux.

`-t` expands tab characters into blanks.

`-t` may be replaced with `--expand-tabs` on linux.

`-w` ,in UNIX, ignores all blank characters.

`-w`, in linux, makes an assumption on the current screen's width.

6.6.13 `file`: Type of File

file *filename...* (general UNIX)
or
file [**-i**] *filename...* (linux)

where `-i` (or `--mime`) would display the mime type string of the file instead of the file type (ex: `text/plain`, `application/pdf`).

`file` displays the type definition of file *filename...*, in a two-column format: the first column shows the file name, and the second shows the file type (examples are ascii, data, C-shell commands, PostScript document, directory, commands, text, assembler, etc.).

```
prompt> file course.tex course.ps
course.tex:         ascii text
course.ps:          PostScript document
```

If the parameter to the `file` command is `core`, `file` will return the name of the executable file which caused the `core` file to be created (a `core` file is a file containing the memory content used by a program that crashed; that file may be used to debug the program).

6.7 Printer Commands (Berkeley; `lpr`/`lprm`)

There are two types of printer commands: Berkeley based, and System V based. They are named as such because of the origin of the UNIX version they were first written for.

Although most systems now support both types, linux tends to use Berkeley based printing.

6.7.1 `lpr`: Line Printer

lpr [**-P***printer*] *filename...*

`lpr` sends a file to the printer spooling area, where it will be processed and printed.

The printer referred to by the `lpr` command may also contain *filters* (most of the time, used transparently). A *filter* is a program that formats data according to the contents of the file. For example, most PostScript printers, such as the Sun LaserWriter and the Apple LaserWriter, have a PostScript filter, which means that any file sent to the printer will be transformed into PostScript format (this is because the printer ONLY understands PostScript, a page formatting language). If an actual PostScript file is to be sent to the printer, a flag would be used to bypass that specific filter.

The default printer destination can be changed by creating an environment variable called `PRINTER`. Its content would be the name of the new default printer. Also, any available printer may be selected on the command line using the `-P`*`printer`* flag.

NOTE: before using the printer connected to your system, contact your system administrator to find out what type of printer it is, and what type of file it expects. You may also look at the content of `/etc/printcap`: it is a configuration file used by the BSD LPR set of commands used to describe the various printers on the system.

6.7.2 `lpq`: Line Printer Queue, Statistics

lpq [**-P***printer*] [*username*]

`lpq` alone simply displays the queue of the default printer. The output contains the rank of the file to be printed (First In First Out list), the owner of the file, the job number, the file name, and the size of the file.

```
prompt> lpq

lp is ready and printing
Rank   Owner      Job  Files                                 Total Size
active cantin     625  scourse.ps                            38467 bytes
1st    cantin     626  history.ps                            44529 bytes
```

To list the queue for printer *printer*, use

```
prompt> lpq -Pprinter
```

6.7.3 `lprm`: Line Printer Remove

To remove a job from the printer queue, simply type:

lprm [**-P***printer*] *job*

where *job* is the job number in the printer spooler and *printer* refers to the spool for printer *printer*.

To remove the file `history.ps` from the previous section,

```
prompt> lprm 626
dfA626neon dequeued
cfA626neon dequeued
```

The output mean job 626 queued on `neon` was taken off the print queue.

6.8 Printer Commands (System V; `lp/cancel`)

This section refers to printer commands for System V based systems.

6.8.1 `lp`: Line Printer

lp [**-d***printer*] *filename...*

`lp` sends a file to the printer spooling area, where it will be processed and printed.

The printer referred to by the `lp` command may contain *filters*. A *filter* is a program that formats data according to the contents of the file. For example, most PostScript printers, such as the Sun LaserWriter and the Apple LaserWriter, have a PostScript filter, which means that any file sent to the printer will be transformed into PostScript format (this is because the printer ONLY understands PostScript, a page formatting language). If a true PostScript file is to be sent to the printer, a flag would be used to bypass that specific filter.

The environment variable `LPDEST` could contain the name of the default printer. Any available printer is also accessible with the `-d`*printer* flag.

6.8.2 `lpstat`: Line Printer Queue, Statistics

lpstat [**-t**] [*job*]

`lpstat` displays the queues for the printers defined on the system. The `-t` option requires that all status information for all printers on the system be displayed.

```
prompt> lpstat -t
scheduler is running
system default destination:  laser
device for laser:  /dev/null
device for decwriter:  /dev/ttyf56
laser accepting requests since Oct 29 13:27
decwriter accepting requests since Nov 20 11:08
printer laser is idle.  enabled since Oct 29 13:27
```

6.8.3 cancel: Line Printer Remove

To remove a job from the printer queue, simply type:

cancel *job*

where *job* is the job number displayed when `lpstat` was issued.

```
prompt> lp course.ps
request id is laser-592 (1 file)
prompt> cancel laser-592
request "laser-592" cancelled
```

6.9 Printer Commands (linux; kprinter)

linux understands both the Berkely `lpr` and the System V (`lp`) set of printer commands. Most distributions include a graphical-based utility allowing people more flexibility in their printing.

`kprinter`

allows people to select which printer to use, and which properties to enable, all with the click of a mouse. It is part of the CUPS (Common Unix Printing System) package, which also understands the `lpr` and `lp` commands.

The SuSE 9.1 version of the `kprinter` interface looks like that in figure 6.1.

6.10 User Related Commands

Up to now, the commands were file and/or directory related. This section will treat user related commands: how to find out who is logged on to the system, how to exchange messages with another user.

6.10.1 who: Who is On

Typing

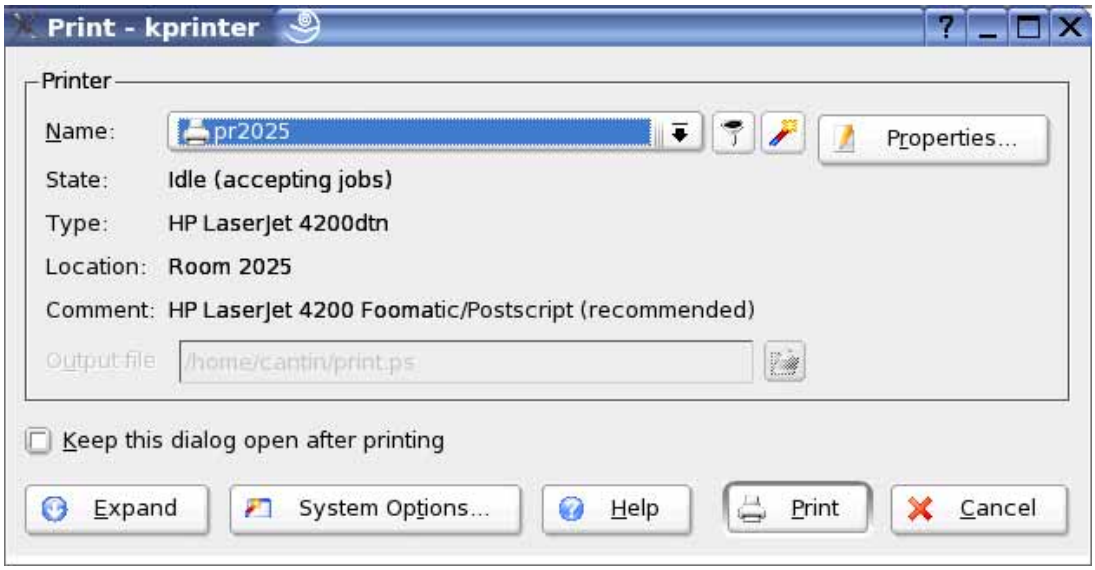

Figure 6.1: kprinter interface

who

will display on the standard output the logon names of the users on the system, and the time at which they logged on.

```
prompt> who

cantin    ttyq1    Jan 12 08:43
cantin    ttyq3    Jan 12 08:43
proulxm   ttym1   Jan 12 09:08
```

tells us that user `cantin` is logged on two times: once on the virtual device `ttyq1` and once in another window, `ttyq3`.

`proulx` is logged in from a modem, or serial port.

If the output had included

```
cantin   ttyq0   Jan 12 08:46   (nickel.sao.nrc.ca)
```

this would mean that `cantin` is logged onto the current machine remotely, from a system whose name is `nickel.sao.nrc.ca`.

6.10.2 `who am i`, `whoami`: Who Am I

who am i

tells me who I am.

```
prompt> who am i

nickel!cantin   ttyq0   Jan 12 08:46
```

On the other hand,

whoami

displays the `effective` current username.

```
prompt> whoami
cantin
```

6.11 Other Miscellaneous Commands

This section deals with a number of miscellaneous commands.

6.11.1 `date`: Display Date

date

displays the date and time on the standard output.
Example:

```
prompt> date
Thu Feb 23 14:12:39 EST 1995
```

`date` can also be used to display specific fields of the date. The next example shows how to display the current hour and minute:

```
prompt> date "+%H %M"
14 12
```

6.11.2 `clear`: Clear the Screen

This will clear the screen, bringing the cursor to the top left corner of the monitor.

6.12 Exercises

1. How do you give option(s) to a command?

2. You would like to change your password. How do you do it? Change it to `my pssd`. Logout, then log back in to make sure it worked. Change it back to something only you know.

3. You would like to use the `rm` command, but do not know how to use it. You do not have system manuals close by. What command can you use to get help? Try it.

4. Now that you know how to move within the directory structure of your system, go and see where the dictionary resides. Which directory is it in? How big is the `words` file? Who owns it?

5. How do you find out which directory you are currently in? Issue the command and see what happens.

6. Assume you are lost. How do you return to your `home` directory? Assuming you don't remember the full pathname of your `home` directory, how can you get there?

7. In your account, create a new directory, called `temp.dir`. Copy the two files `/etc/hosts` and `/etc/fstab` into it as `hosts` and `fstab` respectively.

8. Again from your `home` directory, create a second directory, called `temp2.dir`. Copy the content of `temp.dir` into it.

9. Remove the entire `temp.dir` and `temp2.dir`.

10. You have a file called `list.names` in your directory. How can you find out the last time you wrote to that file?

11. Everyone has files beginning with a dot (.). These are called *hidden* files. How can you list them?

12. Create a file called `myfile`, with length zero.

13. Now rename `myfile` to `hisfile`.

14. Do a long listing of all files in the `/bin` directory and place it into file `hisfile`.

15. How can you look at the content of the file, one screenful at a time?

16. You want to see the first twenty lines of that file. Which command is used? Do it.

17. How about the last fifteen lines?

18. Who is logged on to the system at the moment?

19. You want to communicate interactively with one of those users. Which commands could you use?

20. You would like to print a file. Which command would you use?

21. You send a few files to a printer. You now want to remove your files from teh print queue. Which commands would you use?

Chapter 7

Editors

Now that the user is familiar with some of the commands, UNIX standard editors will be introduced. The first one is `ed`, a line editor used on line-mode terminals. For terminals with full-screen capability, `vi` is more suitable.

NOTE that all UNIX editors use `buffers` as work areas in which they manipulate the files. The original file is updated only when the `buffer` is written to the file, which is done by the user: if the user simply "quits", only the changes prior to the last `write` will be reflected in the file!!! For that reason, you must first `save` the file, then `exit` the editor.

7.1 "ed" Editor

`ed` is the most basic UNIX editor, and is found in every UNIX system. It operates in line-mode, which means that

> the basic unit for change is a line (a string of characters terminated by a newline character). You can give commands to the editor to perform various operations on the lines: lines can be printed (displayed); lines can be changed; new lines can be inserted; existing lines can be deleted; lines can be moved or copied to a different place within the file; substitutions of character strings can be made within a line or group of lines [7, p. 186].

Most people use `ed` only if the terminal they are using at the time cannot support a `screen` editor. For this reason, only the most basic commands will be introduced here. More information can be found in [7, pp. 187-219].

7.1.1 Accessing `ed`

ed *filename*

where *filename* must exist. If *filename* does not exist, use the `touch` command to create it.

`ed` will then display the number of characters (*num*) in the file and wait (there is no prompt). This means that `ed` loaded *num* bytes into the editing `buffer`. `ed` is now sitting at the end of the document, waiting for the user to tell it what to do.

Changes made while in `ed` are made to the contents of the buffer and are not permanent until a `w` write command is issued.

7.1.2 Moving Within a File

Typing a number followed by a carriage return will bring you to that line number in the file. Hence, to go to the top of the file, type

1

To go to line 5 of the file, type

5

To go up *num* lines, type

-*num*

To go down *num* lines, type

+*num*

If `ed` can't get you to that line, it will respond with a

?

7.1.3 Finding a Pattern

To find one specific string, use

/string/

where *string* is the literal to be found. The search starts at the current line going down in the file to the end, then goes back to the top of the file down to the current line. The search will stop upon finding *string*.

7.1.4 "s" Substitute

To substitute a string for another, the format is

[*line_b*[,*line_e*]]**s**/*string_1* / *string_2*/

where *line_b* and *line_e* are the range of the operation, and *string_1* and *string_2* are the string-to-be-replaced and new-string respectively.

For example, to substitute the word `mine` for `his` on the first fifteen lines of the document, use

1,15s/mine/his/

7.1.5 "c" Change Line

c

will enter `input mode`, and change the `current line` to the text that will be entered. More lines may be added. A dot (.) on a line by itself must be entered to leave input mode.

7.1.6 "a" Append Text

To append text immediately *after* the last displayed line, type

a

`ed` will then wait for the user to input text. It will leave `append` mode when a dot (.) is typed as the *first and only* character on a line, or when <CTL-D> is typed.

7.1.7 "i" Input Text

i

is the same as append, but puts the text *before* the current line.

A dot (.) on a line by itself must be typed to leave input mode.

7.1.8 "." Current Line

Typing a dot (.) will display the current line.

7.1.9 "p" Print Line(s)

[*line_b*[,*line_e*]]**p**

where *line_b* and *line_e* are respectively the beginning line and the end line to be displayed. *line_b* must be smaller than *line_e*.

So, to print (display) lines 3 to 6, one would use

3,6p

The line numbers are optional. If no line numbers are given, the current line will be displayed.

To display only one line, the command is simply

*num***p**

A dollar sign ($) is used to represent the last line number. Hence, to display lines 6 to the end of the file, type

6,$p

NOTE: the `l` command is similar to `p` with the addition that it displays non-printable characters.

7.1.10 "d" Delete

d

will delete the current line. If a range (x,x+y) precedes `d`, then y+1 lines will be deleted.

7.1.11 "w" Write

w

writes the content of the buffer into the file being edited, replacing the original version.

7.1.12 "q" Quit

Typing

q

ends the session, but only if no changes have been made in the buffer since the last write to the file. This is to remind the user that perhaps the changes made should be saved. In this case, ed comes back with

?

If **q** is entered a second time, the command will then be executed. Changes since the last write will not be saved, and the user will return to the calling shell.

7.2 "vi" Visual Editor – Introduction

vi is a screen editor

> where a portion of the file is displayed on the terminal screen, and the cursor can be moved around the screen to indicate where you want to make changes. You can select which part of the file you want to have displayed. Screen editors are also called *display* editors, or *visual* editors . vi is one of the more popular screen editors that run on the UNIX system [7, p. 186].

This editor relies heavily on the cursor keys to move around. Again, only the basic commands will be introduced.

Once in vi, commands must be invoked for anything to happen. These commands can be anything from :q to i to <CTL-f>. Following the philosophy of UNIX, vi assumes the user knows what he/she is doing!!!

NOTE: if a command puts the user into a mode (input, append, replace), the escape key must be pressed to get OUT of that mode.

For more information on vi, see [7, pp. 264-310].

7.2.1 Invoking vi

vi *filename*

will put *filename* into a buffer, and display the file on the screen. If the file is larger than the screen can display, the screen will act as a `window` into the file. At the beginning of a session, the screen will display the first part of the file.

If *filename* does not exist, `vi` will create it.

Upon entry to `vi`, the bottom of the screen will print the name of the file being edited, the number of lines in the file, and the size of the file (in characters).

7.2.2 `vi`, `command` and `input` modes

One of the most important aspects to remember about `vi` is that most of the commands fall into one of three modes:

1. `vi` mode: in this mode, most keys on the keyboard are defined to be a specific command. As the key or key sequence is issued, that command is executed.

   This is the mode `vi` starts in.

   At any time, pressing the <ESC> key returns the user to `vi` mode.

2. `command` mode: to reach that mode, one must first be in `vi` mode, then issue a colon (":"). That same colon will appear at the bottom left corner of the screen. Then the command may be issued following the colon.

   One exception to this rule is the search command; a forward slash is issued instead of the colon.

3. `input` mode: this is where most users expect an editor to start. This "mode" actually refers to commands issued from `vi` mode but that allow the user to start inputing data into the file.

The `vi` commands introduced will be grouped in the three major mode types. Table 7.1 displays the most basic commands.

7.2.3 "`vi`" visual Editor – `vi mode`

7.2.4 Moving the Cursor

To move around in the file, use the arrow keys: the up arrow will move the cursor up one line, the down arrow down one line. The right arrow will cause

`vi` mode	`input` mode <ESC> to end input	`command` mode
← ↓ ↑ → - cursor	i - insert	:q - quit
h j k l - cursor	a - append	:q! - quit no save
CTL-f - forward screen	A - append at EOL	:w - write
CTL-b - backward screen	O - open line	:wq - write and quit
G - end of file	r - replace character	:*num* goto line *num*
x - delete character	R - overwrite	/*str* - find *str*
dw - delete word		:set all - vi settings
dd - delete line		:r *file* - import file
yy - copy line in buffer		
D - delete to EOL		
p - paste/put buffer		
u - undo last command		
CTL-r - redo last undo (linux/`vim`)		
. - repeat last editing command		
n - find next occurrence of string		
cw - change word		
# command - repeate command # times		

Table 7.1: Summary of `vi` Commands.

the cursor to move one position to the right. From the end of a line, it will go to the beginning of the next line.

The left arrow will move the cursor one position to the left. If the cursor was at the beginning of a line, it will not move (the same as the up arrow when the cursor is already at the beginning of the file, and the down arrow when the cursor is already at the bottom of the file).

If *number* is typed immediately before pressing the arrow key, the position of the cursor will move *number* positions in the direction of the arrow. If *number* was too large, a beep will be heard, and nothing will happen.

For terminals with no functional arrow keys, four keys will move the cursor around:

h left arrow (⇐).

j down arrow (⇓).

k up arrow (⇑).

l right arrow (⇒).

7.2.5 "^f" Forward One Screen

^f <CTL-f> will move the user forward one screenful in the file.

7.2.6 "^b" Backwards One Screen

^b <CTL-b> will move the window backwards (up) one screenful in the file.

7.2.7 "G" End of File

G will move the window to the end of the file. Note, again, that `vi`, like any other UNIX utility, is always case sensitive.

7.2.8 "x, d" Delete Character

x or **d** *space* (`d`, followed by a space) will delete one character. To delete many characters in a row, *num* **x** or *num* **d** *space*.

`dw` would delete to the end of the word.

7.2.9 "dd" Delete Line

[*num*] **dd** will delete *num* lines beginning at the current line. The default *num* is 1.

`D` deletes until the end of the line.

7.2.10 "yy" Copy Line in buffer

[*num*] **yy** will copy *num* lines beginning at the current line, into a buffer. The default *num* is 1.

7.2.11 "p" Put Buffer

When a line is deleted with `dd` (*num* `dd`), these lines are copied into a special buffer. **p** will put that buffer *after* the current line.

Note that the contents of that buffer are not erased; they can be put (recovered) as many times as desired.

7.2.12 "u" Undo

This is a very useful command. It cancels the effect of the previously executed command.

`vim`, a `vi` clone used mosly on linux distibutions, has an unlimitted *undo*. To redo the undone change, use `CTL-R`.

7.2.13 "." (dot) Repeat

(dot) by itself will cause the last editing command to be repeated. That command may be `dd`, or `i` followed by some text, or a pattern find, or any legal command.

Cursor movement commands will not be repeated by a "dot".

7.2.14 "vi" visual Editor – command mode

7.2.15 ":q" Quit

When in `vi`, typing

:

will bring the user into command mode. As the `:` is typed, it will be displayed on the last line of the screen, and `vi` will wait for a command to be typed.

`q` is such a command. This will exit `vi`, if no changes have been made since the last write-to-file command.

`:q!` (with an exclamation mark) will exit, even if the buffer has not been written to the file.

7.2.16 ":w" Write

:w

will cause the contents of the buffer to be written to the file.

`:wq` will write the buffer to the file, and exit to the calling shell.

7.2.17 "`:r`" Read

:r *filename*

will bring the content of *filename* beginning at the cursor position.

7.2.18 "`:`*num*" Line Number

:*num*

will bring the cursor to line *num*.

7.2.19 "*/string/*" Finding a Pattern

To find a specific string, use

/string/

where *string* is the literal to be found.

Upon finding *string*, the cursor will position itself at the beginning of the literal.

7.2.20 "`n`" Next

After searching and finding a string (using the forward slash "/"), `n` will find the next occurrence of that same string.

7.2.21 `:set all`

This command displays a number of variables that can be altered to configure the editor. An example of such a variable is `set number`, which would then number the lines in the buffer.

7.2.22 "`vi`" visual Editor – `input mode`

7.2.23 "`i`" Insert Mode

Pressing **i** will bring the user into input mode. Anything typed after that will be inserted *before* the original cursor position.

ESC (escape) exits the *insert* mode.

7.2.24 "a" Append to Character Mode

a will enter append mode, with the text inserted *after* the original cursor position.

ESC (escape) exits the *append* mode.

7.2.25 "A" Append to Line Mode

A will bring the cursor to the end of the current line, and enter *append* mode.

ESC (escape) exits the *append* mode.

Similar to that command `O` starts a new line of input.

7.2.26 "r" Replace Character

To replace one character, **r** followed by the new character will replace the current character (where the cursor is) with the newly typed one.

7.2.27 "R" Replace Characters

To replace a string of characters, **R** will enter *replace* mode, and anything typed will overwrite the existing characters.

ESC (escape) exits the *replace* mode.

7.2.28 "cw" Change Word

To replace a word, bring the cursor to the beginning of the word, then issue **cw**. From that point on, the current word will be replaced with the input, until the ESC (escape) character is entered.

7.3 Other Editors

`vi` and `ed` are the two editors included in most UNIX systems. Other editors may be included with UNIX, or can be separately purchased. The next few sections will indicate some of these editors.

7.3.1 GNU Emacs

GNU (Gnu's Not Un∗x) Emacs is a public domain editor, available from MIT (prep.ai.mit.edu or 18.71.0.38), or from anyone who has a copy of the software.

GNU Emacs is a full screen editor, which relies on control keys for commands.

7.3.2 textedit

textedit is Sun's GUI (Graphical User Interface) editor. It works only inside the window system (Open Look), and relies heavily on the mouse.

textedit is easier to use than most other editors because it follows the "point-and-click" concept.

7.3.3 jot

jot is Silicon Graphics' equivalent of textedit.

7.3.4 vuepad

vuepad is available on HP's and is mouse based (point and click).

7.3.5 nedit

This is an X-based, point and click editor, in the public domain. Most servers maintained by the Research Computing Support Group have it installed.

7.3.6 pico

This is another public domain editor. It is ASCII based, and very easy to use. As with nedit, it is installed on most servers maintained by the Research Computing Support Group.

7.4 Exercises

1. Assume you have a standard UNIX system, and that you are logged on to it via a VT100 terminal. Which editors are available to you? Which

should you use?

Chapter 8

Electronic Mail

8.1 Mail at NRC

Officially, NRC employees use the central Microsoft Exchange service offered by the Messenging Group of the Information Management Services Branch (IMSB) based at M-60. That mail system is based on MS Windows Operating System.

Many researchers still prefer to have their mail delivered to a linux/UNIX mail server instead, and use their own mail client to access their mail. GUI-based mail clients used to access their linux/UNIX mail include Netscape, MS Outlook and Eudora. ASCII-based mail clients include BSD Mail, ELM and PINE.

This chapter will cover generic mail concepts, then will go on to describe some of those concepts within the BSD, ELM and PINE mail interfaces.

8.2 Internet *node*: Machine Naming Convention

A *node* is the name of a computer. The computer may be a PC, a workstation, a mini, a mainframe, or a supercomputer. An Internet *node* can be any machine in the world, as long as it is on the Internet network, which NRC is a member (the Internet is discussed in the `Networking` section of `Book II: Advanced Introduction`).

The convention for a *node* on the Internet network is:

machine.section.organization.type_of_organization

where `type_of_organization` may be `edu`, `mil`, `gov`, `com`, `org`, `net` or, outside of the United States, *country_code*.

As of 2001, `type_of_organization` also includes `aero`, `biz`, `coop`, `info`, `museum`, `name` and `pro`. The `networking` section (part II of the course notes) will explain in more details what they are.

At the National Research Council, a *node* is defined as

machine.*institute*.`nrc.ca`

The Research Computing Support Group's SGI O^2 is `nickel.sao.nrc.ca` for the outside world, `nickel.sao` for people within NRC and `nickel` for people within RPSO/SCSG.

8.3 Mail Forwarders

Quite often a mail address will be of the form

`mail` *user@company.type_of_organization*

For example,

```
mail claude.cantin@nrc.ca
```

This obviously does not follow the above discussion on `Machine Naming Convention`.

In such cases, mail is sent to a central *mail forwarder* (in the above example, `nrc.ca`), which then forwards the mail to `user` within the organization.

At NRC, `user` tends to be the user's first name, followed by a period, followed by his/her last name (ex: `Claude.Cantin`). This convention makes it easy for people to send mail to an NRC employee.

Another advantage of using that method is that when an employee is transfered to another department, no electronic mail address needs to be changed: only the entry in the `mail forwarder` needs to be modified. This alleviates the problem of contacting all colleagues to let them know of the change of address.

This method is also used by all organizations with an installed firewall (a firewall is the result of a procedure used to isolate a local network from the Internet; this is usually done for security reasons).

8.4 Mail Folders

A mail folder is a collection of mail messages stored in a single file. By convention, all *folders* are stored within a directory called `$HOME/mail` or `$HOME/Mail` (directory `mail` in the user's home directory).

A `folder` is usually referred to as

+*folder_name* or =*folder_name*

where the plus (+) sign means that the file is residing (or is to be residing) in the folder directory (`$HOME/mail`). Note that that convention does not hold for all mail clients; some require the mail folder directory be included with the folder name.

The default folder is the user's system mail folder, `/var/mail/`*`user`*.

8.5 Signature Files

```
A typical mail program will allow one or more signature files.
   A signature file is a file which contains any information
one would like to append at the end of a mail message.  It
could look like:

-------------------------------------------------------------------------
Claude Cantin                          Claude Cantin
Rm 2025, 100 Sussex Dr.                Piece 2025, 100 Prom. Sussex
Research Computing Support             Soutien informatique en recherche
Information Management Services        Services de gestion de l'information
National Research Council              Conseil National de Recherches
Ottawa, Canada (K1A 0R6)               Ottawa, Canada (K1A 0R6)

claude.cantin@nrc.ca                   1-613-993-0822  (FAX: 993-3127)
-------------------------------------------------------------------------

   A typical signature filename is .signature.
   Often, two such files are used:  one for local mail (mail
to addresses on the local system) and one for mail to remote
users.  The file names could be .lsignature and .rsignature
where local signature files are used for addresses without the
at ("@") sign.
```

Signature files are automatically appended at the end of every message sent.

8.6 Mail Aliases

Mail aliases allow the use of shorter, more meaningful addresses instead of their long form. For example,

```
mail claude
```

could be used instead of

```
mail claude.cantin@nrc.ca
```

Mail client packages handle mail aliases using different methods, and different configuration files.

Group aliases are collections of mail aliases. Using group aliases, one could send mail to many destinations at once.

8.7 MIME: Multi-purpose Internet Mail Extensions

The MIME protocol is a standard used to include (or attach) additional information to a mail message. That additional information may be an ASCII file, a binary file, a video clip, an audio file, an image, or another object.

A MIME compliant mail program will follow the directives embedded in the mail message to prompt the user to save the attachment as a file, play the video/audio clip, or perform a certain function according to the information stored in the attachment.

Although MIME has been around for many years, Microsoft made the term attachments widely used.

Not all mail reader programs are MIME compliant.

8.8 .forward: Forwarding Mail

Often, one has more than one account but want mail to be forwarded to one specific machine. This is done by created file

```
.forward
```

in the $HOME directory.

The content of .forward is the full address to which mail is to be forwarded. For example, if your account contains a .forward file with

```
claude.cantin@nrc.ca
```

then all mail sent to your account will be forwarded to claude.cantin@nrc.ca.

If you want to keep a copy locally, and forward a second copy, simply add a second line in .forward to include your own login. For example, if your local login is cantin:

```
claude.cantin@nrc.ca
cantin
```

8.9 .vacation: I'm away from my desk

The mail system may be set up to automatically respond to incoming mail with a specific message. This is usually done when gone for an extended period of time of longer than a few days.

This is done using the vacation program:

1. Create file .vacation.msg, in your main directory. That file would look similar to

```
Re: away until May 4th.

I am out of the office from April 26 until May 4th.
```

Your mail has been received, and will be read upon my return.

Regards,

Claude

2. Create file .forward, in your main directory:

   \your_login_name, "|/usr/sbin/vacation login_name"

   where your_login_name is your login name on your system.

3. As soon as you have created .forward, initialise the vacation procedure:

   vacation -i -r *interval*

   where *interval* is the reply interval, in days. This is the interval in which a user will receive your reply (content off your .vacation.msg file), no matter how many message he sent you.

   Default is 7 days (2 or 3 is fine). 0 is for every message -- NOT RECOMMENDED as it would interfere with mailing lists, discussion groups, etc.

To disable the facility: remove .forward.

8.10 Securely Accessing POP/IMAP Mailboxes

One of the main advantage (some would say disadvantage!) of using UNIX/linux is the pletora of choices one has in regards to applications. Mail clients are no exceptions.

The POP (Post Office Protocol) and IMAP (Internet Mail Access Protocol) protocols allow a client application to

directly access mail located on a remote server. But, as is often the case with communication software (but this is changing), most POP/IMAP application communicate with the server using cleartext by default.

To get encrypted communication, action is required on the part of the user.

This section will describe two methods one may use their POP/IMAP capable mail client to securely access mail on their POP/IMAP UNIX/linux server.

Both methods apply to systems located within NRC, as well as systems people are using to connect to NRC from home.

First, it should be noted that POP and IMAP are two different protocols. POP uses port 110 whereas IMAP uses 143. A *port* is an access door one uses to communicate to different applications on UNIX/linux. POP uses port 110. IMAP uses port 143. sendmail, the application that accepts mail on the servers, uses port 25. SSH uses port 22. The web uses port 80.

All systems use the same port for the same applications, and all application servers listen to the same ports. This allows for different applications using the same protocol (ex: pine, evolution, netscape, Eudora all use the POP protocol) to know how to contact remote servers because they know which port, and which protocol, to communicate with.

8.10.1 SSH tunneling of email from UNIX to UNIX

One of the capabilities of SSH is to allow the creation of private *tunnels*, creating a VPN (Virtual Private Network). Two uses of SSH-tunneling are to allow X through the SSH connection, and to allow mail to be delivered through that encrypted passage.

A tunnel is create with a command similar to

SSH -L:*l_port*:*mail_server.inst.nrc.ca*:*r_port*
mail_server.inst.nrc.ca

where

- *l_port* is a number between 1024 and 65536.
- *r_port* is the port number on the remote system you would like to tunnel through SSH. For mail uses, this is either 110 (POP) or 143 (IMAP).
- *mail_server.inst.nrc.ca* is the name of the system where your mail resides.

Issueing that command will connect you to *mail_server.inst.nrc.ca* using ssh, while allowing any requests made to *l_port* to be automatically forwarded, through the SSH-encrypted tunnel, to *r_port* on *mail_server.inst.nrc.ca*.

As long as the SSH connection is maintained, your own VPN is active.

For example, let's open a tunnel from the local system to nickel.sao.nrc.ca so you can use IMAP to access your mail on that mail server. Local port 45243 will be used:

```
ssh -L:45243:nickel.sao.nrc.ca:143 nickel.sao.nrc.ca
```

As is with any SSH connection to a remote system, any communication will be encrypted. And any request made to port 45243 on your local system (often seen as localhost:45243) will be translated as port 143 on the remote system (often seen as nickel.sao.nrc.ca:143).

The next step is to configure your mail client (Netscape, kmail, pine, evolution to name a few) to contact the proper mail server. For the name of the mail server, it should be localhost:*l_port*. This will tell your application to use *l_port* on your local system, tunelling the request to port *r_port* on *mail_server.inst.nrc.ca*.

Extending the example above, the local application should define the inbound mail server as localhost:45243. The outbound mail server is still nickel.sao.nrc.ca.

From that point on, any mail request made locally will be tunneled through SSH, to port 143 on nickel.sao.nrc.ca.

Note that for this to work, your connection must be kept active. As soon as the original

```
ssh -L...
```

command is terminated, the tunneling is also terminated, so is your VPN.

8.10.2 SSH tunneling of email from a Windows application to UNIX

Windows mail applications, like MS Outlook, can also connect to UNIX/linux based POP/IMAP servers. The principle is the same as explained when doing so with a UNIX/linux application, except that there is no need to define a specific local port.

For Windows applications, the local and remote port numbers stay the same.

Follow the followin steps to read your UNIX/linux mail from a Windows application:

- Connect to the mail server using an SSH application like putty.
- Configure your mail application (ex: Outlook) to use localhost:143 (for IMAP) or localhost:110 (for POP) as your incoming mail server.
- Start reading mail

Remember that, as was the case above, one must keep the SSH-connection open in order to keep your tunneling going.

8.10.3 `fetchmail`: bring remote mail to local system

Note: fetchmail is fine, but, if possible, we recommend people use a POP/IMAP capable mail client, and connect directly to the mail server, as shown above.

In some instances, people want to use a non-POP/non-IMAP mail application, but still want to read their mail locally, even if their workstation is not a mail server. fetchmail is used to connect to a remote mail server, and bring your email to your local mailbox. You then may use your favourite mail client to browse through your mail.

As with using the POP/IMAP protocols to read your email off a remote server, fetchmail will need to be configure so it is tunneled through SSH.

This is done by first enabling the SSH-tunnel, then using fetchmail to use that tunnel to access mail on the remote server.

Two ways will be shown here. Both will use the .fetchmailrc configuration file to perform the SSH-tunneling of the protocol desired. The first method is simpler while the second one more graceful and more permanent.

In both cases, the configuration file, .fetchmailrc will look similar to:

```
defaults
  fetchall
  keep
  mda "/usr/bin/procmail -d %T"
  poll mail_server.inst.nrc.ca via localhost port
l_port with proto protocol:
  preconnect "ssh -f -L l_port:mail_server.inst.nrc.ca:r_port
mail_server.inst.nrc.ca sleep 20 /dev/null";
```

where

- fetchall means all mail will be transfered. Use that line only the first time you transfer your mail over.
- keep means that the mail will by copied, not moved.
- The poll and preconnect lines log you onto the server and copy the mail over to the local system.
- *protocol* is either pop3 or imap.
- *l_port* is a number between 1024 and 65536.
- *r_port* is the port number on the remote system you would like to tunnel through SSH. For mail uses, this is either 110 (POP) or 143 (IMAP).

- *mail_server.inst.nrc.ca* is the name of the system where your mail resides.

Permissions on .fetchmailrc should be 600 (rw-------).

Manual fetchmail

For a one-time-transfer, fetchmail may be used manually, as in

```
fetchmail
```

The configuration file will be read, and the entire system mailbox from *mail_server.inst.nrc.ca* will be copied locally.

As an example, lets assume the mail server is nickel.sao.nrc.ca. We want to use POP as the protocol, and we will use local port 22334 (any number between 1024 and 65535 is fine).

.fetchmailrc would look like:

```
defaults
   fetchall
   keep
   mda "/usr/bin/procmail  -d %T"
   poll nickel.sao.nrc.ca via localhost port 22334 with proto pop3:
   preconnect "ssh -f -L 22334:nickel.sao.nrc.ca:110 nickel.sao.nrc.ca sleep 20
```

Automated/daemon fetchmail mode

To use fetchmail in daemon mode, use the same configuration file as above, but remove the fetchall line. Then start fetchmail as in

```
fetchmail -d time
```

where *time* is the frequency, in seconds, you want your mailbox checked on the remote system. Ideally, time should be in the order of 300 (5 minutes).

When the command is issued, it will prompt you for your remote password.

Each time the remote server is rebooted, you will have to re-issue the fetchmail -d *time* command, as the tunnel will have been discontinued, and you need to re-create it.

8.11 The Berkely (BSD) Mail Interface

Most systems include this mail-client. Some vendors invoke the program with mail whereas other use Mail.

If the command

mail

returns

No Mail for *user*

the BSD mailer is being used.

8.11.1 Sending Mail

To send mail to a user,

mail *user*[@*node*]

where *user* is the logon name of the recipient, and *node* is his/her machine.

After invoking mail, UNIX will come back and prompt the user for the subject of the message, as in:

prompt> **mail cantin**
Subject:

This is optional.

Once the subject is entered (and the carriage return pressed), mail is in input mode: anything entered is part of the message. End the message by entering a . (dot) on a line by itself, or press (<CTL-d>).

At that time, a

Cc:

prompt will be displayed, allowing the sender to send copies to other users. Cc stands for Complimentary (or Courtesy) copy. Again, this is optional.

Enter a carriage return, and the piece of mail is sent out.

If the entire message has been previously created with an editor, and placed in *file*, it can be sent to *user* with

mail *user* < *file*

~r *filename*: Read Filename

To incorporate a previously edited file into the message, issue

~r *filename*

where *filename* is the name of an ASCII file to be incorporated in the body of the message. The tilde MUST be the first character of the line.

If *filename* is not an ASCII file, use uuencode to encode it into an ASCII sequence.

Note that this must be done while writing the message. Here is a short mail session:

prompt> **mail cantin**
Subject:
the file course.tex will appear following this line
~r course.tex
"course.tex" 80/1888
notice that the name of the file, its number of lines, and the size of it (in bytes) were displayed by the utility, not myself.

The dot on the next line indicates the end of the message.
.
Cc: stan

~v: Invoking the Visual Editor

~v

When writing the body of the message, this command will invoke vi. Once in vi, edit the letter. Exiting vi (write, and quit) will come back to the body of the message (at that time the file being edited by vi will have been written---although the user can't see it!).

Enter a dot on a line by itself to end the composition of the message.

Electronic mail can now be sent from any UNIX system on the NRC Ethernet network to any other machine (UNIX or not)

connected to the Ethernet. To get the *node* name of the receiving system, contact its system administrator.

8.11.2 Reading Mail

Invoking

mail [**-f** *folder_name*]

allows mail to be read.

Upon entering mail, one of two things may happen (depending on how the system was set up):

1. For each piece of mail waiting to be read, one line contains information concerning who the mail comes from, when it was received, and (part of) the subject line of that message. mail prompts the user to enter a command.

2. The last message received is displayed. mail prompts the user to decide what to do with this piece of mail.

Some of the most common commands are explained in the following sections.

n: Next

n or simply <CR>

displays the next message.

p: Print

p

displays (prints) the current message on the standard output.

NOTE: p stands for print, but really means display on the screen. The reason is historical: first UNIX users were using teletype terminals, thus *any* output was printed on hardcopy.

num*: Print message *num

num

displays (prints) message *num* on the standard output.

s: Save

s *filename*

If *filename* does not exist, it will be created. The current message will be appended to that file.

If +*filename* was used, *filename* would be in the folder directory.

Sending a file to a printer

A variation of the save command can be used to pipe the message to a command. For example

s | *command*

will pipe the current message into *command*. If command happens to be lpr or lp, the file will be sent to the printer.

h: Header

h

displays the headers of the active messages.

d: Delete

d

deletes the current message. d also takes a range of message number(s).

r: Reply

r

allows the user to reply to the sender with another message.

s: Forward

s | mail *user*

will forward the mail to *user*.

~m: Incorporating current message

~m

will incorporate the current message (just read) to the message being sent out. The incorporated message will be indented one tab within the message currently being written.

This option is very useful when forwarding a message to another user.

q: Quit

q

will exit mail, and save all undeleted mail messages back into the mailbox, usually /usr/mail/*user*.

8.11.3 Configuring Mail Behaviour

When the mail system is invoked, either for reading or sending mail, a system-wide mail configuration file (/usr/lib/Mail.rc) is read. Then, if it exists, file $HOME/.mailrc is also read.

That file may contain, among mail configuration commands, *aliases* for mail to use. An example of such an alias could be:

```
claude cantin@nickel.sao.nrc.ca +cantin
wayne wayne.podaima@nrc.ca +wayne
group claude stan
```

If mail is sent to group, it will then automatically be sent to both claude and wayne, which actually means cantin@nickel.sao.nrc.ca, the mail folder cantin, wayne.podaima@nrc.ca and the mail folder wayne.

For more information on the .mailrc file, see the Special Files chapter of Book II: Advanced Introduction.

8.11.4 Mail Folders

Creating a Folder

While reading mail, a message can be saved in a folder with

s +*folder_name*

If the folder already exists, the mail message will be appended to the end of it. If the folder does NOT exist, then the file will be created.

Reading from a Folder

To read from a folder,

mail -f +*folder_name*

is used. If only ‘‘mail" is issued, the default folder is /var/mail/$USER.

.record, a special folder

All outgoing mail messages are, by convention, saved in a special folder called .record. This file always grows in size, so it is recommended to periodically clean it up, or rename it.

8.11.5 Mail Aliases

BSD mail aliases are put in file

.mailrc

and are of the form

alias *nickname* real_address
or
alias *nickname* addr_1 addr_2...

8.11.6 MIME: Multi-purpose Internet Mail Extensions

BSD mail is not MIME compliant.

8.12 ELM: ELectronic Mail

Public domain package.

In the late 1980s, early 1990s, ELM was viewed as one of the top emerging ASCII-based mail clients. It was availabe (source compiled) on all major UNIX platforms.

For that reason, and because of its ease of use, it was picked as the default mailer on the UNIX systems manages by the Research Computing Support Group. On those systems, mail was aliased to invoke elm.

By the early 2000s, ELM has not kept up as much as some others (namely PINE, presented in these course notes). Nonetheless, it still is our default ASCII-based linux/UNIX mail client.

If the command

mail

returns a full screen interface similar to 8.1, then elm is being used.

8.12.1 Sending Mail

Standalone Mode

elm *user* [@*node*]

will invoke elm in sending mode. The message:

To: *user*@node
Subject:

will then be displayed. After entering the subject, the string

Copies to:

will prompt you for additional recipient names. A carriage return will leave that field empty.

The editor (pico by default) will be invoked, and the composition of the message may begin. The editor invoked may

```
Mailbox is '/usr/mail/cantin' with 2 messages [ELM 2.4ME+ PL11 (25)]


    1   Apr 30 To cantin@neon.sao (54)
    2   Mar 4  To Claude Cantin   (15)   itest









      |=pipe, !=shell, ?=help, <n>=set current to n, /=search pattern
a)lias, C)opy, c)hange folder, d)elete, e)dit, f)orward, g)roup reply, m)ail,
  n)ext, o)ptions, p)rint, q)uit, r)eply, s)ave, t)ag, u)ndelete, or e(x)it

Command:
```

Figure 8.1: `elm` interface.

be changed by using the option menu, discussed in the previous section.

If you use pico, elm **will prompt you to *save* the file (it will give you a suggested name). It does need to save the file before sending it.**

When the message is ready to be sent, issue the editor command to save the file. elm will come back with the message

```
And now: s
  e)dit message, h)eaders, c)opy, i)spell, !)shell, s)end, or f)orget
                                        p)gp
```

message. Answering to that message will leave elm completely and bring you to the UNIX prompt.

Within the Interface

elm

will invoke the interface.

m

will start the program used to send mail, much the same way as in the previous section.

Batch Mode

Mail could finally be sent in batch mode (or as a one-line command) with

elm *user@node* -s*subject* < *filename*

where *filename* is the name of the file which contains the message to be sent.

8.12.2 Reading Mail

elm

will invoke the interface to read your mail. The interface will look much like Figure 8.2.

Enter the message number to read a specific message, or press ENTER to read the current one (it is highlighted).

```
  Mailbox is '/usr/mail/cantin' with 113 messages [ELM 2.4 PL24 ME7]

   1   Nov 7  SGI Support Mail   (84)   CREATE CASE/LOG SUCCESS
   2   Nov 7  Geof Aers          (22)   Rick's machine-cm8
   3   Nov 7  Geof Aers          (22)   Rick's new enclosure
   4   Nov 7  flash@FlashBack.co (826)  sunflash  82.00
D  5   Nov 7  flash@FlashBack.co (239)  flashback  1386
r  6   Nov 6  Greg Kresko        (20)   VCR to borrow?
   7   Nov 6  Mauro Tomietto     (32)   Itsssss back !!!
   8   Nov 6  Stan Zurawski      (24)   "phone" fixed
   9   Nov 6  SGI Support Mail   (72)   CREATE CASE/LOG SUCCESS
r  10  Nov 6  Clarify user       (52)   Email out for Case 0531153
   11  Nov 6  Clarify user       (49)   Email out for Case 0530997
   12  Nov 6  SGI Support Mail   (73)   CREATE CASE/LOG SUCCESS
   13  Nov 6  Stan Zurawski      (52)   Re: whois...
   14  Nov 6  Clarify user       (47)   Email out for Case 0530997
   15  Nov 6  Jerome, Ron        (29)   UUG list server
   16  Nov 6  Geof Aers          (20)   Re: IMS home page...
   17  Nov 3  Wayne Podaima      (22)   To bug you!
   18  Nov 3  Geof Aers          (23)   ims pages on gold
r  19  Nov 3  Wayne Podaima      (60)   Re: IRIX 6.1 upgrade...
   20  Nov 3  Wayne Podaima      (26)   (Very) Late Monday AM

  |=pipe, !=shell, ?=help, <n>=set current to n, /=search pattern
a)lias, C)py, c)hange folder, d)el, e)dit, f)orward, g)rp reply, m)ail,
n)ext, o)pts, p)rint, q)uit, r)eply, s)ave, t)ag, u)ndel, or e(x)it

Command:
```

Figure 8.2: `elm` Interface: Expert User Level Menu.

n: Next

To read the next message, enter

n

or return to the message list (press **i**) and move the highlighted bar to the message to be read.

To view message *num*, enter the message index number.

s: Save

s

will save the message in a specified mail folder.

p: Printing to a printer

p

will send the current message to the printer, according to the command defined by the ‘‘P)rint mail using" string within the option menu, explained in the Configuring elm section later.

d: Delete current message

d

will perform that operation.

Deleted messages will be tagged with a ‘‘D" in the main menu.

r: Reply to sender

r

will allow the user to reply to the sender, with the option of including/modifying the original message.

Once a message has been replied to, an ‘‘r" will be placed on the left hand side of the message header in the main menu. Figure 8.2 shows a few messages for which a reply was performed.

f: Forward

f

will give the option of editing the current message, then prompt for the user to forward the message to, and will give the option of changing the mail message.

elm will then display:

```
And now: s
      e)dit message, h)eaders, c)opy, i)spell, !)shell, s)end, or f)orget
                                      p)gp
```

and will await further instructions.

q: Quit

q

will end the current elm session.

8.12.3 Configuring Mail Behaviour

The elm configuration file is stored in the .elm hidden directory in your login directory. The configuration file itself is elmrc. This file can be modified with an editor to suit the user's individual tastes, or may be modified with the help of an interactive menu utility invoked within elm itself by issuing the option command

o

The interface should look similar to Figure 8.3.

NOTE that this configuration menu only changes a small number of parameters. Many other elm features may be modified by using an editor on elmrc.

The main options users may want to change are the editor they would prefer using during the composition of the message (press ‘‘e" within the options menu), and the command used to send the current message to a printer (press ‘‘p" within the options menu).

```
                    -- ELM Options Editor --

C)alendar file       : /usr/people/cantin/calendar
D)isplay mail using  : builtin+
E)ditor              : vi
F)older directory    : /usr/people/cantin/Mail
S)orting criteria    : Reverse Date Mail Sent
O)utbound mail saved : =sent
P)rint mail using    : /usr/local/bin/apr %s
Y)our full name      : Claude Cantin


A)rrow cursor        : OFF
M)enu display        : ON

U)ser level          : Expert User
N)ames only          : ON


Select first letter of option line, '>' to save, or 'i' to return to index.

Command:
```

Figure 8.3: `elm` Options Menu.

Another option to consider is the ‘‘level" of the elm menu seen at the bottom of the screen. The higher the level, the more menu items are displayed! Following that logic, most users should select expert as their User level. However, even though a menu item is not displayed, it is still accessible by typing the character representing it. Help (?) may be used to list the available options.]

Once the choices are saved (using >), one can return to the main menu using ‘‘i". Note that most commands within elm do NOT need to be followed by a carriage return (<CR>).

Modifying the file .elm/elmrc allows you to modify:

- The name of the file all outgoing messages are to be stored in. That feature may be turned ON, or OFF.

  The lines in .elm/elmrc allowing to keep outgoing messages in a file called Mail/outbound would look like:

```
# save a copy of all outbound messages?
copy = ON
# where to save copies of outgoing mail to, default file is "=sent"
sentmail = $HOME/Mail/outbound
```

- The name of your ‘‘signature" file. That file is automatically appended to any mail sent out.

  For example:

```
# remote ".signature" file to append to appropriate messages...
remotesignature = .rsignature
# local ".signature" file to append to appropriate messages...
localsignature = .lsignature
```

- The behaviour of the mailer once a message is read. Is the mail deleted? Is it automatically moved to a read folder?

- Should the interface prompt for the ‘‘cc"?

  The lines in .elm/elmrc could be:

  ```
  # would you like to be asked for Carbon-Copies information eac
  askcc = ON
  ```

- Many other features.

8.12.4 Mail Folders

elm *folders* are stored within the directory defined by the F)older directory field in the option menu (or the maildir variable in mailrc). The default is $HOME/Mail.

The folder may be referred to as

+*folder_name*
or
=*folder_name*

where the plus or equal (+/=) sign means that the file is residing (or is to be residing) in the folder directory. If the plus (+) or equal (=) sign is not used, the folder will reside in the directory from which elm was invoked.

Creating a Folder

While reading mail, a message can be saved in a folder with

s +*folder_name*

If the folder already exists, the mail message will be appended to the end of it. If the folder does NOT exist, then the file will be created.

Reading from a Folder

To read from a folder,

elm -f +*folder_name*

is used. If only ‘‘elm" is issued the default folder /var/mail/$USER is read.

Changing Folders

Within the main elm menu, pressing

```
c
```

will replace the current mailbox with another, represented by a folder. It is the equivalent of leaving elm, then re-invoking it with a folder name as an argument.

After entering ‘‘c", a number of strings may be entered:

! : Will read the incoming (system) mailbox.

< : Will read the folder into which outgoing messages are kept (if so defined in .elm/elmrc, the elm configuration file).

folder : Will read the contents of folder *folder*.

If *folder* includes wildcards which expand into more than one folder name, all folder names will be displayed. It would then be possible to choose any one of these.

For example, an asterisk (*) will show all folders.

8.12.5 Mail Aliases

aliases are created, modified and deleted with the use of the alias menu, shown in Figure 8.4.

Five (5) aliases are defined, four (4) of which are of type Person and one of which is a type Group alias. The actual alias file (.elm/aliases.text) is seen in Figure 8.5. It could be edited manually, within elm (by pressing ‘‘e"), or by using commands within the aliases menu.

Figure 8.4 shows a number of options:

c : Change/modify the current alias.

d : Tag the current alias for deletion.

e : Edit the .elm/aliases.text file, using vi. This is often the quickest way to add/modify/delete aliases.

```
                    Alias mode: 5 aliases [ELM 2.4 PL24 ME7]

     1   Claude Cantin                                Person    cantinc
     2   Ratilal Haria                                Person    ratilal
     3   Some Group Name                               Group    group
     4   Wayne Podaima                                Person    wayne
     5   Stan Zurawski                                Person    stan


    Alias commands:  ?=help, <n>=set current to n, /=search pattern
a)lias current, c)hange, d)elete, e)dit aliases.text, f)ully expand,
l)imit display, m)ail, n)ew alias, r)eturn, t)ag, u)ndelete, or e(x)it

Alias:
```

Figure 8.4: `elm` Alias Interface.

```
     Note:  group aliases must not include addresses with an @
     symbol; it should only include either aliases previously
     defined or addresses without the @ symbol.

   f :  Show the real address of the alias.

   l :  Select only specific aliases to be displayed.

   m :  Mail to that alias.

   n :  Create a new alias.

   r :  Return to the main elm menu.

   t :  Tag the current alias.

   u :  If the current alias is marked for deletion, ``untag"
     the alias.

   x :  Exit this menu, without saving any changes made.


wayne = Wayne Podaima = podaima@neon.sao.nrc.ca
ratilal = Ratilal Haria = ratilal.haria@nrc.ca
stan = Zurawski; Stan = zurawski@sys35.di.nrc.ca
cantinc = Claude Cantin = claude.cantin@nrc.ca
group = Some Group Name = wayne, ratilal, stan, cantinc
```

Figure 8.5: Content of `.elm/aliases.text`

```
   The alias file may also be edited manually by modifying file

     .elm/aliases.text

   Once the changes are made, the command

     newalias

must be run.  That command creates the aliases.hash file elm
uses for aliases.
```

8.12.6 MIME: Multi-purpose Internet Mail Extensions

ELM is MIME compliant, but was made so after it was initially written. The result is that using attachments with ELM may not be as obvious as reading/sending mail with the tool.

MIME and elm: receiving messages.

Most current versions of elm installed in the Institute servers can handle MIME messages. Uppon starting elm, all MIME messages will be tagged with a M on the left of the message header (as shown in Figure 8.2).

Reading the message will automatically invoke the proper utility to view the mail.

Or, at the main elm menu, pressing "v" (note that this letter is *not* found in the menu) will bring a new menu containing the list of included files (ie: attachments) in the message.

Figure 8.6 is a mail message containing two files: /etc/hosts and images/tiger.ps. Each of those individual files may be save, and/or viewed using the metamail program.

```
                    Attachments Menu (3 attachments)

   1 (none)                          (437)    text/plain
   2 /etc/hosts                      (877)    text/plain
   3 images/tiger.ps                 (78519)  application/postscript



Attachments:
                s)ave, v)iew using metamail, return to i)ndex
```

Figure 8.6: elm Attachments Menu (incoming messages)

is a mail message with a file containing two files: /etc/hosts and images/tiger.ps. Each of those individual files may be save, and/or viewed using the metamail program.

Pressing ‘‘i" would return to the main menu.

MIME and elm: sending messages.

To send a MIME-encoded mail message, the procedure is very similar to sending a normal message. The main difference is performed after saving the message from the editor but before actually sending the message:

```
And now: s
   e)dit message, h)eaders, c)opy, i)spell, !)shell, a)tt, s)end, or f)orget
```

Figure 8.7 shows what happens when ‘‘a" (for attachments) is pressed. Files could be added/deleted from the message.

```
                        Attachments Menu (0 attachments)



Attachments:
                             a)dd, d)elete, m)odify, q)uit
```

Figure 8.7: elm Attachments Menu (outgoing messages)

8.13 The Pine mail interface

pine, developped at the University of Washington, is a menu driven mail interface. Full screen based, its design allows the interface to be easy to use.

pine

would show an interface similar to 8.8.

8.13.1 Sending Mail

Standalone Mode

pine *user* [@*node*]

```
  PINE 3.91   MAIN MENU                    Folder: INBOX  2 Messages

    ?     HELP              -  Get help using Pine

    C     COMPOSE MESSAGE   -  Compose and send/post a message

    I     FOLDER INDEX      -  View messages in current folder

    L     FOLDER LIST       -  Select a folder OR news group to view

    A     ADDRESS BOOK      -  Update address book

    S     SETUP             -  Configure or update Pine

    Q     QUIT              -  Exit the Pine program



Copyright 1989-94. PINE is a trademark of the University of Washington.
               [Folder "INBOX" opened with 2 messages]
? Help                       P PrevCmd               R RelNotes
O OTHER CMDS L [ListFldrs] N NextCmd                 K KBLock
```

Figure 8.8: `pine` interface.

```
 PINE 3.91 COMPOSE MESSAGE </usr/people/cantin/Mail/[]> (CLOSED) 0 Msgs
To      : cantin@nickel.sao.nrc.ca
Cc      :
Attchmnt:
Subject :
----- Message Text -----




^G Help   ^X Send     ^R Rich Hdr ^Y PrvPg/Top ^K Cut Line  ^O Postpone
^C Cancel ^D Del Char ^J Attach   ^V NxtPg/End ^U UnDel Line^T To AddrBk
```

Figure 8.9: `pine`: Sending Mail.

will invoke the full screen interface, as shown in Figure 8.9, using cantin@nickel.sao.nrc.ca as a example email address.

The up/down arrow keys are used to cycle through the headings.

The last two lines at the bottom of the screen represent the commands used to perform specific tasks. For examples

```
^X
```

is used to Send the mail message (user will be prompted for confirmation).

Within the Interface

```
pine
```

will invoke the interface.

```
c
```

will start the program used to send mail, much the same way as in standalone mode.

Batch Mode

pine does not work in batch mode. It must be used interactively.

8.13.2 Reading Mail

```
pine
```

will bring the interface, as shown in Figure 8.8.
Pressing

```
i
```

will display the mail messages in your current folder.
the new interface will now look similar to Figure 8.10
Press the Enter key to read the highlighed message.

The last two lines on the interface also suggests commands you may use.

```
  PINE 3.91   FOLDER INDEX     Folder: /var/mail/cantin Message 145 of 145
    121 Jun 27 Clarify user        (1,960) Email out for Case 0618266
+   122 Jun 27 Wayne Podaima         (545) Official maple Update
+   123 Jun 28 Andrew Booth        (1,281) Re: Server rights for Dave Rogers.
+   124 Jun 28 Wayne Podaima         (827) Return Old M-60 Door Access Cards
+   125 Jun 30 Wayne Podaima       (1,271) Re: softwindows on neon...
    126 Jul  2 Ratilal Haria         (697) New DB
+   127 Jul  3 Michel Proulx       (1,370) isdn
+   128 Jul  4 Wayne Podaima         (597) 64-bit IMSL for SGI
+   129 Jul  4 Michel Proulx         (424) station.ctn
+   130 Jul  5 Wayne Podaima         (744) objectserver Resets
+   131 Jul  8 Jamie Bennett         (644) xwinnmr
+   132 Jul  8 Andrew Booth          (527) cgi-bin scripts.
+   133 Jul  8 Wayne Podaima         (940) xdvi Size Problem
+   134 Jul  9 To: cantin@nickel.  (2,610) (fwd) Re: Anyone know of a Camera New
+   135 Jul 10 Daryoush Sheikh-Ba  (2,147) If it's possible
+   136 Jul 10 Steve Myers         (1,061) Server Extensions...
+   137 Jul 10 Steve Myers        (37,168) Readme Files
    138 Jul 10 Attila Berces         (534) disk on tc3
    139 Jul 10 Laura Vais          (3,041) IRIS On-Line Update
    140 Jul 11 Robertson, Gilles     (885) Unix
    141 Jul  8 Zborowski, Mary D   (1,486) FW: link or pointer needed
+   142 Jul 11 Wayne Podaima         (512) Blank root menu problem
    143 Jul 11 To: MINOLTA-L@list  (2,449) FS: Minolta SRT-200...
+   144 Jul 11 Andrew Booth        (1,836) Re: Server rights for Dave Rogers.
    145 Jul 11 akachhy@hogpa.ho.a  (1,185) Re: FS: Minolta SRT-200...



? Help       M Main Menu  P PrevMsg     - PrevPage    D Delete      R Reply
O OTHER CMDS V [ViewMsg]  N NextMsg   Spc NextPage    U Undelete    F Forward
```

Figure 8.10: `pine` interface.

n: Next

To read the next message, enter

n

or return to the message list (press **i**) and move the highlighted bar to the message to be read.

To view message *num*, enter

j

followed by the message index number.

s: Save

s

will save the message in a specified mail folder.

p: Printing to a printer

y

will invoke the printer command defined using the SETUP menu off the main pine interface (lp is set by default).

d: Delete current message

d

will perform that operation.

Deleted messages will be tagged with a ‘‘D" in the main menu. They may be undeleted by using

u

r: Reply to sender

r

will allow the user to reply to the sender, with the option of including/modifying the original message.

Once a message has been replied to, a ‘‘A" will be placed on the left hand side of the messages interface.

f: Forward

f

will use the same interface as sending mail, but with the message to be forwarded included in the message body.

q: Quit

q

will end the current pine session.

8.13.3 Configuring Mail Behaviour

file (.pinerc) or by using the

SETUP

menu off the Main Menu.
The SETUP menu offers a number of different settings.
Selecting

S

will give

```
Choose a setup task from the menu below :
^G Help       P [Printer]   C Config
^C Cancel     N Newpassword U Update
```

Selecting

c (Config)

will display something very similar to Figure 8.11. That figure only shows a portion of the configurable options.
Modifying .pinerc also works (a default configuration file may be found in /usr/local/lib/pine.conf

```
  PINE 3.91   SETUP CONFIGURATION         Folder: /tmp/junkmail  145 Messag
personal-name          = <No Value Set: using "Claude Cantin">
user-domain            = <No Value Set>
smtp-server            = <No Value Set>
nntp-server            = <No Value Set: using news.nrc.ca>
inbox-path             = <No Value Set: using "inbox">
folder-collections     = <No Value Set: using /usr2/people/cantin/Mail/[]>
news-collections       = <No Value Set: using *{news.nrc.ca/nntp}[]>
default-fcc            = <No Value Set: using "sent-mail">
postponed-folder       = <No Value Set: using "postponed-msgs">
read-message-folder    = <No Value Set>
signature-file         = <No Value Set: using ".signature">
global-address-book    = <No Value Set>
address-book           = <No Value Set: using .addressbook>
feature-list           =
            Set        Feature Name
            ---   ----------------------
            [ ]  assume-slow-link
            [ ]  auto-move-read-msgs
            [ ]  auto-open-next-unread
            [ ]  compose-rejects-unqualified-addrs
            [ ]  compose-sets-newsgroup-without-confirm
            [ ]  delete-skips-deleted
            [ ]  enable-aggregate-command-set
            [ ]  enable-alternate-editor-cmd
            [ ]  enable-alternate-editor-implicitly
            [ ]  enable-bounce-cmd
            [ ]  enable-flag-cmd
            [ ]  enable-full-header-cmd
            [ ]  enable-incoming-folders
            [ ]  enable-jump-shortcut
            [ ]  enable-mail-check-cue
            [ ]  enable-suspend
            [ ]  enable-tab-completion
            [ ]  enable-unix-pipe-cmd
            [ ]  expanded-view-of-addressbooks
            [ ]  expanded-view-of-folders

? Help       E Exit Config  P Prev      - PrevPage    A Add Value
             C [Change Val] N Next    Spc NextPage    D Delete Val  W Where
```

Figure 8.11: `pine` Configuration Menu.

8.13.4 Mail Folders

pine *folders* are stored, by default in $HOME/Mail. It is configurable in the SETUP menu interface, under

folder-collections

The folder may be referred to as its name (no prefix). To use files not located in the folder directory, the full path name must be used.

Creating a Folder

While reading mail, a message can be saved in a folder with

s *folder_name*

If the folder already exists, the mail message will be appended to the end of it. If the folder does NOT exist, then the file will be created (after being prompted).

Reading from a Folder

To read from a folder,

pine -f *folder_name*

is used. If only ‘‘pine" is issued the default folder /var/mail/$USER is read.

Changing Folders

Within the main pine menu, pressing

l (FOLDER LIST)

(followed by pressing the Enter key in the next menu) will display the various folder names. The folder may be selecting using the arrow keys to navigate.

8.13.5 Mail Aliases

Aliases are handled within the

ADDRESS BOOK

menu, within pine's main menu (enter a in the main menu).
The ADDRESS BOOK menu looks similar to Figure 8.12.

```
  PINE 3.91   ADDRESS BOOK   Folder: /tmp/junkmail  Message 145 of 145
cantinc  Claude Cantin               cantin@nickel.sao.nrc.ca
ratilal  Ratilal Haria               ratilal.haria@nrc.ca
wayne    Wayne Podaima               wayne@neon.sao.nrc.ca
group    UNIX People                 DISTRIBUTION LIST:
                                     cantinc
                                     ratilal
                                     wayne
                                     zurawski@palladium.sao.nrc.ca



                        [Now in addressbook .addressbook]
? Help       M MainMenu P PrevEntry   - PrevPage  D Delete  S CreateList
O OTHER CMDS E [Edit]   N NextEntry Spc NextPage  A Add     Z AddToList
```

Figure 8.12: pine ADDRESS BOOK menu structure

Functions available to the user include

e : Change/modify specific parts of the alias.

p : Move pointer (highlighted line) to previous entry.

n : Move pointer (highlighted line) to next entry.

- : Move pointer to previous page (assuming a large list of aliases).

Spc : Move pointer to next page (assuming a large list of aliases).

d : Delete current alias.

a : Add a new alias.

s : Create a new alias representing a group of people.

z : Add to a group.

w : Find an word within the alias file.

c : Compose a message for that alias.

Aliases may also be altered by modifying file .addressbook.

8.13.6 MIME: Multi-purpose Internet Mail Extensions

pine was written with MIME in mind.

Sending MIME messages

To incorporate an attachment to your message, enter the file name after the

Attchmnt:

subheading.

Receiving MIME messages

When reading a MIME-encoded message, a menu similar to Figure 8.13 will be seen

To see the attachment, press

v

This will start the *companion program* used to view the attachment.

In this case, the program xv will be used to display the content of the GIF file.

```
  PINE 3.91   MESSAGE TEXT   Folder: /tmp/junkmail  Message 150 of 150 ALL
Date: Fri, 12 Jul 1996 10:32:21 -0400 (EDT)
From: Claude Cantin <cantin@nickel.sao.nrc.ca>
To: Claude Cantin <cantin@nickel.sao.nrc.ca>
Subject: gif mime file
Parts/attachments:
   1 Shown   16 lines  Text
   2   OK   155 KB     Image, ""
----------------------------------------

This message includes a GIF image (file cam.gif).

   Claude


------------------------------------------------------------------------------
Claude Cantin                          Claude Cantin
Rm 2025, 100 Sussex Dr.                Piece 2025, 100 Prom. Sussex
Research Computing Support             Soutien informatique en recherche
Information Services Management        Services de gestion de l'information
National Research Council              Conseil National de Recherches
Ottawa, Canada (K1A 0R6)               Ottawa, Canada (K1A 0R6)

claude.cantin@nrc.ca                   1-613-993-0822  (FAX: 993-3127)
http://www.nrc.ca/rpso/scsg
------------------------------------------------------------------------------

  [Part 2, ""  Image/GIF  155KB]
  [Not Shown. Use the "V" command to view or save this part]




? Help       M Main Menu  P PrevMsg     - PrevPage    D Delete      R Reply
O OTHER CMDS V ViewAttch  N NextMsg   Spc NextPage    U Undelete    F Forwa
```

Figure 8.13: `pine` Attachments Menu

8.14 Exercises

1. What is the identifying name of the UNIX system you are using?
2. Send yourself a piece of mail.
3. Send someone else a piece of electronic mail.
4. Send someone else on another system a mail message.
5. Send yourself a piece of mail, and include inside it one of your files.
6. In a file, write a short message. Then, send yourself that file using mail (hint: one command followed by a redirection...).
7. Read the mail you have sent to yourself. Issue a ? and try some of those facilities. Save the piece of mail in a folder. Reply to yourself.
8. You have sent a piece of mail to a friend. He/she has lost it, so he/she asks you to send it again. Where can it be found?
9. Using elm, change the user level of the menu system.
10. Using elm, create an alias, and send a piece of mail to that alias.
11. Using elm, repeat questions 2 through to 6.
12. Using pine, repeat questions 2 through to 6.

Chapter 9

Commands II

This chapter will introduce a few additonal commands useful to the intermediate user.

9.1 Location Commands

9.1.1 whereis: Where Is

whereis *filename...*

By looking only in a list of standard directories, whereis tries to locate *filename*.

For IRIX 6.5 (SGI):

```
/sbin, /etc
/lib, /lib32, /lib64
/usr/{bin, sbin, bsd, ucb, etc, games, demos, lbin}
/usr/{lib, lib32, lib64}
/usr/local/{bin, etc, lib}
/usr/bin/X11
/usr/local/freeware/{bin, lib}
/usr/share/catman/*
/usr/share/man/*
/usr/catman/*
/usr/man/*
/usr/freeware/catman/*
/usr/src/cmd
```

```
prompt> whereis ls
ls:  /usr/bin/ls /sbin/ls /usr/share/catman/u_man/cat1/ls.z
```

9.1.2 which: Which Program

which *file1...*

To find where a program/utility is, whereis looked in a predefined set of directories. which will look in the list of directories defined by the path (or PATH) variable. If the program is found, its full path will be immediately displayed.

The following example shows how the whereis and the which commands could show different results:

```
prompt> whereis rsh
rsh:  /usr/bsd/rsh /usr/lib/rsh
/usr/share/catman/u_man/cat1/rsh.z
prompt> which rsh
/usr/bsd/rsh
```

9.1.3 find: Find

find is a powerful utility that allows the user to find a specified file or files, and execute a program upon a positive result of the search.

This course will show a simplified version of the utility.

find *path* **-name** *filename* **-print**

path is the directory (use absolute path name) where the search will begin. From that directory, all subdirectories will be searched, and so on.

filename is the name of the file to be searched for. If the file name contains metacharacters (wildcards), it should be in quotes.

find /usr/demos -name "ba*" -print

will find all files beginning with ba in directory /usr/demos and below. Note that the -print keyword is needed to display the filenames. Many other commands could have been used instead of print.

```
prompt> find /usr/demos -name "ba*" -print
/usr/demos/backgammon
/usr/demos/banner
/usr/demos/battlestar
/usr/demos/lib/quiz.k/babies
/usr/demos/lib/quiz.k/bard
/usr/demos/lib/battlestar.log
/usr/demos/lib/backrules
```

To find all core files in your directory,

```
prompt> find $HOME -name core -print
```

Instead of -print, you can use any command. Use {} to represent the filename, and the line must end with \; (backslash semicolon).

For example, if you want to produce a long listing of each occurance of file core within your $HOME directory:

```
prompt> find $HOME -name core -exec ls -l {} \;
```

Or even better: if you want to remove all core files:

```
prompt> find $HOME -name core -exec rm {} \;
```

9.1.4 locate: find files

If all you want to do is to find files, and you are using linux, use locate:

```
locate filename
```

It will search a pre-build database made out of files from the various filesystems/directories on your system.

That database file is usually updated daily, in the early morning hours. If a file was removed from the system after the

database file was updated, locate will still list that file. The same is true if a file was added after the database file was updated: the file will not be displayed.

locate is very fast, very efficient. But be aware of the pitfalls mentionned above.

It is typically found only on linux systems.

9.2 Process Commands

For one reason or another, every user manages to block programs: an infinite loop may occur, or a program takes too long to finish. A running program is called a process. Every process can be displayed and, if necessary, stopped (or killed).

This section will describe process-related commands.

9.2.1 ps: Process Status

ps [-eflu]

will display, on the standard output, information about the processes running on the system at that time.

-e : display every process running, as well as the CPU time used by each one.

-f : display more information about every process running, such as the user running it, when it started, etc.

-l : display the long version, including the sizes of the jobs, in 4k pages.

-u : must be followed by a *login/user* name. The result will display all processes run by *user*.

In the following example, the user is logged on to the console, and has no jobs running, except for his/her shell and of course the command itself.

```
prompt> ps

  PID TTY      TIME COMD
14304 ttyq2    0:00 ps
13943 ttyq2    0:02 tcsh
```

The next example shows a user using a windowing interface (in this case, SunView):

```
prompt> ps

 PID TT STAT  TIME COMMAND
1338 co IW    0:00 /bin/sh /home/nrccsb2/cantin/bin/sunview
1339 co IW    0:04 /usr/bin/sunview -background /home/nrccsb2/cantin/img/space
1342 co S     3:59 textedit -Wp 479 98 -Ws 673 764 -WP 840 0 -Wi
1343 co S     0:28 clock -Wp 497 32 -Ws 210 47 -WP 704 0 -Wi -S
1345 co S     1:30 perfmeter -Wp 976 0 -Ws 170 69 -WP 0 0 -v cpu
1340 p0 D     1:34 cmdtool -Wp 0 0 -Ws 673 471 -WP 0 0 -Wl $<<$ CONSOLE $>>$
1341 p0 S     0:12 -bin/csh (csh)
1427 p0 S     0:24 perfmeter nrccsb2
1438 p0 R     0:00 ps
1347 p1 S     1:38 cmdtool -Wp 0 350 -Ws 673 550 -WP 772 0 -Wi
1348 p1 IW    0:06 -bin/csh (csh)
1351 p1 S     0:02 rlogin nrccsb2
1352 p1 S     0:02 rlogin nrccsb2
```

This user has many processes running, each of them taking some CPU time. Some of the programs running are the windowing system (sunview), textedit (a full screen ‘‘point-and-click" editor), a clock, a performance meter, a C shell, two windows (cmdtool), a remote logon to node nrccsb2, and the ps program.

The first column of the output is the process id, the second column is the control terminal (co = console, p0 = ttyp0, p1 = ttyp1), the third column is the state of the job (I = idle process, W = swapped out, S = sleeping, D = in disk waits, R = runnable). The next column displays the CPU time used by the process so far, in minutes:seconds, and finally, the last column displays the command used.

9.2.2 kill: Kill Process

kill [*signal*] *pid*

is used to destroy or terminate a process whose process id is *pid*. The *pid* is found by using the ps command.

kill 1427

would terminate the performance meter.

Sometimes, a process needs to be killed, then immediately restarted.

kill -1 1427

will send a HUP (Hang Up -- kill and restart) signal to process-id 1427.

In some cases, the process will not catch the normal *kill* interrupt. When that happens, the ‘‘sure kill" is applied:

kill -9 1427

will send an interrupt to that process.

9.3 Verifying System Behaviour

9.3.1 df: Disk Space

```
df (SUN, Linux)
df -k (SGI)
bdf (HP)
```

displays the current usage of disk drives used on the system. The output is usually in blocks (512 or 1024 bytes, depending on the vendor) and includes both used and free space for each of the logical/virtual drives installed and configured.

The directory onto which the drive is attached to the filesystem is shown as the last field of the output.

```
prompt> df -k
Filesystem                Type  kbytes     use   avail %use  Mounted on
/dev/root                 efs   471376  445886   25490  95%  /
/dev/dsk/lv1              efs  1975050 1362496  612554  69%  /usr2
neon.sao.nrc.ca:/usr/local nfs 3961000 2507509 1453491  63%  /tmp_mnt/usr/local
```

The above example (executed on a SGI) shows that:

- The root filesystem (/) is 95% used; that there is only 25MB available on that disk.
- /usr2 is a logical volume (/dev/dsk/lv1); in reality it is two 1GB drives *stripped* together to form a larger logical volume.
- /usr/local is not local to this system. It is NFS mounted (attached to the filesystem through the network) from system neon.sao.nrc.ca

9.3.2 du: Directory Usage

du [-ks] *directory...*

where

k : Use units of kb (1024 bytes) for size.

s : Display summary only.

directory : Is the directory where du is to start reporting sizes.

The default behaviour of du is to show the sizes of all directories, beginning with the current directory. Optionally, a starting point may be used (*directory* above).

The following shows the output from du -ks * issued within the /usr/local directory of gold.sao.nrc.ca:

```
prompt> du -ks *
865     PowerMonII
6730    adobe
10548   aswedit
```

```
153801  bin
2348    catman
975     data
9978    doc
13688   emacs
89424   etc
2       flexlm
160617  ftp
1       g92
87886   g92.old
126121  g94
4482    groff
118933  imsl
154     include
31      info
74294   lib
12455   pvm3
21791   src
391     tclX
29417   texmf
1260    tkX
2       tmp
223     uniXEDIT
```

9.3.3 `top`: Show Top Processes

```
top
```

will show the processes (programs) using the most CPU cycles on the system. It uses a full screen format, with the top 4 lines giving information about:

- load average.
- system name, time of day.
- concurrent processes.
- CPU state.

- memory installed, used, free.
- swap available and free.

The remaining portion of the screen gives snapshot of user processes, similar to ps output, giving information, for top processes, concerning:

- owner.
- priority.
- total size, resident (in RAM) size.
- running time.
- %CPU used.

The entire screen information is updated every few seconds.
On a single CPU system, the output is similar to

```
IRIX nickel 6.5 IP32           load averages: 0.31 0.20 0.06                17:45:5
69 processes:  66 sleeping, 2 stopped, 1 running
CPU: 99.5% idle,  0.0% usr,  0.5% ker,  0.0% wait,  0.0% xbrk,  0.0% intr
Memory: 384M max, 326M avail, 283M free, 512M swap, 512M free swap

    PID       PGRP USERNAME PRI  SIZE   RES STATE    TIME WCPU% CPU% COMMAND
2590702    2590702 cantin    20 2216K 1000K run/0    0:00  0.1  0.59 top
2572241        250 root      20 2996K 1688K sleep    0:06  0.0  0.10 sshd1
    172        172 root      20 2492K 1092K sleep    6:10  0.0  0.02 nsd
    333        333 root      32 2060K  900K sleep   13:25  0.0  0.02 xntpd
    241        241 root      20 3332K 1588K sleep    6:20  0.0  0.01 httpd
```

But on a multiple CPU system (SGI Origin 200/4 CPUs) it may look more like:

```
IRIX64 gold 6.5 IP27           load averages: 0.99 0.99 0.85           17:45:03
164 processes: 116 sleeping, 12 zombie, 33 stopped, 3 running
4 CPUs: 46.9% idle, 26.9% usr, 26.1% ker,  0.0% wait,  0.0% xbrk,  0.1% intr
```

```
Memory: 1024M max, 908M avail, 305M free, 2193M swap, 2108M free swap

    PID       PGRP USERNAME PRI  SIZE   RES STATE    TIME WCPU% CPU% COM
2460748        522 jolanta   18  142M   21M run/1   36:11  6.3 99.03 l70
2460374        522 root      20  800K  624K run/2    0:00  1.7 26.88 nic
2461894    2461894 cantin    20 2384K 1536K run/0    0:00  0.1  1.71 top
    178        178 root      20 4944K 1824K sleep   36:20  0.0  0.65 nsd
    180        180 root      20 7504K 5664K sleep   26:29  0.0  0.34 nam
2458709    2439134 scstest   15 1648K  560K sleep    0:00  0.3  0.26 sle
2439679    2439134 scstest   15  592K  448K sleep    0:02  0.0  0.13 469
2439862        276 root      20 5264K 1792K sleep    0:00  0.0  0.08 ssh
    813        813 root      20 2944K  320K sleep   10:35  0.0  0.05 med
1943273    1943273 gabriel   20   21M 2688K sleep    1:31  0.0  0.03 tvd
 764286     764286 root      20 3936K 2720K sleep    2:40  0.0  0.01 htt
2229293    2229293 root      20 6240K 2064K sleep    2:25  0.0  0.01 pbs
2229422    2229422 root      20 6256K 2256K sleep    5:52  0.0  0.01 pbs
```

The output of top may be interactively custimized with a few simple commands. A few include (from IRIX 6.5; linux interactive commands vary -- issue ? to view the list):

o : specify order of precesses shown (size, cpu, time).

k : specify which process to kill (by PID).

n : specify number of processes to display.

u : specify *user* process to show.

q : quit.

? : display help information on commands available.

NOTE: issue h or ? (help) within top on your own system. The interactive options vary not only from UNIX to linux, but they also differ between linux distributions AND versions of the same linux distribution (ex: SuSE 8.0 and SuSE 7.3 use different interactive commands).

9.3.4 sar: System Activity Report

```
This section specific to both IRIX and linux.
   sar takes a periodic snapshot of the system.  When invoked,
it displays the results of those snapshots:


prompt> sar

IRIX64 tp0 6.5 07201611 IP27    02/22/01

00:00:02  %usr  %sys %intr  %wio %idle %sbrk  %wfs %wswp %wphy %wgsw %wfif
01:00:02    70     0     0     7    22     0   100     0     0     0     0
02:00:02    64     0     0     2    33     0    51    49     0     0     0
03:00:02    64     0     0     0    36     0    75    24     0     0     0
04:00:03    63     0     0     2    36     0    81    19     0     0     0
05:00:02    65     0     0     5    30     0   100     0     0     0     0
06:00:02    65     0     0     2    33     0   100     0     0     0     0
07:00:02    65     0     0     1    34     0   100     0     0     0     0

    skip a few lines

11:40:02    71     0     0     0    28     0   100     0     0     0     0
12:00:02    71     0     0     0    29     0   100     0     0     0     0
12:20:02    73     0     0    15    12     0   100     0     0     0     0
12:40:02    74     0     0     1    25     0   100     0     0     0     0
13:00:02    73     0     0     0    26     0   100     0     0     0     0
13:20:02    76     0     0     0    23     0   100     0     0     0     0
13:40:02    74     1     0     3    22     0   100     0     0     0     0
14:00:02    71     0     0     8    20     0   100     0     0     0     0
14:20:03    70     0     0     2    27     0   100     0     0     0     0
14:40:02    67     0     0     0    32     0   100     0     0     0     0
15:00:02    71     0     0     3    26     0   100     0     0     0     0
15:20:02    70     0     0     1    28     0   100     0     0     0     0
15:40:02    74     0     0     0    25     0   100     0     0     0     0
16:00:02    70     0     0     1    29     0   100     0     0     0     0
16:20:03    71     0     0     3    26     0   100     0     0     0     0
16:40:02    71     0     0     1    27     0   100     0     0     0     0
17:00:02    71     1     0     7    21     0   100     0     0     0     0
```

```
17:20:03     65      1      0      1     33      0    100      0      0      0      0
17:40:02     65      0      0      1     33      0    100      0      0      0      0
18:00:02     64      0      0      1     34      0    100      0      0      0      0
19:00:02     61      0      0      4     35      0    100      0      0      0      0
20:00:02     69      0      0      0     31      0    100      0      0      0      0
Average      67      0      0      4     29      0     98      2      0      0      0
```

```
sar incr steps
```

may be used to take *steps* snapshots every *incr* seconds.

Memory usage and disk usage may be looked at using the r and d flags, respectively.

9.3.5 swap/swapon: Virtual memory

swap is an extension of RAM, located on a hard drive.

linux uses swapon instead of swap. The same command is used to both view swap allocation on the system, and add new swap partitions/files (only root).

To determine the amount of swap on a system,

```
/etc/swap -l
```

may be used. This will display the amount of swap space available on the system.

```
prompt> /etc/swap -l
lswap path         dev    pri swaplo   blocks     free  maxswap    vswap
    3 /usr/local/VSWAP
                128,55     7      0        0        0        0   409600
    2 /dev/dsk/dks4d2s7
                128,1063   5      0   200000   200000   200000        0
    4 /.swap.virtual
                128,272    2      0        0        0        0    80000
    1 /dev/swap
                128,273    0      0   525632   456904   525632        0
```

The above example also shows *virtual swap* (vswap). vswap refers to an amount of swap space defined, but not available for physical use.

vswap is only on SGIs.
/usr/local/VSWAP and /.swap.virtual refer to vswap.
/dev/swap and /dev/dsk/dks4d2s7 refer to real swap on two separate hard drives.
The command

```
/etc/swap -s
```

may be used to determine the current usage of virtual memory on the system. It may be useful to use to find out how much or the virtual memory is currently being used and/or reserved for running programs to use.

```
prompt> /etc/swap -s
total: 33.56m allocated + 759.22m add'l reserved = 792.78m bytes used,
    303.47m bytes available
```

The above example shows 33MB allocated (only from swap space; since this is a non-zero number, all RAM is currently being used), 759MB reserved (not currently used but reserved by running processes) and 303MB available for new programs.
This example was performed on a very large system. More typically, the output would look more like

```
prompt> swap -s
total: 0.00k allocated + 68.73m add'l reserved = 68.73m bytes used,
    242.95m bytes available
```

The first three numbers refer to swap space only, while the last one refers to the virtual memory available (includes RAM + swap + vswap).
With linux:

```
prompt> /sbin/swapon -s
Filename                        Type            Size    Used    Priority
/dev/sda6                       partition       2048248 112356  42
/dev/sda7                       partition       2048248 112440  42
/dev/sda8                       partition       2048248 112372  42
/dev/sda9                       partition       2048248 112308  42
/dev/sda10                      partition       2048248 112324  42
/dev/sda11                      partition       1967920 112392  42
```

This shows that the linux system has 6 swap partitions, each roughly 2 GB in size, to total 12 GB swap space.

Note that for linux, RAM is directly mapped in the first portion of swap. If there is more swap than there is RAM, the additional swap may be used as extra RAM.

This is not the case with IRIX, where no RAM is directly mapped into swap: all swap is used as an extension to RAM.

9.4 Exercises

1. You are looking for file myfile in your directory tree. Which command could you use to locate it?
2. You wish to remove all occurences of the core files in your directory tree. How would you do it?
3. You have a program running, called myprog. It is caught in an infinite loop. Assuming it cannot be killed using CTL-C or stopped using CTL-Z, how can you eliminate it?
4. You would like to know how a system was behaving the last few hours. How would you do that?
5. You would like to know how a system is currently behaving. Which command(s) might you use?

Chapter 10

Solutions to Exercises

This section represents possible solutions to the exercises in the course material. The subsection numbers represent the section covered, and the solution number corresponds to the question number within that section. Solutions to each chapter of the book (except for Chapter 1) will start on a new page.

10.1 Introduction

1. UNIX is the operating system that covers the widest range of architectures. You can find UNIX on PCs (SCO, BSD/OS, FreeBSD, NetBSD, Linux), workstations (Solaris, IRIX, HP/UX, A/UX, AIX), mainframes (Amdhal and UNISYS have it) and supercomputers (UniCOS on the Cray).

   Why is it on so many architectures? As it is mostly written in the C language, it is very easy to port from one processor to another. It is found on proprietary chips as well as on the 80x86+Pentium, 68000 family, MIPS R{3,4,8}000 families, SPARC families, 88000 family, and others.

2. Some of the advantages of running UNIX are its multi-tasking, time sharing, multi-user and network capabilities. It is also very flexible, portable and has virtual memory. Thousands of programs are available for

it both in the commercial world and in the public domain.

3. UNIX's main philosophy is that it assumes the user knows what he/she is doing. If the user tells UNIX to erase his/her entire home directory, it will go ahead and do it... no questions asked!

4. People would use/develop software on linux because:

   - linux is free. It developing software is free.
   - linux may be installed on many different plaform, including any Pentium-based system, SPARC, PowerPC. It even runs on IBM mainframes!
   - linux includes (all free) web servers, mail servers, DNS server, and many other types of server software.
   - No money is going to Microsoft, while using better software :-)

10.2 File System

1. You cannot change permissions on someone else's file. You must be the owner of the file to do so.

   On Sun workstations,

   > chmod: *file*: Not owner

   will be the error message. On Silicon Graphics systems,

   > chmod: WARNING: can't change *file*

   will be echoed back.

2. Assuming I don't know what the present permissions are, the only command recommended is

   > chmod 744 *filename*

3. The two possible commands to change the permissions from rwxrwxr-x to rwxr-xr-x are

   - chmod 755 *filename*.
   - chmod g-w *filename*.

4. I simply want to disable the execute privilege on the file. The only command to be used is chmod go-x *filename*.

   I could not use chmod with octal values because I was not sure what the remaining permissions are.

10.3 Tour of the File system

1. The demonstration programs, if available on your system, could be in:

   - /usr/demos on Silicon Graphics systems.

2. Most of the commands used are found in the following directories:

   - /usr/bin.
   - /usr/bsd.
   - /usr/sbin.
   - /bin.
   - /usr/local/bin.

3. Most likely in /usr/share/lib/dict, file words.

4. On Sun workstations, your home directory would most likely be in /home/*machine_name*/*your_login*.

   On Silicon Graphics workstations, it would most likely be in /usr/people/*your_login*.

   For the other vendors, issue

   echo $HOME

10.4 Shells

1. The easiest way to do this is to use the input redirection sign as:

   names < *file_of_names*

2. This time, the input redirection would be used to read the data, then the output redirection would be used to capture what would normally be displayed on the screen:

   names < *file_of_names* > sorted.names

   To append to sorted.names, use >> instead of >.

3. I could use the rm command with a wild-card, as

   ```
   rm *.o
   ```

   (be careful when using *: type ls *.o before the rm *.o).

4. Again, use a combination of different types of wild-cards:

   ```
   rm [0-9]*[A-Z]
   ```

5. This time, use

   ```
   rm z???
   ```

6. rm my{fil,cap}e[1-4,7,9] is equivalent to:

   ```
   rm myfile1
   rm myfile2
   rm myfile3
   rm myfile4
   rm myfile7
   rm myfile9
   rm mycape1
   rm mycape2
   rm mycape3
   rm mycape4
   rm mycape7
   rm mycape9
   ```

7. To remove any file whose name contains a wildcard character, use the escape character (the backslash) as in rm nam*er.

8. I'm not interested in the actual output. All I want to know is the number of lines the program outputs.

   This could be done using the wc (word count) command, with the -l flag (that flag counts the number of lines in a file).

   ```
   display.dates | wc -l
   ```

   As display.dates is processed, its output is *piped* to wc -l, which displays how many lines it read.

9. I know that running that job interactively will take all the time I have left. So, I want to run it in the background. But I can't log off when I have a job running in the background...

   Use the batch command: it will send the job to the batch queue, and the results will be sent by mail. Meanwhile, I can work on that letter I have to write.

10. In the C shell, one recalls the last command previously executed by issuing

    ```
    !!
    ```

    The last command used beginning with la is recalled using

    ```
    !la
    ```

    In the last command 54 can be changed to 45 by issuing

    ```
    ^54^45^
    ```

    (*NOTE*: the command will immediately be executed).

    I can find out which aliases I am now using by typing

    ```
    alias
    ```

11. If the shell I am presently using can use the alias and the history features, I am certainly NOT in the Bourne shell, because those features are not supported in that shell.

    In many instances

    ```
    echo $shell
    ```

    will reveal which shell you are using.

12. One would use SSH on their system because SSH encrypts all communication between two systems, preventing people from *sniffing* the communications, including the exchange of login/password.

10.5 Security

1. I should use a password to prevent people from using my account, and corrupt some of the data I have.

   I should also use a password because my system is probably on a network, and anyone from the network could get into my account easily, gain special privileges, and corrupt not only my account but other accounts on the system.

   If the Internet Worm of November 1988 were reproduced today, our systems would be affected, just like the 6000 that were affected at the time.

   To see if any account does not have a password, simply look at the /etc/passwd file. The second field of that file is the encrypted password: if it is empty, no passwords are needed to get into that account.

2. Passwords should be at least five characters long, and contain one of the special characters (space, asterisk, comma, bracket, etc). The password used should NOT be a word contained in the on-line dictionary.

3. I am a system administrator for a group of workstations. Only myself, my backup and my supervisor should know the root password of that system. This will restrict the number of people that can do damage to the system: the fewer people involved, the better.

   Remember: root has super-user privileges. There is nothing root cannot do in the file system, including deleting EVERYTHING and/or corrupting the system. An inexperienced person could easily remove essential files. Even experienced people do!

   Also, one person should be the ‘‘major" administrator: that person should keep a log containing the changes made in the system, packages added, files modified, etc. That information becomes EXTREMELY handy, even essential, when the system crashes.

```
The system administrator should login as root ONLY when
performing functions that REQUIRE root privileges.
Otherwise, he/she should log into his/her personal
account:  this will prevent accidental deletions of
critical files.
```

10.6 Commands I

1. Options to commands are given using a dash followed immediately by the options desired.

   For example,

   ```
   ls
   ```

   lists all files within the current directory. Adding the option of listing all the files (including ones that start with a period) changes the command to

   ```
   ls -a
   ```

   If another option is added, the character representing that option is simply added to the string immediately following the dash, such as

   ```
   ls -as
   ```

   If a third option is required, the *option string* will then be three characters long.

2. Your password is changed using the passwd command. What follows is a transcript of what to do to change the password (*NOTE*: the password is actually NOT echoed back to the user):

   ```
   prompt> passwd
   Changing password for cantin
   Old password:  old_password
   New password:  new_password
   Re-enter new password:  new_password
   prompt> exit
   ```

   Most recent versions of the operating system will also not accept the password if it does not contain a non-alphabetic character.

3. Most systems have on-line manuals, accessible using the man command.

   To see how the rm command works, one simply types

   ```
   man rm
   ```

   and the syntax and description of the command is displayed on the screen.

   For many commands, issuing the command itself without any arguments returns its usage. rm is one such command:

   ```
   prompt> rm
   usage: rm [-rif] file ...
   ```

4. The dictionary should be in /usr/share/lib/dict, file words.

   There are many ways to find out how big that file is. Two ways will be shown here:

   ```
   wc words
   ls -l words
   ```

   root owns it.

5. Two commands could be used: pwd, and echo '$cwd'. Both will give the same answer, although echo '$cwd' will be done quicker since cwd is a variable whose value is the current working directory, whereas pwd has to go and calculate it.

6. One of the following three simple commands can be used to return to your home directory:

   ```
   cd
   cd $HOME
   cd ~
   ```

7. The first step is to create the directory. Once that's done, simply use cp to copy the files over:

```
cd
mkdir temp.dir
cd temp.dir
cp /etc/hosts ./
cp /etc/fstab ./
```

This is one way of using the cp command to do that. There are many variations, all as good as the one shown.

8. Again, create the new directory, then copy the first one into the second:

```
cd
mkdir temp2.dir
cp temp.dir/* ./temp2.dir
```

Another, simpler way of doing it is

```
cp -r temp.dir ./temp2.dir
```

but the results will be slightly different (temp.dir would be another directory within temp2.dir).

9. To remove the two directories along with their content,

```
rm -rf temp.dir temp2.dir
```

The rf flag will perform a recursive remove. If the file does not exist, or if you own the file but do not have write permission, it will not issue an error message.

10. The time a file was last changed can be found by doing a long listing on that file (ls -l list.names). The sixth column (on SGI's) will tell you the last time the file was updated. On Suns, this is the fifth field.

    That field will tell you the Month, Day and time. If the file is more than one year old, the format will be Month Day and Year.

11. *Hidden* files can be listed using the a option on the ls command (ls -a).

    All files beginning with a dot (.) will be displayed.

12. To create a file of length zero, use the touch command: this updates the time stamp of the file to "now" if the file exists, or creates the file with length zero if the file does not exist.

13. There is no ‘‘rename" command. Instead use the mv (move) command, as in

    ```
    mv myfile hisfile
    ```

14. To capture the long listing of /bin into hisfile, use the output redirection sign:

    ```
    ls /bin/* > hisfile
    ```

15. To look at the content of a file, one screenful at a time, use the more command: it will display one screenful worth of the file; pressing the space bar will display the next screenful; pressing the return key will display one additional line.

16. To see the first twenty lines of a file, use the head command, with the -*count* flag:

    ```
    head -20 hisfile
    ```

    No options on the command displays the first 10 lines.

17. The last fifteen lines can be seen with the tail command, using the -*count* option:

    ```
    tail -15 hisfile
    ```

    The default value again is ten lines. Experiment with different values. What happens if you put a + instead of the -?

18. To find out who is logged on to the system at this moment, issue the who command.

19. To communicate interactively, there are two commands to use, depending on whether both users can use full screen mode, or simply line mode.

    - In full screen mode, use the talk *user@node* command. This will split your screen in half, and send a message to the remote user you want to talk with. He/she will then answer back with the same command, and his/her screen will also be split horizontally in half.

      Your upper half of the screen reflects what you type, as you type it. This is the remote user's bottom half. When the remote user types on the keyboard, what he/she types is reflected on his/her upper screen, and your lower screen.

      To exit that mode, issue CTL-C.

    - In line mode, use the write command. This puts you in write mode: when you press RETURN, what you just typed is sent to the user. It stays in that mode until you hit CTL-D.

20. To print a file, two commands may be used:

    - lpr *filename*
    - lp *filename*

21. To remove print files from the print queue, one of two commands may be used, depending if the job is queue in a BSD, or System V type queue:

    - Use lpq or lpstat to see the jobs queued, and their JOB numbers.
    - Use lprm or cancel to remove the jobs from the queue. Make user you use the JOB number you got from the lpq or lpstat command.

10.7 Editors

1. You are at a VT100 terminal. Most UNIX systems understand what a VT100 terminal is, but they may not automatically know that you are a VT100 type terminal.

   To tell UNIX you are using a VT100 type terminal, issue

   ```
   setenv TERM VT100
   ```

   You could then use vi in its full screen mode. Other editors which may be available are emacs, edt and xe.

   If your UNIX system does not understand VT100 terminals, you would have to use a line editor, such as ed.

10.8 Electronic Mail

1. The name of the system you are using is usually found by issuing the command

   ```
   hostname
   ```

2. To send yourself some mail, simply use the mail command such as

   mail *my_logon*

   and fill up the subject line (if any). Write a line or two, then enter a dot (.) on a line by itself to finish the main body of the letter.

   You may then see a Cc:. The system is asking if you want to send a *complimentary copy* to someone else. Press the return key, and mail will be sent.

3. To send someone else a message, follow the same procedure as outlined above, but use their logon name for *my_logon*.

4. Assume the remote user is cantin and that his FULL system name is nickel.sao.nrc.ca. To send him mail,

   ```
   mail cantin@nickel.sao.nrc.ca
   ```

   is used, and the same procedure outlined above follows.

   BUT: your system has to be configured to use mail. If it is not, contact your system administrator.

5. When writing a message, a file can be inserted using

   ~r *filename*

   on a line by itself (that is ‘‘tilde"-r *filename*).

6. If the message to be sent is already in a file (let’s call it message), and my user name is cantin, I can send myself that file via mail using

```
mail -s "Subject of this Message" cantin <
message
```

7. Mail can be read using

   ```
   mail
   ```

   When reading mail, the current letter can be saved in the folder folder by issuing the s +folder command. The plus sign (+) means that the following file is to be a folder.

   To reply to yourself, simply enter

   ```
   r
   ```

   at the mail command line.

8. Any outgoing mail is saved in a file called .record in your own $HOME/mail directory. To send the mail to your friend, use the mail command to read the .record folder, save the message in a file by itself, and reply to your friend, using the content of the new file for the body of the message.

9. Issue elm to enter the package. From within the package, ‘‘o" will bring the menu option. Issue ‘‘u" to go to the user level menu, and press the space bar for your selection.

   Save your selection by pressing the greater than (>) sign.

10. Re-enter elm. Issue ‘‘a" to bring the aliases menu. Follow the directions elm issues.

10.9 Commands II

1. The find command would look like:

   ```
   find $HOME -name myfile -print
   ```

2. To find and remove a file:

   ```
   find $HOME -name core -exec rm {} \;
   ```

3. myprog has been running for 30 minutes now, and it should have run only 2 minutes.

   To remove the job, I can log in through another window, or system and issue the ps command (on SGIs issue ps -e; on Suns issue ps -asx). Look for the name of your running job in the last column of the output. Once you find it, look at its PID (process I.D.) number.

   To remove it from the system, issue

   ```
   kill -9 PID
   ```

   where PID is the process I.D. of that job.

4. To see how a system has been behaving for the last few hours, one would use the sar command.

5. But to see how a system is currently behaving, one might use any of the following commands:

   - sar 1 10, so show how the system is behaving for the next 10 seconds.
   - top, to see how the system is behaving AND what jobs are currently running.

Appendix A

vi Quick Reference

This table is taken from Sun Microsystems' User's Guide:
Getting Started (pp. 5-25 to 5-27).

	Starting vi
vi *filename*	open or create file
vi +18 *filename*	open to line 18
vi +/"mustard" *filename*	open file to first occurence of ‘‘mustard"
vi -r *filename*	recover crashed file
view *filename*	open file read-only

	Cursor Commands
h	move left
j	move down
k	move up
l	move down
w	move right one word
W	move right one word (past punctuation)
b	move left one word
B	move left one word (past punctuation)
Return	move down one line
Back Space	move left one character
Space Bar	move right one character
H	move to top screen
M	move to middle of screen
L	move to bottom of screen

Ctl-F	scroll forward one screen
Ctrl-D	scroll forward one-half screen
Ctrl-B	scroll backward one screen
Ctrl-U	scroll backward one-half screen
	Insterting Characters and Lines
a	insert characters to right of cursor
A	insert characters to right of cursor, at end of lin
i	insert characters to left of cursor
I	insert characters to left of cursor, at beginning o
o	insert line below cursor
O	insert line above cursor
	Changing Text
cw	change word (or part of word right of cursor)
cc	change line
C	change part of line to right of cursor
s	substitute string for character under cursor
r	replace character under cursor with one other chara
r-Return	break line
J	join current line and line below
xp	transpose character at cursor & character to right
	change case of letter (upper or lower)
u	undo previous command
U	undo all changes to line
:u	undo previous last-line command
	Deleting Text
x	delete character
dw	delete word (or part of word to right of cursor)
dd	delete line
D	delete part of line to right of cursor
:5,10 d	delete lines 5-10
	Copying and Moving Text
yy	yank or copy line
Y	yank or copy line

dd	delete line
p	put yanked or deleted line below current line
P (upper case)	put yanked or deleted line above current line
:1,2 co 3	copy lines 1-2 and put after line 3
:4,5 m 6	move lines 4-5 and put after line 6
	Setting Line Numbers
:set nu	show line numbers
:set nonu	hide line numbers
	Finding a Line
G	go to last line of file
21G	go to line 21
	Searching and Replacing
/*string*/	search for *string*
?*string*?	search backwards for *string*
n	find next (or previous) occurence of *string*
:g/*search*/s//*replace*/gc	search and replace, consult at each occurence
	Clearing the Screen
Ctrl-L	clear scrambled screen
	Inserting a File Into a File
:r *filename*	insert (read) *filename* after cursor
:34 r *filename*	insert *filename* after line 34
	Saving and Quitting
:w	save changes (write buffer)
:w *filename*	write buffer to *filename*
:wq	save changes and quit vi
ZZ	save changes and quit vi
:q!	quit without saving changes

Appendix B

Bibliography

Bibliography

[1] Stephen R. Bourne. *The UNIX System V environment*. Addison-Wesley Publishing Company. Don Mills, Ontario. 1987.

[2] D. Dougherty, R. Koman, and P. Ferguson. *The Mosaic Handbook for the X Window System*. O'Reilly & Associates, Inc. Sebastopol, California. 1994.

[3] E. Foxley. *UNIX for Super-Users*. Addison-Wesley Publishing Company. Don Mills, Ontario. 1985.

[4] Æleen Frisch. *Essential System Administration*. O'Reilly & Associates, Inc. Sebastopol, California. 1992.

[5] Ed Krol. *The Whole INTERNET User's Guide & Catalogue*. O'Reilly & Associates, Inc. Sebastopol, California. 1994.

[6] Jerry Peek, Tim O'Reilly, and Mike Loukides. *UNIX Power Tools*. O'Reilly & Associates, Inc. Sebastopol, California. 1993.

[7] H. McGilton and R. Morgan. *Introducing the UNIX SYSTEM*. McGraw-Hill Software Series for Computer Professionals. Toronto. 1983.

[8] R. Thomas, and R. Farrow. *UNIX Administration Guide for System V*. Prentice Hall. Englewood Cliffs, New Jersey. 1989.

[9] Silicon Graphics Inc. *IRIS-4D User's Guide*, man pages.

[10] Sun Microsystems. *SunOS 4.0, 4.1 Reference Manuals*.

[11] SuSE Linux LTD. *SuSE Linux 7.3 Reference Manual*.

[12] UNIX International. *The UNIX Operating System: A Commercial Success Story*. Nov 1, 1989. Parsippany, NJ.

[13] http://www.canarie.ca; November 1997 version.