CCLRC / Rutherford Appleton Laboratory **SUN/145.6**
Particle Physics & Astronomy Research Council
Starlink Project
Starlink User Note 145.6

M D Lawden, G R Mellor, B Smalley
12 July 1996

UNIX
An Introduction

Abstract

This is a brief introduction to the Unix operating system as it is used in the Starlink Project. It is aimed at users who are new to Unix and to Starlink. Its purpose is to get you started quickly, so it is simple rather than comprehensive.

It should be read in conjunction with any local guides provided at your local Starlink Site – these give details of the local facilities available and how to use them. This is important because Starlink Sites differ in their computer hardware and in their versions of Unix.

A "Quick Reference Card" is available which summarises the contents of this note.

Contents

1 Introduction

This note will get you started on a Starlink Unix computer.

It is not comprehensive. It tries to be simple. It is Starlink-specific. It is a survival kit to help you through the initial learning phase, during which you are suffering from information overload. To ease this overload we omit lots of options and alternatives. There is a lot more to Unix than described here, but you don't need to know it in order to do useful work.

In addition to reading this guide, have a chat with your Site Manager about running Unix at your Site; he will know the local setup and will show you things. Find a colleague who already knows Unix and give him the pleasure of showing off his knowledge by asking some callow questions about it. Get hold of a book on Unix; there are lots of good ones around and they take time to explain things that are skimped or omitted here.

2 Getting started

This section introduces you to the basic skills you need in order to do anything at all with a Unix computer. It mentions things that are covered more systematically, or in greater depth, later on. Try the examples yourself.

The first thing you need is an account on a Unix machine – see your Site Manager. Then, you need to know how to login to the machine from your keyboard. Having logged in, you can try out some commands to see what files you've got and what's inside them, see who else is logged in, and try out the help system. You can also check to see that you are set up to run Starlink software. This section guides you through all this.

2.1 Logging in

Your Site Manager should show you how to login to a Unix machine. You've just got to know the local details (like computer names, available equipment and networks), which we can't go into here.

You login by entering your username and password. Unix is case sensitive, so that upper and lowercase characters are regarded as different. Having logged in, you will probably see a prompt '`%`' (or something similar) generated by the C shell. You can now enter commands which will be interpreted by this shell.

2.2 Changing your password

The first time you login, the first thing you ***must*** do is change your password. The command used to do this depends on how your Starlink site is configured. Your Site Manager will tell you the appropriate method to use at your site.

Passwords are not echoed to the screen when you type them. They should be seven or eight letters long and be non-dictionary words and definitely *not* names.

A good way to produce a secure password is to mix upper or lowercase letters, numbers, and punctuation. Do not just make obvious substitutions (*e.g.* "$" for "s", "1" for "i") or combine a word with a digit as these are easily crackable.

Please take password security seriously – our machines are on the Internet and are exposed to hackers worldwide with sophisticated password cracking programs.

Do not tell anyone else your password.

2.3 Logging out

To logout, type:

```
% logout
```

If the system responds with "`There are suspended jobs`", then type it again – those suspended jobs will be abandoned.

2.4 Controlling your terminal

Here are some keys to control your input, output, and programs:

DEL	Rubout the last character typed.
ctrl/S	Stop output appearing on your screen (you won't lose anything).
ctrl/Q	Start output appearing on your screen again (use after ctrl/S).
ctrl/U	Cancel and clear the command line.
ctrl/C	Terminate the running command (it cannot be restarted). You can then type in another command.
ctrl/D	'End of File'. Used to terminate data or text input from the keyboard. *If you use it when being prompted for a command, it will log you out!*

There are plenty of other control keys, but the ones above enable you to deal with most emergencies.

2.5 Looking around

If you logged out, log back in again. Now use some commands to look around. The usual form of a Unix command is:

```
command [-options] [arguments]
```

The brackets '`[ ]`' indicate that a field is optional. For example, if you want to know what the date and time are, enter the command:

```
% date
```

This command has been entered (in response to the prompt '`%`') with no options or arguments. Other commands that don't need options or arguments include:

```
% cal
```

which lists a calendar for the current month,

```
% who
```

which lists the usernames of the people who are logged into the machine you are using (a similar command is `finger`, which is nicer because it tells you their names), and

```
% pwd
```

which tells you the name of your current working directory. (If you have just logged in, this will be your 'home' directory.)

Although a command may not *need* an argument, you can usually use one to control its behaviour more exactly. Thus, the `cal` command will show a calendar of the current month, by default. However, you can get a whole year's calendar if you specify a year:

```
% cal 1996
```

or, if you just want to know when Christmas is, you can specify a month and year:

```
% cal 12 1996
```

Arguments specify what a command works on. *Options* specify more exactly what sort of work a command does. For example,

```
%  ls
```

lists the names of the files in your current directory in a compact form. However, you may specify *options*, as in:

```
%  ls -la
```

The options are introduced by the '-' character. Each character that follows specifies a command option. In this case 'l' specifies a more detailed type of listing, and 'a' specifies that a type of file that is normally hidden should be listed.

The `ls` command is also often used with *arguments*, for example:

```
%  ls /star
```

lists the names of the files in directory `/star` in compact form. The shell knows that `/star` is an argument and not an option because it doesn't have that '-' sign in front of it.

Finally, you can combine options and arguments, as in:

```
% ls -l /star/docs
```

which lists full details of the files stored in directory `/star/docs`. You probably found that the first part of the listing shot off the top of the screen. One way to control this is to add '`| more`' onto the end of the command, thus:

```
%  ls -l /star/docs | more
```

What this does is route the output of the `ls -l /star/docs` command directly into the input of the `more` command. The `more` command displays its input one screenful at a time. To get the next screen, press the `<space>` key (*not* the `return` key). To quit the listing, type '`q`'.

The '`|`' character is called a 'Pipe' and is one of the most powerful features of Unix. It routes the output of a command into the input of the next command and lets you string commands together. For example:

```
% ls -s /star/bin | sort -nr | head
```

will list the ten largest files in directory `/star/bin`. The command `ls -s /star/bin` generates a list of the names and sizes of the files in directory `/star/bin`. The `sort -nr` command sorts this list into numerical order, based on the size field. Finally, the `head` command displays the first ten lines of the output of the `sort` command.

2.6 Setting up your environment

When you login, the system reads commands from two files (called *startup scripts*) in order to set up an appropriate environment. The files are called `.login` and `.cshrc` and should be stored in your home directory (the directory you start in). Initial versions will probably have been set up by your Site Manager. However, you should make sure they contain the commands which let you use Starlink software. These are:

```
source /star/etc/login
```

in your `.login` file and

```
source /star/etc/cshrc
```

in your `.cshrc` file. You can see what these commands are by:

```
% cat /star/etc/login /star/etc/cshrc | more
```

The `cat` command lists the contents of a one or more files. More information about startup scripts is given in sections 4.7 and 4.8.

2.7 Getting help

Unix has a 'Manual' stored on-line which includes command descriptions. You can look at these with the `man` command. For example, to find out about the `ls` command, type:

```
% man ls
```

Instead of a command name, you can specify a keyword. For example, to find commands which might have something to do with printing, type:

```
% man -k print
```

If you want to know what a command does, type:

```
% whatis cat
```

for example. This will tell you what `cat` does.

2.8 Finding information

Starlink stores lots of information for reference, but how do you find what you need? Section 6 of the Starlink User's Guide (SUG) tells you how this information is organised and how to search it. Here, we get you started by describing some of the most useful tools.

To find out what is going on within Starlink, type:

```
% news
```

Then type the title of one of the items in the list (enough to make it unique will do). For example, if one of the titles is 'JOBS', then reply to the prompt by typing:

```
...: JOBS
```

To get back to the shell prompt, type "q" one or more times (SUN/195).

If you want to find information related to a specific topic you can use a command called `findme` (SUN/188). For example, if you wanted to find information on programs which did Fourier analysis you could type:

```
% findme Fourier
```

This will start up a "Mosaic" window which will have links to those sections of Starlink documentation which deal with Fourier analysis. To follow these links, just use your mouse to click on them.

If you know the code of a document you want to read (SUN/1 for example) you can examine it on-line by using the `showme` command (SUN/188):

```
% showme sun1
```

You can use any document code that refers to an issued document. If the document exists in hypertext form, this is a good way to read it. However, if you just get a picture of the document pages you may find it easier to use a paper copy.

Finally, the World Wide Web ("web") is a very powerful way to search for and read information. The most popular browser to use for "surfing" the web is "Netscape." To start this up, just type:

```
% netscape &
```

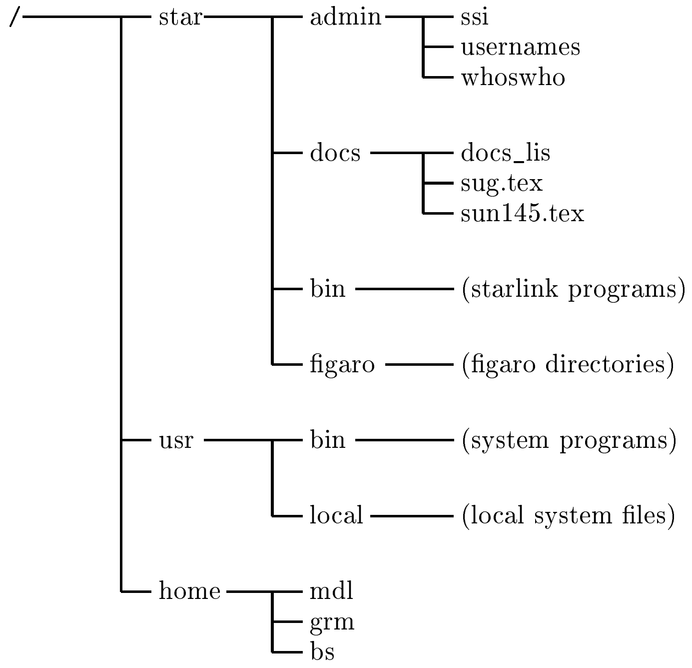
If any of the commands shown above don't work, seek help from your Site Manager.

2.9 Review

Here is a list of the commands used so far:

logout	logs you out.
date	reports the system time and date.
cal	displays a calendar.
who	reports who is currently logged into the system (terse).
finger	reports who is currently logged into the system (gives names).
pwd	gives the name of your current directory.
ls	lists the contents of a directory.
sort	sorts and collates lines.
head	displays the first few lines of a file.
cat	displays the contents of one or more files.
more	displays input to the screen, one page at a time.
man	gives details of a command.
whatis	gives a one-line description of a command.
news	lists recent news items (Starlink).
findme	searches Starlink documents for a topic (Starlink).
showme	displays a Starlink document (Starlink).
netscape	browses the World Wide Web.

3 Files and directories



Files are stored in directories. The Unix directory structure is a tree structure, branching into multiple levels of subdirectories. There are no disk names – extra disks get mounted as subdirectories; you just see one directory structure for the whole system. Some sites implement a disk quota system, others do not. Check with your Site Manager.

An example of a Unix directory structure is shown above. The top of the structure is called `/` (pronounced "root"). In the diagram, this top level directory is shown containing three directories called `star, usr, home`. These are just for illustration – in practice, all the directories in the diagram would contain many more files than listed.

The `star` directory contains the Starlink software, and its substructure is shown in detail. Major packages are stored in their own subdirectories, for example the `figaro` subdirectory holds the FIGARO software. The `admin` subdirectory holds administrative files such as `ssi` (the software index) and `usernames` (a list of Starlink users). The `docs` subdirectory holds files such as `sun145.tex` which contain the text of Starlink documents.

3.1 Naming things

Files and directories are located by specifying a 'pathname' – the name describes the path through the directory structure which ends up at the place you want. Pathnames can be 'absolute' or 'relative.'

- A *relative* pathname is relative to your current directory.
- An *absolute* pathname is the path from the root directory `/` (absolute pathnames begin with `/`).

As an example, in the directory structure shown above there is a file called `ssi`. Its absolute pathname is `/star/admin/ssi`. However, if your current directory is `/star`, its relative pathname would be `admin/ssi`. The `/` character is used as the name of the top level directory, but it is also used to separate components of the pathname.

Three useful shorthand names are:

.	current directory (the directory you are in at the moment).
..	parent directory (the directory which contains your current directory).
~	'home' directory (the directory you are in immediately after login).

3.2 Moving around

The following command lets you move around the directory structure:

cd change working directory.

Examples

`cd test`

move down one level to subdirectory `test` (relative pathname).

`cd test/lower`

move down two levels to subdirectory `test/lower` (relative pathname).

`cd /star/docs`

move to this directory (absolute pathname).

`cd`

move back to your home directory (equivalent to **cd** ∼).

`cd ~user`

move to the home directory of 'user'.

`cd ..`

move up one directory level.

3.3 Looking after directories

The following commands enable you to look after your directories:

pwd	print working directory.
ls	list the names of the files in a directory.
mkdir	create a new directory.
rmdir	delete a directory.
du	summarize disk usage.
df	report on disk partition details.

Examples

`pwd`

display the path name of your current directory.

`ls -a`

list the names of all files in a directory (including hidden '.' files).

`ls -F`

list file names with a character appended to each name indicating the file type:
/ indicates a directory.
* indicates an executable file.
@ indicates a soft link (points to another file).

`ls -R`

list subdirectories recursively.

`ls -l`

give full details of files.

`ls -lt`

give full details of files, sorted by time of last modification with the most recent listed first.

`ls -ld`

give details of the directory file only, rather than its contents.

`mkdir mydir`

create a directory called `mydir` below the current location.

`mkdir /home/grm/mydir`

create a directory called `mydir` in the specified place.

`rmdir mydir`

delete directory `mydir`. Directories will only be deleted if they are empty. If you cannot delete a directory that appears empty, look for hidden files with `ls -a`.

`du -sk`

if you issue this command from your login directory, it will tell you the grand total size (in Kbyte) of all your directories and subdirectories. If you omit the "s", the size of each individual directory will be shown in addition to the grand total.

`du -sk /star/docs`

show the grand total size (in Kbyte) of the Starlink documentation directory and its subdirectories.

`du -sk * | sort -nr | head`

show the grand total size (in Kbyte) used by the ten largest files and directories in your current directory.

`df -k`

show the disks available on your system, together with the space used and the space available.

3.4 Looking after files

The following commands enable you to look after your files:

cp	copy files.
mv	move (*i.e.* rename) files.
rm	delete files.

Examples

`cp geoff.1 geoff.2`

copy file `geoff.1` to file `geoff.2` in the same directory, creating `geoff.2` if necessary.

`cp geoff.1 /home/md`

copy file `geoff.1` to directory `/home/md`, giving it the same name as before.

`cp geoff.1 /home/md/geoff.2`

copy file `geoff.1` to directory `/home/md`, giving it the name `geoff.2`.

`cp f* /home/grm`

copy all files beginning with the letter 'f' from current directory to directory `/home/grm`.

`cp /star/help/jcmtdr/html/* .`
: copy every file in directory `/star/help/jcmtdr/html` into your current directory.

`cp -i geoff.1 /home/grm`
: `cp` usually overwrites existing files. The `-i` option will prompt for confirmation if the copy would overwrite an existing file and will proceed only if reply `y` is given.

`cp -r /home/grm /home/md`
: copy recursively a directory and its contents to the specified location.

`mv geoff.1 geoff.2`
: rename the file `geoff.1` to `geoff.2`.

`mv geoff.1 /home/md`
: move file `geoff.1` to a different directory. If a new filename is specified, the file name will change.

`mv -i f* /home/grm`
: prompt for confirmation if a move would overwrite an existing file.

`rm geoff.1`
: delete file `geoff.1`.

`rm *`
: delete all files in the current directory (*without prompting!*).

`rm -i *`
: prompt first before deleting the files.

`rm -r /home/grm`

delete all files and directories below the specified directory, and also delete the named directory itself.

`rm -ir`

the recursive option is probably safer when used with the interactive option.

3.5 Looking at files

The following commands show what is in a file:

cat	show the contents of one or more files.
more	show the contents of a file, one screen at a time.
head	show the first few lines of a file.
tail	show the last few lines of a file.
diff	show file differences.
sort	sort or merge files.

Text files can be created by using the command `cat` (see below), or by using a text editor.

Examples

`cat > note`

create a file named `note` and type text directly into it. Terminate the input with a `return` key, followed by ctrl/D.

`cat note1`

show the contents of file `note1`.

`cat dic1 dic2`
concatenate the contents of `dic1` and `dic2` and show the result.

`more /star/docs/sun.tex`
show the contents of `sun.tex`. Initially, the first screenful of the file will be shown. What happens next depends on what keys you press:
`return` — next line
`d` — next half screenful
`u` — last half screenful
`space` — next screenful
`b` — last screenful
/pattern — search for 'pattern'
`n` — next occurrence of 'pattern'
`q` — terminate command

`head /star/docs/sun.tex`
show the first 10 lines of `sun.tex`. Option `-n` shows the first `n` lines.

`tail /star/docs/sun.tex`
show the last 10 lines of `sun.tex`. Option `-n` shows the last `n` lines.

`diff file.v1 file.v2`
show the lines which differ in these two versions of a text file.

`sort file1 > file2`
sort the lines in `file1` and store the sorted output in `file2`. By default, `sort` sorts a file using the beginning of the line as the key field. However, you can use any field in the file as the key field.

3.6 Finding out about files

The following commands give information about files:

find	find files.
file	show file type.
wc	show line, word, and character counts.

Examples

`find /star/* -name unixnames -print`

search the directory tree headed by `/star` for files with the name `unixnames`. If any are found, write their full pathnames to the standard output stream. (*Warning:* this search may take a long time, so don't be alarmed if nothing seems to happen for a while.) This command is useful for searching a directory tree for a file whose name you know, but whose location you don't. This is a powerful command which can search many directory trees for filenames which satisfy many complex criteria.

`file /star/docs/*`

show the type of each file in `/star/docs`.

`wc /star/docs/sun1.tex`

show the number of lines/words/characters in `sun1.tex`.

3.7 Finding things in files

The following commands find things in files:

grep find a string in a file.
awk find a string in a file and perform an action.

These are both powerful and complex commands, and we can't describe them fully here. The examples below illustrate the sort of things they can do, but you should look them up in a Unix book if you want to understand and use them extensively. The precise syntax varies on different machines.

The strings can be in the form of *regular expressions*. These can be used in commands such as `grep, fgrep, egrep, sed`. Here is a summary of the meaning of the special symbols that can be used in regular expressions:

`^`	beginning of line
`$`	end of line
`.`	any single character
`[...]`	single character in list or range
`[^...]`	character not in list or range
`*`	zero or more of preceding character or pattern
`.*`	zero or more of any character
`\`	escapes special meaning

Examples

`grep "pattern" filename`

search `filename` for the character string `pattern` and display the lines that contain it. The character string can be any regular expression (see above).

`grep Lawden /star/admin/whoswho`

search `whoswho` for information on 'Lawden'.

`grep -l IUE /star/docs/* | more`

find which Starlink documents mention IUE. The `-l` option restricts the output to just the file names.

`grep -i den /star/admin/usernames`

list the records of all users in `usernames` which contain the character string '`den`'. The `-i` option makes the search case-insensitive, so that it would also list 'Den', 'dNe', *etc.*

`who | awk '{print $1}' | sort | uniq -c`

show in alphabetical order who is logged in, and how many sessions they've got.

`ls -l | awk '$5 > 10000 {print}'`

show which files in your current directory are bigger than 10000 bytes.

3.8 Controlling access to files

The following command controls file access:

chmod change file permissions.

File protection is implemented by *file ownership* and *permissions*. The creator of a file is the owner, and the file is created with default permissions. The owner can then alter these permissions to allow the required degree of access to the files.

Permissions are split into three categories:

- `user` (u) the file owner.

- `group` (g) members of the user's group.
- `others` (o) everyone else.

There are three types of permission, but their effect varies slightly between directories and files:

Normal files:

READ (r)	can look at the contents.
WRITE (w)	can modify the contents or delete.
EXECUTE (x)	can execute.

Directories:

READ (r)	can list contents.
WRITE (w)	can modify contents.
EXECUTE (x)	can move through (use in a pathname).

File permissions can be seen by typing an `ls -l` command. This displays information similar to that shown below:

```
drwxr-xr-x   2   grm   users    512   Nov   18   04:10   mydir
-rwxr-xr-x   1   grm   users   1640   Nov   21   18:15   workfile
```

The first field is the permission field. Its first character shows the file type: the most common are normal files (`-`) and directory files (`d`). The next three characters describe the 'user' permissions, followed by three for 'group' and three for 'others' permissions, making a ten-character field in all. The permissions can be described in octal format instead of by letters. Those for each category are concatenated, giving three digits to describe the file. For example:

`-rw-r--r--` is equivalent to `644`.

`-rwxr-xr-x` is equivalent to `755`.

The owner can change the permissions of a file with the `chmod` command. Either octal or character notation can be used.

Examples

`chmod 700 geoff.1`

set full access to user, none to group or others.

`chmod 740 g*`

set full access to user, read access to group, none to others, for all files beginning with g.

`chmod og=rx g*`

set others and group to read and execute access.

`chmod og+r g*`

add read access to others and group.

`chmod og-w g*`

remove write access from others and group. Generally, you *should not* allow write access to group or others.

`chmod a=rx g*`

set all (*i.e.* user, group, and others) to read and execute access.

`chmod -R 755 mydir`

recursively set access to files below `mydir` directory.

`chmod 755 script`

a script (a file of shell commands) must have execute permission to run; you can set it like this.

3.9 Backing up files

Files can be backed up with the command:

tar create and use tape archives.

You can backup your files to a tape or disk file with the `tar` command. The resulting file is called a tarfile. The basic syntax is:

```
% tar cvf <destination> <source>
```

(The omission of the `-` character before the options `cvf` is deliberate; for this command, all the options are stored in the first argument.) This recursively backs up all the files in the `<source>` directory structure into a `<destination>` tarfile or tape drive. The `c` option specifies that a new tarfile is to be created; the `v` option specifies verbose mode (each file's name is logged to your terminal); the `f` option indicates that the next field (*i.e.* `<destination>`) specifies the name of the tarfile.

A tarfile can be *relative* or *absolute*, depending on how the source file name is specified. The use of relative archives is recommended, since they can be restored in any position within a directory structure.

The use of tape drives is site dependent, so ask your Site Manager how to use them at your site.

Examples

`% tar cvf /dev/rmt0h /home/grm`

produce an absolute tarfile (because `/home/grm` is an absolute file name). When this absolute tarfile is unpacked, the files are restored to directory `/home/grm`, regardless of the current working directory. The destination `/dev/rmt0h` specifies that the tarfile will be written to tape drive `rmt0h`. The tape device name can vary depending on your operating system and other factors such as the tape density required. Your Site Manager will advise on the correct ones for your site. Please note that the default action of a Unix tape drive is for the tape to be rewound after the required operation. If you wish the drive to remain in position after an operation (*e.g.* in order to write a second tarfile) then you must use the "norewind" version of the tape drive name (in the above example, that would be `/dev/nrmt0h`).

`% tar cvf /home/grm/tarfile.tar *`

produce a relative tarfile (because the `<source>` – the files in the current directory – is specified as a relative file name). It is conventional to use the extension `.tar` in a tarfile name. When a relative tarfile is unpacked, it will recreate the stored file structure relative to the current directory.

`% tar tvf /home/grm/tarfile.tar`

list the contents of a tarfile (this is specified by the `t` option). The example shown will produce a detailed listing because it specifies the `v` (verbose) option. If you omit the `v` you will just get a list of file names, which is usually all you need.

`% tar xvf /home/grm/tarfile.tar`

restore the contents of a tarfile, using the `x` option. Many other options are available; see the `man` pages for details.

3.10 Tape drive names

In the previous section we sometimes specified a tape drive in the second argument of the `tar` command. An example of this usage is:

```
% tar cvf /dev/nrmt0h /home/grm
```

On Dec Unix systems, tape devices have names like `/dev/nrmt0h` where "`/dev`" gives the location, "`nr`" means "no-rewind" after the write operation is complete, "`mt0`" is the device name, and "`h`" indicates high-density or compressed mode.

On Solaris 2 systems, tape devices have names like `/dev/rmt/1n` where "`/dev`" is the location, "`rmt`" is the tape driver type, and "`1n`" indicates the specific device *and* that it is a no-rewind type. Another example is `dev/rmt/0`, indicating "rewind" device "0".

4 Shells – The command languages

'Shell' is the Unix term for 'command language interpreter.' Several different shells are available:

- C shell (csh).
- T-C shell (tcsh).
- Bourne shell (sh).
- Korn shell (ksh).

Starlink's software can be run from the C shell or T-C shell, but not from the Bourne or Korn shells. Most sites use the T-C shell, which is a variant of the C shell with file name completion and command line editing.

The following sections describe features of the C and T-C shells.

4.1 Standard files

A program works with three special 'files' which are normally connected to your terminal's screen and keyboard. Normally, it receives input from `stdin` (the keyboard), sends output to `stdout` (the screen), and error messages to `stderr` (also the screen). However, this can be redirected by special operators:

>	directs `stdout` to a file. *e.g.* `date > date.file` writes the date/time to file `date.file`.
>&	directs `stdout` and `stderr` to a file.
>>	appends `stdout` to an existing file.
<	directs the standard input `stdin` to read its input from a file. *e.g.* `cat > newfile < date.file` writes the contents of `date.file` to `newfile`.

4.2 Pipes

A Pipe is an important and useful feature of Unix:

| directs the `stdout` of one command into the `stdin` of another, without creating a temporary intermediate file. *e.g.* `ls -l | more` shows a directory listing, page by page.

4.3 Filename expansion

You can specify multiple file names in a single expression by including certain characters in the name you type:

***** represents one or more characters. *e.g.* `A*` represents any name beginning with `A`.

? represents a single character. *e.g.* `CA?` represents any three character name beginning with `CA`; thus `CAD` and `CAT` would be recognised.

[a-z] a string of characters enclosed in brackets is known as a 'character class'. It means *match any single character which appears within the brackets.* You can specify individual characters like `[aft]`, or sequences of characters like `[a-c]`. *e.g.* `CA[DT]` represents the two names `CAD` and `CAT`.

~ represents your home directory. *e.g.* `cd ~/subdir` will change directory to `subdir` below your home directory.

4.4 History

The following command accesses previous commands:

history display previous commands.

The history file allows commands entered earlier in your session to be recalled for reuse or modification. With the T-C shell you can recall previous commands by pressing the up-arrow key.

To activate the history mechanism, type:

```
% set history = <n>
```

where `<n>` is the number of previous commands to store. This command is best included in your `.cshrc` file (see section 4.7) as this will cause the history mechanism to be activated automatically for every shell you start. Once activated, commands like the following can be used.

Examples

`history`

displays a list of previously entered commands.

`!p`

working back from the most recent command in the list, execute the last command beginning with `p`.

`^t^d`

replace `t` by `d` in the last command and re-execute. `t` and `d` can be character strings of any length.

`!l:p`

recall the last command beginning with `l`, but do not execute. This allows a command entered earlier to made the most recent one. It can then be edited using the method in the previous example.

`!!`

re-execute the last command.

4.5 Variables

The following commands are concerned with variables:

echo	show value of a variable.
set	define or display value of a shell variable.
unset	remove definition of a shell variable.
setenv	define value of an environment variable.
printenv	show values of environment variables.

Two types of variable are used by the C shell:

- Shell variables
- Environment variables

Their main features are shown below:

	Shell variables	**Environment variables**
Scope	Current shell	Global
Naming	lower case	upper case
Type	strings, numbers, arrays	strings (only)
Reference	`$var`	`$VAR`
Set value	`set var = value`	`setenv VAR value`
Show value	`echo $var`	`echo $VAR`
Show all	`set`	`printenv`

Variables control the way the shell operates, and certain ones are set by default. Here is a list of shell variables that are frequently defined:

`cwd`	your current working directory.
`home`	your home directory.
`path`	list of directories searched for a command.
`prompt`	the shell prompt.
`shell`	the default shell.
`status`	reports whether a command succeeded.
`term`	terminal type.
`user`	username used to login.

Many other shell variables can be used; the manual page for `csh` describes them. Some don't need values but can be toggled on or off by `set` or `unset`.

Examples

`set filec`

enables filename completion. Type in an unambiguous part of a file specification, press `<Escape>`, and the shell will complete the filename.

`set noclobber`

restricts output redirection so that existing files are not destroyed by accident.
> redirections can only be made to new files.
>> redirections can only be made to existing files.

`unset noclobber`

removes the definition of the `noclobber` variable and cancels its effect.

Here is an example of a shell variable being defined and then used:

```
% set mydocs = ~/docs
% cd $mydocs
```

Some variable names, such as `path` and `term`, are unusual in that they refer to both shell and environment variables which are identical. Thus, you can add directory `/usr/local/bin` to the `path` by treating it as a shell variable:

```
% set path=($path /usr/local/bin)
```

or as an environment variable:

```
% setenv PATH $PATH:/usr/local/bin
```

One difference is that when `path` is used as a shell variable, its value is an array whose components are separated by blank characters, but when it is used as an environment variable, the components are separated by `:` characters. You can see this by trying `echo $path`, followed by `echo $PATH`.

Shells use hashing techniques to speed-up the search for commands. Consequently, if you change your `path` or add a command to one of the directories, you must execute the `hash` command to force the shell to rebuild its hash table.

Remember, shell variables are only valid in the shell in which they are set. They are not passed down to any subsequent shells that are started. However, you can make them available automatically by putting their definitions in your `.cshrc` file (see section 4.7).

4.6 Aliases

The following commands control the renaming of commands:

alias	give another name to a command.
unalias	cancel an alias definition.

The `alias` command saves you time by letting you give shorthand names to long command strings that you use frequently.

Examples

`alias h history`

shortens the command `history` to the single letter `h`.

`alias ll 'pwd;ls -l'`

allows a sequence of commands to be executed as a single command.

`alias`

on its own, shows all the aliases currently defined within the shell.

`unalias ll`

removes the alias `ll`.

4.7 Startup scripts

As already mentioned (section 2.6), two files containing commands are executed automatically when you login:

.cshrc	C shell startup script.
.login	login startup script.

They must reside in your home directory. The difference between them is that `.cshrc` is executed whenever a C shell (or T-C shell) is started, but `.login` is only executed at login time. Thus, `.login` is used to set up a global environment which affects your entire terminal session. For example, it can define environment variables such as `TERM` and `PATH`, set up terminal characteristics using `stty`, and start up X-windows. On the other hand, `.cshrc` is used to define shell variables and aliases (which are defined only for a specific shell).

Special definitions are required in order to run Starlink software successfully. These are stored in the files `/star/etc/login` and `/star/etc/cshrc`. These contain definitions that should be added to those in your `.login` and `.cshrc` files. This is done by reading these extra files from within the original files; the Unix term for this process is 'sourced'. Thus, your `.login` file should contain the command:

```
source /star/etc/login
```

and your `.cshrc` file should contain the command:

```
source /star/etc/cshrc
```

These commands should appear at the end of the files, so that anything else you do in them does not override Starlink definitions.

4.8 Shell scripts

Instead of typing in repetitive sequences of commands, you can store them in a file and execute them by typing the name of the file. A text file that is interpreted as a set of commands by a shell is called a *shell script*. These are used to:

- automate a sequence of commands that is used regularly.

- do complex sequences that you might forget to do in the correct order.

The shells also have built-in functions to provide flow control, argument handling *etc.*, which allow rudimentary programming tasks to be performed.

If you type a command that is the name of a text file, the shell assumes this file contains a shell script and creates a new process running a new shell. This new shell reads the standard startup file for that shell (`.cshrc` for the C shell) and executes the commands in your text file.

The shell that executes a shell script might not be the same type as the one into which you are typing commands. The exact rules for which shell is used are rather complex, but briefly, if the first line of a shell script is:

- a blank line or a normal command, the Bourne shell is used.
- a comment, the C shell is used, except that:
- if the comment is `#!/bin/sh`, the Bourne shell is used;
- if the comment is `#!/bin/csh`, the C shell is used.
  (This mechanism can be generalised for other shells.)

A common problem for users of the C shell or T-C shell is to write a quick shell script and wonder why it fails completely. Often, this is because there are no comments at the top of the script (yes, we all do it), so the script is run by the Bourne shell. Many Bourne shell commands are different from the corresponding C shell commands.

The Bourne shell is generally preferred for shell programming as its scripts run faster. However, the syntax of the C shell is similar to that of the C language and is much easier to comprehend. For that reason, many users write their scripts for the C shell.

The programming features of the shells are rather limited. If you require more advanced functionality, then look at Perl. This is a programming language which allows easy manipulation of text, files, and processes. It is installed at all Starlink sites and your Site Manager can provide information about it.

4.9 What happens when you type a command

When you enter a command, the shell you are talking to examines what you have typed and works out what to do. What it does is quite complicated and there isn't space to explain it in detail here; we will just deal with some of the basic characteristics.

One of the first things the shell does is see if the command (the first word you typed) matches an alias (see section 4.6). If it does, the shell replaces the alias with the equivalent command.

There are now two possibilities:

If the command is a shell command (one that it understands such as `setenv`), the shell processes the command itself.

Otherwise, it takes the command to be the name of a program to run. It must now find the program. It does this by examining the environment variable PATH. It looks in turn in each of the directories specified in PATH for a file that matches the name of the command. Once found, there are again two possibilities. If the file is executable (*e.g.* the result of compiling and linking a Fortran program), the shell creates a new process that is an almost identical copy of the current one (this is known as forking a process) and executes the program in that new process. If the file contains ASCII text (known as a shell script), the shell again forks a new process, but this time the program that is run is a shell. If it is a C shell, it will read the `.cshrc` file in your home directory. The new shell then reads the contents of the text file and executes each command, using the above rules.

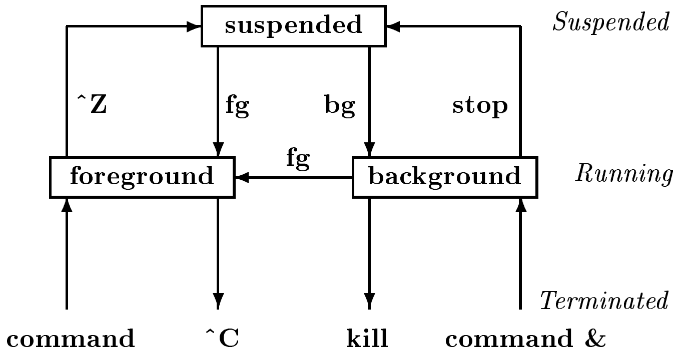
5 Controlling processes

The following commands are concerned with controlling processes:

ps	list current processes.
jobs	list current jobs.
at	run a job later.
fg	run job in foreground.
bg	run job in background.
stop	suspend background job.
kill	terminate background job.
nice	run a command at a different priority.
renice	alter the priority of a running process.

In Unix, you can have several things going on at once. You may be compiling a large package, editing a couple of documents, monitoring your mail messages, keeping your eye on the clock, and watching the load on your computer. You keep switching your attention between these various tasks. In computing terms, this means that there are several processes competing for system resources.

Normally, when you enter a command, you have to wait until the job or process that carries out this command has finished before you can enter another command. In this case, the job runs in the 'foreground'. However, if you end the command with an '`&`', the job runs in the 'background'; which means that you can type another command while the previous one is being processed. You can also stop a job from running. This can be a temporary suspension (*i.e.* you can start the job again), or you can 'kill' the job so that it ceases to exist and cannot be restarted. The diagram below illustrates these various states, and the transitions between them.



Processes are identified in three ways:

PID — Process identifier.

%N — Job number N.

% — Current job.

Thus, if you want to terminate a background job, you can do it with any of the commands:

```
% kill PID
% kill %N
% kill %
```

You can find out the process identifiers of your existing processes with the `ps` command. For example:

```
% ps
    PID TTY      S           TIME COMMAND
   8585 ttyp8    S        0:03.02 -tcsh (tcsh)
   8612 ttyp0    R        0:00.10 ps
```

This example shows a situation in which two processes exist. The first (PID=8585) is running `tcsh` and is suspended. The second (PID=8612) is running the `ps` command and is in the foreground.

You can find out your current job numbers with the `jobs` command:

```
% jobs
[1] + Running    du >storage &
[2] - Running    cc -o foo foo.c >&errs &
[3] - Stopped    find / -size 0 -print
```

The job numbers are enclosed in brackets. The '+' indicates which job is the 'current job'.

A background job will halt when it needs input from your terminal. To proceed, bring it to the foreground with the `fg` command and type some input.

If you wish to execute a job at a later time (rather than immediately by using an `&`), you can use the `at` command. For example:

```
% at 23:00 Jul 07 < spectrum
```

will run the commands in file `spectrum` at 23:00 on 7th July.

```
% at -l
```

will list all your scheduled jobs.

You can control the priority at which commands execute with the `nice` and `renice` commands. Priorities for user processes range from 0 (default interactive priority) to +20 (slow). For example:

```
% nice +19 bigprog &
```

will run `bigprog` at priority +19 (a low priority).

You should always reduce the priority of CPU-intensive jobs in order to minimise their adverse effect on the response time of other interactive jobs. Note that the speed of execution of a process is only reduced when in competition with other processes of higher priority (*i.e.* when other users are logged into the same system).

Alternatively, if you have started a program and need to lower its priority to avoid complaints from other users, you can do something like:

```
% renice +19 886
```

which lowers the priority of process PID=886 to +19 (slow).

6 Editing text

The following commands invoke editors:

vi	standard Unix editor.
emacs	the powerful emacs editor.
jed	a text editor.
tpu	nu/TPU editor.

Editors are personal things, so it is impossible to make a recommendation that will satisfy everyone.

The standard Unix screen editor is `vi`. This is fast and powerful (deadly, in fact). You can find some documentation for it in MUD/122 and MUD/123.

`emacs` (SUN/34, MUD/102) and `jed` (SUN/168) are alternative editors.

A document describing `nu/TPU` has been distributed to every site – see your Site Manager. To use it, just type the command `tpu`, followed by the name of the file you want to edit (SUN/192).

Other editors may be available, check with your Site Manager.

Further information on editors is given in SUN/170. Editors can have a big impact on your work, so we strongly recommend that you read this carefully and think about the issues.

7 Producing documents and graphs

7.1 Documents

The following commands help you produce documents:

tex	TEX typesetter.
latex	LATEX document preparation.
xdvi	display document in an X window.
dvips	prepare PostScript output.
gs	GhostScript PostScript previewer.
lpr	print files.
ispell	spell checker.
a2ps	Ascii to PostScript file conversion.

We recommend you use LATEX to produce documents (SUN/9). We also recommend that you base your documents on standard Starlink styles, such as `/star/docs/sun.tex` (SGP/28).

To use TEX or LATEX to process file `filename.tex`, type:

```
% tex filename
```

or

```
% latex filename
```

You need not give the `.tex` file extension, but it won't matter if you do.

To view or print the resulting `filename.dvi` file there are several options. To view your document in an X-window, use the `xdvi` utility (SUN/9):

```
% xdvi filename
```

(If you get the error message '`Error:  Can't open display:`', type `xdisplay` and try again – if this doesn't work, consult your Site Manager. This isn't the place to discuss X-windows, but you can look at the description of `xdisplay` in SUN/129 for enlightenment.)

To make printable versions, use the `dvips` translator:

```
% dvips filename
```

This will create an output file `filename.ps`. This can be previewed using `ghostscript` (SUN/197):

```
% gs filename.ps
```

Then, print the resulting PostScript file by a command like:

```
% lpr -Pps filename.ps
```

The precise details of how to print files is locally dependant and cannot be described properly here. For example, you may need to use the `lp` command rather than the `lpr` command. You should, therefore, ask your Site Manager or consult your local guide.

You can check your spelling in a document by using the `ispell` command (SUN/189) to examine its file:

```
% ispell filename.tex
```

Another useful utility for handling documents is `a2ps` (SUN/184) which converts an ordinary ascii text file into PostScript form which can then be printed on an appropriate printer. This usually results in more pleasing output than you would get from using the raw text file. A typical use of this command is:

```
% a2ps -p -ns -nn -nh filename | lpr -P1
```

(You may need to change the `P1` to something else at your site.)

7.2 Graphs

The following command is needed if you use an X terminal:

xdisplay	connect workstation with X device.

If you are using X to run programs on a remote workstation, use command `xdisplay` to set up the connection between the workstation and your X device before running any graphics program (or any other X application). Reply `xwindows` if prompted for a graphics device name.

8 Mailing and Networking

8.1 Mail services

The following commands invoke mail services:

mail	standard mail utility.
pine	friendly mail utility.

The default `mail` utility is adequate but unfriendly. We recommend `pine`. It is designed to be easy for a novice to use. For example, it tolerates mistakes and its command menus are always present. It can be learnt by exploration rather than by reading manuals, however SUN/169 can be consulted if needed. To start it, just type `pine`.

8.2 Mail addresses

The following command helps you find mail addresses:

email	mail address search utility.

On Unix you send mail to addresses like:

```
cac@star.rl.ac.uk
```

Here, the username precedes the address, and they are separated by an `@`. This form of address is known as the 'Internet' or 'IP' form, and this should be used on Starlink's Unix machines.

The file `/star/admin/whoswho` contains the network addresses of Starlink sites. Network addresses for people are best found on Starlink by using the `email` utility (SUN/182). For example, to find out Geoff Mellor's network address, type:

```
% email mellor
```

8.3 Copying files across networks

The following commands copy files across networks:

cp	copy files (local).
ftp	file transfer program.

There are several ways to copy files from one machine to another. The best method may depend on how your local system is set up. Nevertheless, there are several standard methods that normally work:

Local machines (`cp`)

All the disks on your local group of machines will probably appear as part of your file system. If so, just use the `cp` command in the normal way:

```
% cp <source-path> <destination-path>
```

Other machines (`ftp`)

You need a username on the remote machine. Type:

```
% ftp remote_host
```

where `remote_host` is either the name of a host (for example `starlink-ftp.rl.ac.uk`), or its 'dotted quad' network number (like `130.246.36.1`). You will be prompted for your remote username and password, and will be logged in for file transfer work only. You can use directory changing and listing commands, such as `cd` and `ls`, to locate the files you want. To copy a file from a remote machine to your local one, type:

```
ftp> get remote_filename local_filename
```

To copy a file from your local machine to a remote one, type:

```
ftp> put local_filename remote_filename
```

You can `get` files from any remote host on which you have a username, and you can `put` files to any remote host on the same basis.

By default, `ftp` will transfer data in `ascii` mode – which is correct for text files. To transfer binary files (executables, tar files, NDF container files *etc.*), set `ftp` into `binary` mode. Otherwise, although the transfers will probably happen, the destination file will probably be useless. To get `ftp` into binary mode, type:

```
ftp> binary
```

To get help in an `ftp` session, type:

```
ftp> help
```

To disconnect from an `ftp` session, type:

```
ftp> quit
```

Some hosts have a special facility called 'anonymous ftp'. This has some disk space for a 'public' area, and is commonly used to allow informal distribution of common software utilities, pictures *etc.* To use it, start a normal `ftp` session to the host, but instead of using your username, login as `anonymous`. You will be prompted for your local username and node name in lieu of a password. Specify your full network address (for example `mdl@star.rl.ac.uk`) as the password so that the owners of the system can find out who is accessing it. You will gain access to a limited set of files, provided for the anonymous ftp facility, which you can copy to your local machine as described above.

8.4 Logging into another machine

The following command allows remote login:

telnet remote login.

To login to another machine *from your current session* use `telnet`. The command is:

```
% telnet remote_host
```

You will be prompted for a username and password, depending on what the `remote_host` requires for user identification. The form of the name you give as a `remote_host` will depend upon how your Site Manager has set up your systems. In some cases you will be able to type just the machine name. For more remote machines, *i.e.* those in different network 'domains', you may have to give the full network address. For example, to connect to the Starlink Database machine, users of RAL machines need only type:

```
% telnet stadat
```

whereas users of machines outside RAL would probably need to type:

```
% telnet stadat.rl.ac.uk
```

You can also use the 'dotted quad' network number of the machine:

```
% telnet 130.246.32.91
```

9 Programming

Skip this section if you only want to use application programs, not write them.

The following commands provide programming facilities:

f77	compile a Fortran 77 program.
cc	compile a C program.
star_dev	set up links required for developing Starlink programs.
fsplit	split a multi-routine Fortran file into individual files.
ftncheck	check Fortran 77 source code.
ar	insert subroutine in a library.
ranlib	update archive index.

In addition to the standard compilers, some sites have Fortran 90 and c++. Check with your Site Manager if you need to use these.

The compilers produce an object file (extension `.o`). The object code, and any other objects specified, are passed to the loader which produces an executable image. All this is done by one command.

To compile and link a Fortran program stored in file `prog1.f`, type:

```
% f77 prog1.f
```

This will produce an executable file called `a.out`. If you would rather name the executable file `prog1`, use the `-o` option:

```
% f77 -o prog1 prog1.f
```

If you only want to compile the program without linking, use the `-c` option:

```
% f77 -c prog1.f
```

This will create an object file called prog1.o.

You can compile a program that is stored in several files with a single command:

```
% f77 -o progall prog1.f sub1.f sub2.f
```

or in stages:

```
% f77 -c prog1.f
% f77 -c sub1.f
% f77 -c sub2.f
% f77 -o progall prog1.o sub1.o sub2.o
```

or with a mixture of the two, such as:

```
% f77 -c sub1.f
% f77 -c sub2.f
% f77 -o progall prog1.f sub1.o sub2.o
```

If you wish to link your program with a subroutine library, use the -L option to specify the name of the directory containing the library, and use the -l option to specify the name of the library itself. In general, to link with library libxxx.a, specify -lxxx in the compilation command. Thus, to link with library libnag.a, specify -lnag:

```
% f77 -o prog2 prog2.f -L/star/lib -lnag
```

To compile C programs, follow the above instructions, replacing the f77 command by the command (cc) that invokes the C compiler on your computer.

If you want to link with Starlink subroutine libraries, there are linking scripts provided to make the job easier (see SUN/202 for information about specific Starlink libraries). For example, to link a program that calls the AGI subroutine library, type:

```
% f77 -o agitest agiprog1.f -L/star/lib `agi_link`
```

If you are using Starlink subroutine libraries in your Fortran programs, you may need to have INCLUDE statements in your code. Use the file name given in the documentation, *e.g.*

```
INCLUDE 'SAE_PAR'
```

In order for the compiler to find this file, the name `SAE_PAR` must appear in the current directory. To make this happen, type the command:

```
% star_dev
```

before running the compiler. You only need do this once per directory. This sets up a link from the current directory to the real file. Each subroutine library has its own *package*`_dev` command file: `ems_dev`, `fio_dev`, *etc.*

In C programs, extra source files may be included with statements like:

```
#include "sae_par.h"
```

You must tell the C compiler where to find these files, using the `-I` option:

```
% cc -I/star/include cprog1.c
```

Note that the way you tell the compiler where to look for 'include' files is different in Fortran and C. Unfortunately, the Fortran compiler has no equivalent of the C compiler's `-I` option.

To build your own subroutine libraries and link to them, divide up your Fortran source code so that there is one subroutine or function in each file (use `fsplit`), and then compile each routine and insert it into a library (called 'archives' on Unix) as follows:

```
% f77 -c mysub.f
% ar r libmine.a mysub.o
```

When all the modules have been inserted, update the archive index with:

```
% ranlib libmine.a
```

(This isn't needed if you are using Solaris.) To link with the new library, use the `-L` option to specify the directories containing the archives, and the `-l` option to specify the name of the archive. For example:

```
% f77 myprog.f -L. -lmine
```

will cause the linker to look in the current directory for an archive called `libmine.a`.

A program often requires many files to be compiled and linked to produce an executable file; there may be source files, header files, and so on. The sequence of commands needed to control this process may be complex. Fortunately, in Unix there is a `make` facility which allows programmers to create a file, known as a `makefile`, which defines dependency relationships between files, and specifies the command sequences required to create a program. This facility is not described here, but if you are going to do serious programming you should find out about it. Starlink's software uses `make` files extensively in its software distribution and management.

10 Going further

This document is only an introduction. For more information, refer to the system documentation and the online `man` pages. Other recommended reading is as follows:

MUD/102 : emacs – Unix editor.

MUD/121 : Unix for beginners.

MUD/122 : An introduction to display editing with vi.

MUD/123 : vi – Quick reference card.

MUD/124 : An introduction to running Fortran programs on Unix

SGP/7 : Unix and Starlink.

SGP/28 : How to write good documents for Starlink.

SSN/66 : Starlink software organisation on Unix.

SUG : Starlink User's Guide.

SUN/1 : Starlink Software Collection.

SUN/9 : LATEX — Document preparation system.

SUN/129 : XDISPLAY - Setting remote X windows.

SUN/144 : ADAM — Unix version.

SUN/168 : JED — The text Editor.

SUN/169 : PINE — Electronic mail interface.

SUN/170 : Editors and Mail on Unix.