An Introduction to Unix

Last Edit
March 24, 1992

Reg Quinton, Weimei Shyr, and Lori Corrin
Computing and Communications Services
The University of Western Ontario
London, Ontario N6A 5B7
Canada

Contents

1. History and Goals

Unix is a word play on MULTICS and owes much:

(1) The filesystem and interface to it.
(2) Command line interpretors are user processes.

MULTICS tried to be all things to all people. Unix has more modest goals — to be a programmer's environment for getting things done.

Unix is over 20 years old and still developing (eg. threads, Mach, C++, X11, Plan 9, etc.).

First version at Bell Labs in 1969 by Ken Thompson on a PDP-7, later joined by Dennis Ritchie (the authors of C). Developed as a "word processor" — a feint to avoid the bad taste left by MULTICS.

First public version released in 1971 (time on Unix starts at Jan 1, 1970, 00:00 GMT; see **ctime(3)**).

Unix Time Sharing System, Sixth Ed. (aka V6) first widely available in 1976. Seventh Ed. (aka V7 and UNIX/32V) 1978 designed as a portable system to run on a variety of platforms.

Early releases from Bell Labs *given* to other researchers in source form! Berkeley Software Distribution (BSD) started with UNIX/32V. Development on 32 bit machines with virtual memory made it possible to do lots!

At about 1985 ATT was split up (anti-trust laws, Reagan's notion of free enterprise). This allowed for the first commercialization of Unix by ATT.

Two important Unix streams are ATT System V (Release 4, not to be confused with 5th Ed.) and BSD (Release 4.3 Reno).

Political streams are Open Software Foundation (OSF includes IBM, Dec, etc.) and Unix International (Sun and ATT). Both streams have a common source and owe much to one another. Lately a third party, ACE (the Advanced Computing Environment).

Today Unix runs on a very wide set of platforms — AT through 386/486 class PC's, Motorola, Dec VAX, IBM mainframes, etc.

The principal virtue of Unix is that it is open and available on all these machines. The last system you'll ever need to learn or know!

2. Command Line Interpretors (Shells)

A shell is a command line interpretor that provides an interface to the Unix operating system. A shell is just another process on Unix.

> This means the Unix kernel needs to be concerned with very few things — managing I/O, scheduling jobs, IPC.

The usual shells are **/bin/sh** (the Bourne shell) and **/bin/csh** (the C shell). Most people use the Csh, system admins program with sh.

The C shell is very common these days (a BSD development). The notion of **job control** came with BSD Unix and the C shell, now very common under SysV and Bourne shell.

Common extensions are **/bin/ksh** (the Korn shell) and **/bin/tcsh** (Tenex shell). Both provide command recall and editting, CCS supports the Tenex shell (it’s a free ware extension to Csh and works!).

All shells take keyboard input, parse it into an argument list, and invoke utilities. In this course we’ll only talk about the Csh.

2.1. Login and Logout

Users login to a shell. You supply your user identifier and password.

```
EP/IX (julian.uwo.ca)

login: reggers
Password:
Last login: Fri Aug 30 13:15:59 from xchange.ccs.uwo.
julian.uwo.ca, EP/IX Version 1.4.1

All Suns are now YP integrated. Passwd and group
files are shared.
           reggers
TERM = (vt100) <cr>
[8:36am julian] whoami<cr>
reggers
[8:38am julian] pwd<cr>
/usr/people/guru/reggers
[8:39am julian] logout<cr>
shove off reggers
```

One of the first things new users should do is change their password to something memorable but hard for others to guess, see **passwd(1)**:

```
[8:39am julian] passwd reggers<cr>
```

2.2. Login Environment

When you login you have a current working directory (ie. location in the filing system), a controlling terminal (**/dev/tty**), and are running a shell (usually **/bin/csh** or some other if you’ve changed your shell).

```
[11:48am julian] pwd
/usr/people/guru/reggers
[11:48am julian] echo $SHELL
/bin/tcsh
```

The C shell, **/bin/csh**, (and C shell derivatives like tcsh) will execute commands stored in **.cshrc** and **.login**. At CCS these files are supplied and you shouldn’t need to touch them (but you should read them, become familiar with what they do).

These are available in the area

/usr/doc/proto

And available by anonymous ftp from

julian.uwo.ca:˜ftp/doc/proto

The directory **/usr/doc** is a link to **/usr/ftp/doc**. **˜ftp** is the login directory for user **ftp** (for anonymous ftp).

2.3. Login/Logout Problems

Type case —
Unix is case sensitive. User "**REGGERS**" is not the same as user "**reggers**". Likewise for passwords.

Upper case —
If you login as **REGGERS** instead of **reggers** Unix assumes you have an upper case only terminal. All input is translated to lower case, all output to upper case.

Account overdrawn —
talk to the accounting office.

Parity —
Some shells are fussy about parity. While the default is **even** there have been shells which require **none** (early versions of **tcsh**).

There are stopped jobs —
You've left some jobs running and can't logout. Type **fg** to bring these jobs to the foreground.

logout: not a login shell —
You're running in a subshell, type **exit** to get out.

2.4. Csh command line input

The shell prompts and you give commands

```
[10:54am julian] ls
Makefile  RCS     README   paper    sizing
slides    title
```

Commands look like

verb [arguments]

The shell executes the verb (or command) by

(1) construct an argument vector **argv** from the command line (after expanding meta characters)

(2) sets up **stdin**, **stdout**, and **stderr** (for input and output to the command) for the child.

(3) runs the appropriate child process (**fork(2)** and **exec(2)**) with the argument and environment vectors (**argv** and **argp**).

(4) waits for child process to terminate (**wait(2)**).

2.5. Csh meta characters

The Csh expands meta characters into file names, for example

```
[10:59am julian] echo *
Makefile RCS README paper sizing slides title
```

Note that Csh expands the wild card character when it constructs the argument vector. *It isn't the command that does that*! Unlike Vax/VMS.

***** matches any string (in a file name)

? matches one character (in a file name)

[abc]
matches any of the characters specified

[a-z]
matches any in the range specified

~user
the home directory of the specified user (or your home home directory if none mentioned).

3. Customizing the Csh Environment

The Csh environment can be customized to suit your needs. You can customize the environment by instructing the Csh from **.login** and **.cshrc**.

Our intention at CCS is to provide you with a fully functional environment where you *can* customize your environment if you need to. However, we're trying to give you an environment that you *don't need* to customize.

It is difficult, if not impossible, to support many different environments. Consultants often find that users have problems because they have deviated from the standard environment (and no longer have a good search path, or the right variables set, or etc.).

Beware:
In short, don't customize your environment unless you know what you're doing!

3.1. Csh Customizing Commands

3.1.1. **alias** *verb value*

Alias is a **builtin** command. You can alias new commands to existing ones. For example,

```
[2:19pm julian] alias
bye     exit
cp      cp -i
delete  rm
dir     ls
ed      emacs
ls      ls -lasFC !* | more
mv      mv -i
rename  mv
plotq   lpq -Pplotter; lpq -Pgplotter; lpq -Psplotter
type    more
unshar  sed "1,/^#/d" !* | sh
[2:19pm julian] alias print lpr
```

Commands like **cp**, **mv** and **rm** take a flag to help protect you from making mistakes.

```
[2:19pm julian] alias cp cp -i
[10:28am julian] cp foo2 foo
overwrite foo? y
```

3.1.2. **set** *variable[=value]*

Set is a **builtin** command. C shell variables control the behaviour of the Csh (eg. “history” defines the number of commands kept in the history stack). Usually these variables are set in .cshrc.

```
[12:33pm julian] set
...etc.
cwd     /usr/people/guru/reggers
edit    emacs
history 40
home    /usr/people/guru/reggers
noclobber
notify
path    (/usr/people/guru/reggers/bin ... etc.
prompt  [%S%t%s %m]
term    xterm
...etc.
[12:46pm julian] set prompt="% "
% echo $term
xterm
```

The **noclobber** prevents me from wiping out files with redirection, **notify** tells me when background jobs are done, **history** to maintain a history of what’s been done.

Note: you refer to variable values with the dollar notation.

3.1.3. **setenv** *variable[=value]*

Setenv is a **builtin** command. Environment variables are different from C shell variables. These are passed on to all children (ie. commands) which may use them (eg. lpr honors the **PRINTER** environment variable). C shell variables are specific to the C shell and cannot be passed on to children.

By convention environment variables are usually upper case, C shell variables are usually lower case.

```
[12:44pm julian] printenv
...etc.
EDITOR=emacs
HOME=/usr/people/guru/reggers
HOST=julian.uwo.ca
MANPATH=/ccs/export/man
PATH=/usr/people/guru/reggers/bin:... etc.
PRINTER=draft
SHELL=/usr/local/tcsh
TERM=xterm
TZ=EST5EDT
USER=reggers
TERMCAP=xterm|...  etc.
[12:46pm julian] setenv PRINTER apple
```

3.1.4. The Unix Process Environment

All processes (or jobs) have environment variables (*fundamental* to Unix). Csh lets you define the environment you pass on to the jobs you run.

C programs always have this fragment:

```
main(argc,argv,argp)
int    argc;       /* argument count */
char   *argv[]; /* argument vector */
char   *argp[]; /* environment vector */
```

You refer to environment variables with the dollar notation.

```
[12:46pm julian] echo $PRINTER
apple
```

Csh variables are not passed on to processes (or jobs) that are run from the Csh.

Hence, the file **.cshrc** which is executed for each instance of the Csh. That's how you share Csh variables, aliases, etc.

3.1.5. The Csh Search path

The command line you type, eg.

```
[10:41am julian] ls -l .login
  5 -rw-r--r--  1 reggers      4223 Apr 16 13:58 .login
```

is parsed by the shell into a vector of strings

verb [argument [argument ...]]

The **verb** you've indicated will either correspond to a Csh **builtin** command (ie. the shell does it) or to some executable program within your *search path*.

```
[10:41am julian] which printenv
/usr/ucb/printenv
[10:44am julian] which setenv
setenv: shell built-in command.
```

The Csh **path** variable is the *search path*. This is stored internally in a **hash** table (and explains the **rehash** builtin).

3.1.6. Beware: Search Path

At CCS we support a search path that gives a BSD rather than SysV feel. Changing the search path can radically change your environment; eg.

```
[10:49am julian] unset path
[10:49am julian] which printenv
printenv: Command not found.
[10:49am julian] logout
```

Or, compare the behaviour of BSD stty to SysV stty.

Note that setting your Csh **path** variable will have the side effect of also setting the **PATH** environment variable. Many are tightly coupled — but not all.

BEWARE:
Don't change your search path unless you know what you're doing!.

3.1.7. umask *nnn*

Umask is a **builtin** command used to set a file creation mask. A fundamental notion to Unix (all jobs have a file creation mask, just as all jobs have a set of environment variables). The umask is usually set in **.login**. Note: it's a mask to define what should *not* be allowed.

```
[10:59am julian] umask 027
```

The number is 3 digit octal. The first digit is for the **user(u)**, the second for the **group(g)**, and the last for **others(o)**. **read(r)** access is 4, **write(w)** is 2, and **execute(x)** is 1.

```
[10:59am julian] umask
27
[10:59am julian] touch foo
[10:59am julian] ls -l foo
  0 -rw-r-----  1 reggers        0 May  9 10:59 foo
[10:59am julian] umask 77
[11:00am julian] touch goo
[11:00am julian] ls -l goo
  0 -rw-------  1 reggers        0 May  9 11:00 goo
```

Umask 027 (the default at CCS) lets the owner do everything (none of **rwx** is denied), the group can only read/execute (**w** is denied), and others have no access.

3.1.8. cd *directory*

Cd is a **builtin** command. All jobs have a current working directory (where am I in the Unix file system? what does an unqualified file name mean?). This is a fundamental notion of Unix.

Two commands for manipulating and displaying your working directory are **cd** to change directory and **pwd** to print the working directory.

```
[11:08am julian] pwd
/usr/people/guru/reggers
[11:08am julian] touch foo
[11:08am julian] ls -l foo
   0 -rw-r-----  1 reggers        0 May  9 11:08 foo
[11:08am julian] cd /etc
[11:08am julian] pwd
/etc
[11:08am julian] touch foo
touch: could not create foo: Permission denied
```

I cannot create a file in /etc because the directory is protected against me (and others). Your home directory should be protected so you can create files, other directories likely aren't (and shouldn't be).

3.1.9. stty

The **stty** command defines terminal characteristics (eg. what character does a rubout?). This is usually done once in **.login**.

```
[12:53pm julian] stty everything
new tty, speed 9600 baud, 70 rows, 80 columns
even odd -raw -nl echo -lcase -tandem -tabs -cbreak
crt: (crtbs crterase crtkill ctlecho) -tostop
-tilde -flusho -mdmbuf -litout pass8 -nohang
-pendin -decctlq -noflsh
erase  kill   werase rprnt  flush  lnext  susp   ... etc.
^?     ^U     ^W     ^R     ^O     ^V     ^Z/^Y  ... etc.
[12:53pm julian] stty erase ^H
```

Note the control characters **^C** for interrupt, **^D** for end of file, **^?** for delete character, **^U** for delete line, **^R** for retype, **^S,^Q** for gag, **^O** for stop output. These are very familiar to Digital users (eg. Vax/VMS).

One important character is **^Z** for suspend process (*no it doesn't mean end of file!*). Unlike Vax/VMS.

3.1.10. tset

The other terminal manipulation command is **tset** — this is what prompts you for your terminal type at login:

```
TERM = (vt100)
```

Tset does some setup and, with some work, defines the environment variable TERM which is required for full screen applications like vi and emacs — they have to know the terminal type.

From .login

```
eval ‘tset -s -Q -I ’?’$TERM‘
```

BEWARE:

Don’t muck with the stty and tset in your .login; you can get in serious trouble!.

3.2. Configuration Files

By convention, startup and customization files are found in the user’s login directory and hidden as “dot” files. Within CCS we provide new users with most of these.

.cshrc
each instance of /bin/csh will execute the commands in .cshrc. This means that they’re executed when you login, also when you start up Csh (eg. from within Mail). Put **set** commands here.

.login
executed once at login after .cshrc. Set some environment variables, and terminal hassles. Put **setenv** commands, **stty**, and **tset** commands here.

.logout
executed at logout. For example,

```
[12:10pm julian] page .logout
echo shove off $user
```

.aliases
Within CCS the supported .cshrc will source the commands in .aliases. This makes it possible for users to customize their environment while preserving all of .login and .cshrc.

.forward

direct your mail to some other site. For example,

```
[12:11pm chaplin] page .forward
reggers@uwovax
```

.mailrc

customize your mail environment. Define mailing lists, editors, etc.

.rhosts

specify other systems where the same user can rlogin (rsh and rcp) without going through Password: prompting. For example,

```
[12:10pm julian] page .rhosts
grafxa.uwo.ca reggers
grafxb.uwo.ca reggers
julian.uwo.ca reggers
shadow.ccs.uwo.ca reggers
myst.ccs.uwo.ca reggers
hadrian.ccs.uwo.ca reggers
icarus.ccs.uwo.ca reggers
hydra.uwo.ca reggers
```

Note: there are some security problems with this file (eg. don't let anyone write to it!).

4. Help — where to find it

Most Unix systems usually have a full set of on line manuals organized in sections as:

1. User Commands (eg. mail)

2. System Calls (eg. getuid)

3. Library Routines (eg. sqrt)

4. Devices/Special Files (eg. tty)

5. File Formats (eg. a.out)

6. Games (eg. banner)

7. Miscellaneous (eg. filesystem)

8. Systems Administration (eg. fsck)

These correspond to the directories /usr/man/man{1-8} and /usr/man/cat{1-8} (source and output of manuals). But watch out for System V machines. As well, people often store locally developed software into a local manual section — section **l**.

4.1. Help Commands

help
some Unix systems have an on line help system (especially System V machines). On BSD system this is usually just a hint to read the manuals.

man *entry*
display the manual page entry for some entry.

man -k *topic*
what manual pages are about a particular topic? Ie. look up this topic in the index. The command **apropos**, if it exists, does the same thing.

whereis *file*
show locations of a file (actually this only searches a few places, not the entire file system).

which *command*
when I type a command which one am I using?

mail *postmaster, usenet, ccs, thugs*
You can usually get help by E-mail. The **postmaster** answers questions about mail, **usenet** for news, **ccs** should get help from the consultants desk, and **thugs** is the Unix group.

Pnews
Post gripes, questions, etc. to **uwo.ccs.talk** or **uwo.comp.unix**. Alternatively, try **postnews**. You can even **mail** to **uwo.ccs.talk** — your mail will be posted to the group.

cd /usr/doc
On line documentation is stored here. Start with the README.

cd /usr/doc/FAQ
On line Frequently Asked Questions (or FAQ's) are stored here. Start with the README and the files in `unix-faq`.

5. The Unix File System

The Unix file system is a rooted tree of directories and files. A directory is a special file that will contain links (or pointers) to other files and directories.

5.1. Fundamentals

Files are allocated space (called **inodes**) on the disk.

attributes
Inodes have *attributes* — size, date, owner, group, protection, etc.

numbers
Inodes are **numbered**. There is no name or place in the directory tree as an attribute of the inode.

contents
Inodes have *contents* or data.

links
Inodes are *linked* into directories. The directory entry contains the *file name* and a pointer (or *link*) to the inode number.

5.2. Links and Names

Note, the *name* is not an attribute of the file! When you create a file (eg. by **touch(1)**) you

(1) allocate an inode for the file contents and attributes.

(2) create a link in a directory to give a file name to the inode.

An inode can have several links. An inode is removed (in the sense of disk space retrieved) when the link count is zero. The **rm(1)** command doesn't remove files, it unlinks them.

```
[9:42am julian] ls -l foo
  1 -rw-r-----  1 reggers       33 May 10 09:42 foo
[9:43am julian] ln foo goo
[9:43am julian] ls -l goo
  1 -rw-r-----  2 reggers       33 May 10 09:42 goo
[9:43am julian] rm foo
[9:48am julian] ls -l goo
  1 -rw-r-----  1 reggers       33 May 10 09:42 goo
```

The **mv(1)** command doesn't move files, it links and then unlinks them.

5.3. The Current Working Directory

File specifications are either fully qualified or rely on the notion of the *current working directory*. This is *fundamental* to Unix and applies to *all* processes (eg. your shell or the **chmod(1)** command).

The login process sets your current working directory to your login directory (as specified in **/etc/passwd**).

```
[11:19am julian] pwd
/usr/people/guru/reggers
[11:19am julian] echo $cwd
/usr/people/guru/reggers
[11:19am julian] touch foo
[11:28am julian] ls -l foo
  0 -rw-r-----  1 reggers        0 May  9 11:28 foo
[11:28am julian] ls -l /usr/people/guru/reggers/foo
  0 -rw-r-----  1 reggers        0 May  9 11:28 /usr/people...
```

In the first case the current working directory is assumed, in the second none is assumed. It's Unix, not the shell, that takes care of resolving file names.

5.4. Dot directories

Each directory has two special subdirectories — "." is a link to the directory, ".." is a link to the parent of the directory. These redundant links help to hold the file system together (so that it can be recovered when a link is broken). They also help for walking around within the tree.

```
[11:28am julian] pwd
/usr/people/guru/reggers
[11:29pm julian] ls -l
total 561
  3 drwxr-xr-x 17 reggers     3072 May  8 15:04 ./
  1 drwxrwxr-x 10 root         512 Apr 26 13:50 ../
...etc.
[11:35am julian] cd ..
[11:35am julian] pwd
/usr/people/guru
[11:35am julian] cd .
[11:35am julian] pwd
/usr/people/guru
[11:35am julian] cd ./reggers
[11:36am julian] pwd
/usr/people/guru/reggers
```

5.5. Dot is a real file

A lot of people think that “.” and “..” are meta syntactic notions of the Csh — they are real files in the filing system.

Compare with the Csh notion “˜*user*” — the home directory of a particular user.

```
[11:22am julian] echo ..
..
[11:22am julian] echo ˜
/usr/people/guru/reggers
```

Dot directories are a fundamental notion of the Unix filing system, and processes can use them; they’re really there!

There is no file called **˜reggers**; that’s only a notion of the shell.

Compare with **.login**.

5.6. Directory Delimiters

Unix file names are arbitrary strings of characters — watch out for the file called “*”. The character “/” is *reserved* as a delimiter for directories.

Compare foo/bar with foo.bar

foo/bar
The file named **bar** within the directory called **foo** within the current working directory.

foo.bar
The file named **foo.bar** in the current working directory.

In the first case you have two files, in the second only one.

Note you can have files with lots of dots in the them; eg. .login.old. Compare with some systems where there is a notion of a file-name and file-extension.

The *only* special character is the directory delimiter. But watch out for shell meta characters when specifying a file name! eg. you can have a file called **˜reggers** but it is hard to talk about with the Csh.

5.7. Root

The root of the Unix file system is the directory “/”. A fully qualified file specification starts at the root of the tree. The current working directory is always a fully qualified directory specification.

```
[11:36am julian] pwd
/usr/people/guru/reggers
[11:36am julian] cd /
[11:37am julian] ls -ld usr
  1 drwxr-xr-x 29 bin        1024 Apr 30 10:20 usr/
[11:38am julian] cd usr
[11:39am julian] ls -ld people
  1 drwxr-xr-x 75 bin        1024 Apr 30 10:20 people/
[11:40am julian] cd people
[11:40am julian] ls -ld guru
  1 drwxr-xr-x 39 bin        1024 Apr 30 10:20 guru/
[11:41am julian] cd guru
[11:50am julian] ls -ld reggers
  1 drwxr-xr-x 24 bin        1024 Apr 30 10:20 guru/
[11:54am julian] cd reggers
[11:54am julian] pwd
/usr/people/guru/reggers
```

5.8. A Quick Tour of the Unix Filing System

Some important directories in the Unix file system (for more detail see the manual page **hier(7)**)

/
The root of the tree. All file specifications start here. This is usually the login directory for the user called **root**. An important file is **vmunix** — the Unix kernel!

/bin, /usr/bin, /usr/ucb
Binaries, eg. **chmod**. These are executeable program areas. Watch out for /bin/mail, /usr/bin/mail, and /usr/ucb/mail. Different programs all with the same name.

/dev
Devices, eg. **tty**. This area is only important to the system admin. The Unix filing system is *mounted* on devices.

/etc, /usr/etc
Various administration files, eg. **passwd** and **group**.

/lib, /usr/lib
Library information, eg. **libc.a** and **terminfo**.

/tmp, /usr/tmp
Temporary files, eg. while editting, compiling, etc.

/usr/local, /usr/local/bin, ...
A tree for locally developed tools utilities, etc.

/usr/doc
Documentation, eg **introunix.txt**.

/usr/man
Manuals, eg. **cat1/ls.1** and **man1/ls.1** (the BSD organization has manual sources on line).

/usr/people, /usr/people/guru, /usr/people/guru/reggers
The user area, where people are located.

/usr/spool
For files that are spooled for service. eg. **lpd** for the line printer daemon, **news** and **mail**.

/usr/src
The Unix tradition is to have sources on line, eg. **usr.local** is our tree of local software sources.

5.9. File protections

File protection codes are attached to each file (fundamental to the Unix Filing System). These control **read(r)**, **write(w)**, and **execute(x)** access to the files. Each of these can be specified for the **user(u)**, **group(g)**, and **others(o)**.

```
[12:29pm julian] ls -lg paper
  74 -rw-r-----  1 reggers  guru  75742 May  9 12:17 paper
```

The file belongs to user **reggers** and group **guru**. The various protections are:

User: rw-
The user can read and write the file. It isn't executable.

Group: r--
Anyone in group **guru** can read the file. They cannot write, they cannot execute.

Others: ---
Others have no access at all.

Note: to delete a file you have to write into the directory (to create a link).

5.10. Changing protections

Changing file protections is done with **chmod(1)** and you can only do this to your files.

```
[12:31pm julian] chmod o+r paper
[12:34pm julian] ls -lg paper
  74 -rw-r--r--  1 reggers  guru  75742 May  9 12:17 paper
```

Your default protection mask can be changed with **umask**.

```
[2:53pm julian] umask 027
[2:53pm julian] touch foo
[2:53pm julian] ls -l foo
   0 -rw-r-----  1 reggers        0 Dec 31  1969 foo

[2:53pm julian] umask 077
[2:54pm julian] touch goo
[2:54pm julian] ls -l goo
   0 -rw-------  1 reggers        0 May  9 14:54 goo

[2:55pm julian] umask
 077
```

The default **umask** supplied by CCS is **027** — this gives the group some access, but other no access.

6. Basic Unix Utilities

6.1. Access to System

login [*user*]
login to system, this terminates current session.

su [*user*]
substitute user id. Temporarily become specified user. If no user id given, then become **root**. A Password: prompt, of course.

logout
terminate login session.

exit
terminate shell, if this is the login shell then logout.

6.2. Directory Commands

ls [*dir***]**
list contents of directory. If none specified then list current working directory.

cd [*dir*]
change current working directory. To **$home** if none specified.

pwd
print current working directory (where am I?).

mkdir *dir*
make a directory.

rmdir *dir*
remove a directory (it has to be empty).

mv *file [file ...] dir*
move file(s) into directory.

ln *file1 file2*
make a directory entry *file2* — a link to *file1*.

6.3. User profile

passwd [*user*]
change password for user (usually just for yourself). See also **yppasswd(1)** on YP/NIS environments.

chsh [*user*]
change login shell for user (usually just for yourself). eg. to change to **tcsh**. Same as **passwd -s**.

chfn [*user*]
change personal information (name, phone number, etc.). Same as **passwd -f**.

finger *user[@domain]*
display personal information (name, phone number, etc.).

who
who is logged in, what are they doing?

6.4. File Display

To display a file on your terminal there are several commands

cat *file*
display file, no pagination. Useless on a fast terminal.

more *file*
display a file a page at a time. Required for fast terminals. Interfaces to **vi(1)**.

There are several variations on **more**. **Page** is usually the same command (a link). **less** and **pg** are SysV variations that do the same.

pr *file*
display file with date, title, page number, etc. This doesn't paginate, pretty useless. But nice for listings.

6.5. File Manipulation

ls *file*
list file, the **-l** flag will give a long listing with size, date, link count, etc.

chmod *mode file*
change the file mode (ie. protection).

cp *file2 file2*
copy file *file1* to *file2*.

mv *file2 file2*
move (ie. rename) file *file1* to *file2*. (Or move into directory *file2*).

rm *file*
remove (ie. delete) file *file*.

ln *file1 file2*
make a directory entry *file2* — a link to *file1*. (Or link into directory *file2*).

file *file*
what kind of file is *file*?

6.6. File tools

grep *string file*
search for string *string* in file *file*. The history of this name is the **ed** command "**g/***re***/p**".

diff *ffile1 file2*
report differences between files *file2* and *file2*.

head *file*
display head of file *file*.

tail *file*
display tail of file *file* (the **-f** flag means follow the file).

wc *file*
count characters, words and lines in *file*.

find *directory* **-name** *name* **-print**
search the entire directory structure looking for a file, print out what you find. Powerful, but a bit cumbersome.

6.7. Printer commands

There are two different printer systems — System V and BSD. On CCS systems we support BSD printing (this is by far the most common on campus). Common commands are:

lpr *[-P printer] [file]*
print file on line printer (historically **pr** is a program to create paginated output with titles, etc. and **lpr** used **pr** to display listings).

lpq *[-P printer]*
display contents of printer queue

lprm *[-P printer] job*
delete job number *job* from queue

lpc *[-P printer] [command]*
Line printer control. For the system administrators.

6.8. Compilers and languages

6.8.1. C

C programs are stored as files ending in “.c” C compiler is **cc**. The **make(1)** command knows all about building programs from C source:

```
[5:30pm julian] ls test.c
test.c
[5:30pm julian] make test
        cc  test.c  -o test
[5:30pm julian] ./test
```

6.8.2. Fortran

Fortran programs are stored as files ending in “.f”. Fortran compiler is **f77**. Again, **make(1)** knows how.

```
[5:31pm julian] ls fort.f
fort.f
[5:31pm julian] make fort
        f77  fort.f  -o fort
[5:31pm julian] ./fort
```

6.8.3. Other languages

There are lots of programming languags:

Shell programming — system admins often program using **/bin/sh** and write scripts.

Don’t use **/bin/csh** for writing scripts — not recommended.

Yacc — yet another compiler compiler.

Lex — lexical analysis front end for yacc.

Awk, sed — string manipulation languages.

C++, Objective C, Perl, and lots more!

6.9. Editors (ed, vi, ex)

There are a number of editors on Unix

(1) **ed/sed/ex/edit**, simple line based editors.

(2) **vi**, full screen (built on **ex**).

(3) **emacs**, full screen (recommended).

To fully describe any editor is a course in itself. Will briefly describe each.

See attached quick intro documents.

6.10. ed/sed/ex/edit

These are line based, easy to use, pretty much the same, and worth knowing. If you can handle **edlin** on that other environment you can handle these.

Each runs in command mode. Commands look like

[<range>]<command>[<arguments>]

Commands are usually single characters, eg. **d** for delete, **p** for print.

The range specifies which lines to apply the command, with none given the current line is used. Use line numbers.

Special meta characters are available for specifying lines, eg. **$** for last line, **.** for current line, and ^ for first line (cf. meta characters for **Mail(1)**).

6.11. Ed, an example

Ed(1) is the original line based editor. Some find it irritating that it doesn't prompt for commands (Unix is terse).

```
[9:53am julian] ed foo
?foo      ... foo doesn't exist
a  ... append text
some text, line 1
more text, line 2
.
```

You append text to an empty file (or insert in an existing file) and terminate with a dot.

```
[9:53am julian] ed foo
1,$p  ... print all lines
some text, line 1
more text, line 2
1s/some/first/  ... substitute on first line
p      ... print current line
first text, line 1
w      ... write
37
q      ... quit
[9:54am julian]
```

6.12. vi

Vi was one of the first full screen editors. It's built on **ex** so if you can handle that, you can handle vi. But, these days you likely don't know the line editor.

Vi operates in one of three modes.

Command Mode

In command mode vi is ready to accept commands from the keyboard. Commands are printable characters, usually single characters.

In command mode vi will beep at you if you enter a bad command (eg. you mean to enter text but have not instructed vi to do so).

Ex Mode

In Ex Mode (eg. from the ":" command) you can enter any valid **ex** command. The command **:wq** will write your file and terminate yoursession.

Insert Mode

In insert mode (eg. from the "i" command) you enter text and terminate with an *escape* character.

6.12.1. Some vi commands

h,j,k,l
historical commands for left, down, up, right (arrows should work on most terminals).

i,a
insert and append text. Text is entered and terminated with **Esc**.

o,O
open a new line after (before) this line.

r,cw,cc
replace single character, word, line.

x,dw,dd
delete character, word, line.

/*string*
search forward for string.

:*command*
talk to **ex(1)** to execute commands, eg. **:wq** to write and quit.

ZZ
to write and quit, same as **:wq**

6.13. Emacs

Emacs(l) is the latest and greatest full screen editors. It’s built on a **lisp** interpretor and is very powerful. Variations are **jove** (Jonathon’s own visual editor) and micro versions. Emacs is freeware and the cornerstone of the GNU project.

6.13.1. Insert Mode

Unlike **vi** emacs is *always* in insert mode, any text you type will be inserted into the file. To instruct emacs to do something else requires some effort. Most people find this design better.

6.13.2. Commands

Commands are always at least two key strokes and will always require either the **control** or **meta** characters (on most keyboards the meta character is the escape character but X11 terminals have extra characters like meta and alt).

The command to exit emacs is **c-x c-c** (type control X followed by control C).

Emacs has a builtin help system, start with **c-h** and follow the instructions (easier said than done).

6.13.3. Some emacs commands

c-x c-c
write file and leave emacs.

c-h
get help. **c-h t** for the emacs tutorial (learn by doing).

c-p,c-n,c-b,c-f
Control sequences for previous/next line, back/forward character (also arrow keys).

c-s,c-r
Search forward, search backwards (not that control S is often the gag character).

c-v, m-v
scroll forward, back.

c-d, m-d, c-k, m-k
Delete character, word, line, sentence.

c-y
yank text, eg. delete and yank to move text.

Emacs can support multiple edit sessions, very powerful.

7. Communications Utilities

7.1. Sending Mail

To send mail you need to know the person’s address. **finger** is a useful tool but the best is the telephone.

```
[10:46am julian] mail purple
Subject: just a test
Hello how are you today?
.
Cc:
```

To send mail you supply the address and the mail utility prompts you for a subject and message. For help on addressing conventions see **mailaddr(l)**.

In brief, remote addresses (ie. for people at other sites not on this machine) look like:

user@do.ma.in

For example, **reggers@uwovax.uwo.ca**.

During message composition the dot signals end of the message (so would **^D**). It’s important not to start a line in the message with a dot.

7.2. Receiving mail

Csh is usually configured to watch your mail box. At login, or when mail arrives, you’ll see:

```
You have new mail
```

To read your mail try the **Mail(1)** command (actually there are tons of mail readers, even one within emacs!).

```
[10:47am julian] mail
Mail version 5.3 2/18/88.  Type ? for help.
"/usr/spool/mail/reggers": 1 messages 1 new
>N  1 purple@julian.  Fri May 10 09:14  18/760  "ria"
&
```

For help on the various mail commands type **help** (or look at the manual page).

Mail should be easy to use (especially if you’ve used mail on our Vax/VMS service).

7.3. Chatting

See **talk(1)** and **write(1)**. The first sets up a split screen where you type in one half, your colleague in the other.

7.4. File transfer

See **ftp(1)** and **rcp(1)**. The first is available on most machines, rcp is restricted to Unix machines under the same management (ie. /etc/hosts.equiv).

7.5. Remote login

See **telnet(1)** and **rlogin(1)**. The first is available on most machines, rlogin is specific to Unix.

7.6. News

News or Usenet is a facility for talking to other people about subjects. The other people can be anywhere in the world, and the subjects can be pretty well anything. Usenet is traditionally organized with a tree structure organized by function. One tree for Computer topics, another for Recreational topics, etc.

For example

> **comp**: Topics of interest to both computer professionals and hobbyists, including topics in computer science, software source, and information on hardware and software systems.
>
> **uwo**: The "uwo" news tree is for newsgroups local to the campus of the University of Western Ontario. See especially **uwo.general**.

The complete set of newsgroups is alt, bionet, bit, can, comp, gnu, misc, news, ont, rec, sci, soc, talk, uwo, and vmsnet. In total there are over 1000 different newsgroups.

Each newsgroup is a hierarchy of subgroups.

For a complete listing and for for information on the groups see *List of Active Newsgroups* posted each month to news.announce.newusers, news.lists, news.groups.

For example, some groups in the comp.sources tree:

comp.sources.amiga
Source code-only postings for the Amiga.

comp.sources.apple2
Source code and discussion for the Apple2.

comp.sources.atari.st
Source code-only postings for the Atari ST.

And, of course, lots of comp.sources groups for Unix!

7.6.1. Rn (Read News)

Rn is a good news utility. It will setup a history file (**.newsrc**) and subscribe you to all groups. In your first session unsubscribe from many groups.

```
[11:13am julian] rn
Trying to set up a .newsrc file--running newsetup...
Creating .newsrc in /usr/people/guru/purple to be used by
news programs.
Done.
```

If you have never used the news system before, you may find the articles in **news.announce.newusers** to be helpful. See **rn(l)** for more information.

8. Advanced Topics

Some of these topics are fundamental, others advanced.

8.1. I/O redirection

All jobs under Unix should have three I/O units set up — **stdin** (standard input), **stdout** (standard output), and **stderr** (standard error). These are usually attached to your terminal (when you login your shell has these attached to your terminal).

When you direct the shell to run a command it sets up **stdin**, **stdout**, and **stderr** for the command and then runs the command.

Csh can redirect **stdin**, **stdout**, and **stderr** to files just as easily as to the terminal.

```
[1:11pm julian] sort <names >names.sorted
```

8.2. Redirection Summary

command <file
Run the command with input from file rather than the terminal.

command >file
Run the command with output to file rather than the terminal.

command >>file
Run the command with output to end of file rather than the terminal.

command >&file
Run the command with output and error to file rather than the terminal.

The **noclobber** setting will prevent you from directing output to files that already exist.

command >!file
Run the command with clobber output to file rather than the terminal.

8.3. Piping

The novice might say

```
[1:01pm julian] make >make.log
[1:02pm julian] lpr make.log
```

But Csh can be instructed to run several commands at once or, more importantly, to run several commands in sequence with output from one as input to the next. This is called piping.

The piping operator, to connect commands, is the vertical bar, “**|**”. In general the format is

command [| command ...]

This instructs Csh to connect **stdout** of one command to **stdin** of another. For example

```
[1:02pm julian] make | lpr
```

8.4. More Piping

The operator “**|&**” will connect **stdout** *and* **stderr** to **stdin**. For example

```
[1:01pm julian] make |& lpr
```

The command “**tee**” (see **tee(1)**) is useful for seeing what is happening in a pipeline. For example

```
[1:01pm julian] make |& tee /dev/tty | lpr
```

This will push **stderr** and **stdout** to your terminal (that’s always **/dev/tty**) as well as to the line printer.

8.5. Job Control

Unix is a multi tasking environment. Users can control jobs. This is a powerful *feature* (which new users may find annoying).

```
[2:12pm julian] logout
There are stopped jobs.
... cannot logout until you deal with them!
```

To see some of the processes you have use **ps(1)** or the Csh builtin **jobs**. For example:

```
[8:21am julian] ps
  PID  TT STAT  TIME COMMAND
 1885  q2 S     0:12 emacs slides
 1285  q2 S     0:01 tcsh
 2513  q2 R     0:00 ps
```

Or, alternatively, print out the job table your shell knows about:

```
[8:36am julian] jobs
[1]  + Running               emacs slides
[8:36am julian]
```

8.5.1. Fore/Background Jobs

Usually a job is run in the foreground — Csh stands back and lets the job communicate with you through the terminal. You can instruct Csh to run jobs in the background, so that you can do other things while your job is working. Background jobs will be **stopped** when they require keyboard input.

The ampersand at the end of line is an instruction to run in the *background*. Only one job can run in the *foreground*, lots in the background.

```
[1:30pm julian] du >usage &
[1] 9491   job number and process id
[1:30pm julian]  work, work,  ... etc.
```

When your job is done, if you have **set notify**, you'll get a message:

```
[1:38pm julian]
[1] - Done       du >usage
```

8.5.2. Suspending a job

You can **suspend** any job. The keyboard input **^Z** sends an interrupt signal to suspend a the current job.

```
[1:30pm julian] du >usage
... working, working, ...
^Z
Stopped (tty input)
[1:33pm julian]
```

A job that has been suspended can be resumed in the foreground or background. It's often nice to be able to suspend a job, go off and do something else, then come back and resume what you were doing. Alternatively, to stop what you're doing and run it in the background

```
[1:53pm julian] bg
[1]    du >& /tmp/usage &
[1:53pm julian] ... job runs in background
```

Use the command **fg** to bring a suspended job to the foreground.

To resume a stopped job in the foreground

```
[1:13pm julian] mail purple
Subject: just a test
I'm going to suspend and resume this job
^Z
Suspended
[1:31pm julian] fg
mail purple   ... the command resumed
(continue)
~q
[1:31pm julian]
```

Vax/VMS users often type **^Z** (as end of file) when they mean to type **^D.**

BEWARE:
Don't make this mistake; or, if you do, remember how to control the jobs you suspend!

8.5.3. Control Commands

Since you can have many jobs running in the background (some stopped waiting for keyboard input) you need tools to control them.

The command **jobs** lists your current jobs (with the **-l** flag you also get the process id).

```
[1:50pm julian] jobs
[1]   Running             du >usage
[2]  - Stopped (tty input)  emacs
[3]  + Stopped (tty input)  mail
```

Jobs are referenced with a **%** notation — **%n** (for job number n), and **%command** (for the job running that command). When no job is mentioned, the current job is assumed.

8.5.4. Control builtins

The Csh builtins for job control are

stop
to stop (or suspend) a job. A stopped job can be restarted. All background jobs will stop when they need terminal input.

kill
to kill a job. This sends the SIGKILL signal.

bg
to put a job in the background. eg. after a ˆZ.

fg
to bring a job to the foreground. eg. when keyboard input is requred.

For example:

```
[1:51pm julian] stop
[1:51pm julian] kill %1
[1:51pm julian] fg %1
[1:51pm julian] bg %du
```

8.5.5. Tenex Csh

The Tenex version of the Csh has a number of important extensions to the Csh. It's distributed as a patch set for BSD Csh.

Some important features are:

emacs line editting
With **c-p** (previous), **c-n** (next), **c-b** (back), **c-f** (forward) you can recall previous commands, make corrections and execute them.

speller
With **m-s** you can have typos corrected, eg.

```
[2:06pm julian] ls /user/spool/lpd
/user/spool/lpd not found
[2:06pm julian] c-pls /user/spool/lpdm-s
[2:06pm julian] ls /usr/spool/lpd
```

To change your default shell to **/bin/tcsh**

```
[2:09pm julian] chsh reggers
Old shell: /bin/csh
New shell: /bin/tcsh
```

This will be effective the next time you login.