Short Guide To UNIX for Java Programming Courses

The following is a condensed version of the CICS introductory guide to UNIX, which was written by Bob Booth and Ian O'Sullivan, and can be viewed in its entirety at:

```
http://www.shef.ac.uk/cics/docs/unixpage.html
```

In this abridged version, material that is specific to the CICS network has been removed and we cover only those aspects of UNIX that are relevant to those learning Java on the DCS UNIX network. It is assumed that you already know how to logon to a UNIX box.

1. Introduction

The UNIX operating system was developed in the early 1970s by a group of enthusiasts at Bell Laboratories in the USA. Since then it has been modified by a number of different groups and has evolved into many similar but not identical flavours. In addition, UNIX is structured so that the user works within a 'shell' that can be configured by the UNIX administrators, working behind the scenes.

1.1 The Shell

The user interface of UNIX is provided by a command language called the *shell*. There are several shells available differing in the facilities they provide. At Sheffield you will use the C shell by default.

1.2 General syntax of commands

UNIX distinguishes upper case letters from lower case, and *insists* that many commands are written in lower case. Typing a command in upper case will probably generate the response

```
Command not found
```

A typical UNIX command consists of a general command word, which may be followed by optional parameters that specify more precisely what you want the command to do. Many of these options consist of a single letter. If a command operates on files then the filenames must come after the options.

```
command [option …] [filename …]
```

1.3 Entering commands

UNIX is a case sensitive operating system and it insists that (almost) all commands be typed in lower case. For this reason you should make sure that the CAPS LOCK key on your keyboard is not activated.

If you make a mistake when typing a command you can correct it using the BACKSPACE or DEL key. What you can't do is use the arrow keys because strange things will happen. If you want to clear the entire command line you can hold down the CTRL key and press u (CTRL-u). Pressing ENTER / RETURN sends a typed command to the shell.

2. Basic UNIX Commands

Basic UNIX commands consist of a lower case command word that uses the minimum number of characters to signify its function. Command words may be followed by one or more optional parameters, often consisting of a single letter preceded by a hyphen, e.g.:

`ls`	list directory
`ls -l`	list directory in long format
`ls -al`	list directory in long format, listing all entries

Format of UNIX commands

```
command [option …] [filename …]
```

2.1 Filenames

When you work on a computer you create files. Files hold information such as programs, data sets and text passages. Each file is distinguished by a unique name.

Filenames in UNIX may include any number of letters and digits. You can use full stops, hyphens, and underscores (`.` `-` `_`) to divide the name into sections for clarity. There must be no spaces within a filename. For example a name of a report for 11 March 1991 might be.

```
rep.11-3-91
```

UNIX distinguishes between upper and lower case, so the two filenames `MYFILE` and `myfile` are different.

The asterisk * has a special meaning, it is called a wildcard character. We use the * to stand for any character or sequence of characters.

2.2 Listing Your Files

The first thing you need to be able to do is produce a list of your files. To do this you use the `ls` command with the options `-l` (long listing) and `-a` (all entries). So you would type in

```
ls -al
```

and you will receive a response similar to:

```
{22} ls -al
total 9
drwxr-xr-x  2 course01       512 Oct 18 14:31 .
drwxr-xr-x 27 root          512 Oct 17 09:52 ..
-rw-r--r--  1 course01       715 Mar 18  1991 .cshrc
-rw-------  1 course01        27 Oct 18 14:31 .history
-rw-------  1 course01       708 Jan 25  1993 .login
-rw-------  1 course01       402 Feb  5  1991 .logout
-rw-------  1 course01      1891 Oct 18 11:09 UNIX.intro
-rw-------  1 course01         0 Oct 18 11:08 empty_file
-rw-------  1 course01        57 Oct 18 11:09 hello.c
{23}
```

From left to right this list details:

File type and access permissions
Number of links
Name of the owner
Number of bytes (characters) in the file
Date and time the file or directory was last updated
File or directory name

2.3 Displaying the Contents of a File

You can look at your files on the screen using the `more` command:

```
more filename
```

The contents of the file will be displayed and, at the bottom of the screen will be the prompt:

```
--More--(xx%)
```

To continue you can press:

SPACEBAR	next screenful
ENTER	next line
q	quit listing
?	list commands

The `more` command has many subcommands for finding strings and moving up and down the file. You can find out about these by consulting the on-line help (see later).

2.4 Copy, Rename and Remove

The command to copy a file is `cp` and the syntax is:

```
cp oldfile newfile
```

To copy a file called `empty_file` to a file called `copyfile` use the command:

```
cp empty_file copyfile
```

To rename and move files we use the `mv` (move) command. For example, to rename the file called `copyfile` to `newfile` we would use

```
mv copyfile newfile
```

If you attempt to to overwrite an existing file at Sheffield, then the C-shell will generate a warning. This behaviour is not the default behaviour (which enables overwriting).

Lists of one or more files can be deleted with or without confirmation using the `rm` command. The command was designed to delete files without confirmation, unless otherwise stated. However we have re-configured our machines so that the command asks you to confirm each delete.

To delete files with confirmation before each one, type

```
rm file1 file2 ...
```

Or you can bypass our configuration by preceding the command with the \ character. So to delete files without confirmation the command is

```
\rm file1 file2 ...
```

Be careful when using this command not to delete your entire filestore.

3. Redirection and Piping

The results from a command are normally displayed on the screen. There are special symbols which redirect the output of a command. For example, you may want to save the output to a file rather than display it on the screen.

For example, to save a directory listing in a file called `dir_list` you would use:

```
ls -l > dir_list
```

The output of one command can become the input to another command. Commands strung together in this fashion are called a pipeline. The symbol for this kind of redirection is a vertical bar | called a pipe.

```
who | wc -l
```

The command `wc` with the `-l` option counts lines. So the pipeline is counting the number of users who are logged in.

Another example is to use the `more` command to scan through a long directory listing

```
ls -al | more
```

4. Getting Help

Comprehensive on-line help is available for all UNIX commands. The information is stored in files called reference manual pages and these can be viewed using the `man` command. To get a description of any command, type

```
man command
```

So to view the manual pages on, for example, the `ls` command you would type

```
man ls
```

The pages would then be displayed via the `more` command, so you can use the SPACEBAR to advance the manual page and the q key to quit the listing. It is well worth looking at the manual pages on the `more` command simply to help you view the manual pages! In addition you can get a full description of the man help system by typing

```
man man
```

UNIX has a rich variety of commands for doing many things. The manual pages give you full information on each command and, usefully, the last few lines of each entry give cross references to other related commands. If you are having trouble locating information on a particular topic you can type

```
man -k topic | more
```

and this will list possible sources.

5. Working with Directories

When you start to work on your UNIX machine you will quickly accumulate files and so will need some means to manage them before they multiply out of control.

To do this you can create a directory structure. Directories are like divisions that can contain files. By using a directory structure you can keep related files together, but separate from other files. In addition any directory can contain further divisions called subdirectories in case you need to sub-divide your groups of files further.

5.1 Working Directory

At the very top of UNIX's directory structure is the *root* directory. This contains a number of directories, each of which contain their own subdirectories. When you log on to UNIX you are put into your own private directory, called your *home* directory. To get an idea where you lie in the directory structure give the command

```
pwd
```

This stands for 'print working directory', and is useful as you navigate the directory structure. You will get a response similar to:

```
/home/guy
```

Note that the character that divides the directories and filename in a path is a forward slash '/', whereas in MS-DOS a backslash '\' is used.

5.2 Changing Directory

Once you know where you are you can start to look around using various forms of the change directory command, `cd`.

`cd ..`	Travel up the structure, towards the root, one step at a time.
`cd /`	Jump straight to the root.
`cd`	Return to your home directory.
`cd mydir`	Go to a subdirectory called, in this case, `mydir`.

5.3 Making and Deleting Directories

You can create and remove your own subdirectories using the command:

`mkdir mydir2`	To make a directory called, in this case, `mydir2`.
`rmdir mydir2`	To remove it again.

6. Running Programs

Programs run on Unix will operate either in the *foreground* or the *background*. Programs run in the foreground are connected to the terminal session, and will occupy that session until the program terminates. Programs run in the background are independent of the terminal session; whilst the program is running you can execute other Unix commands and even log out from Unix leaving the program to complete in its own time.

6.1 Running a Program

To run a program in the foreground type the program name and press the ENTER or RETURN key. To run `ansys` in the foreground you would type:

```
ansys
```

To run a program in the background type the command then add the & character and press the ENTER or RETURN key. To run `clock` in the background you would type:

```
clock &
```

If a background program produces output then you must direct this to a file

```
myprog >& outputfile &
```

This command runs the program `myprog` in the background. Output (and errors `>&`) are directed to the file `outputfile`.

6.2 Controlling a Program

For a program running in the foreground of the current session:

CTRL c	will completely kill the program.
CTRL z	will stop the program, but the program will still exist in the system.

To control background programs in the current session the following commands are available:

`jobs`	List programs (jobs).
`bg %`*`job_id`*	Places a job in the background.
`fg %`*`job_id`*	Returns a job to the foreground.
`stop %`*`job_id`*	Stops a background job.
`kill %`*`jobs_id`*	Kill a job.

Where *`jobs_id`* represents the number or a job returned by the `jobs` command.

7. Repeating Previous Commands

The system can keep a history of the commands you type and is able to repeat previous commands. The `history` command will list your previous commands:

```
history
1 ls
2 ls -a
3 more UNIX.intro
...
```

To re-issue a command type `!n`, where `n` is the number of the command from the history list. Alternatively you could type !*string* to execute the most recent command starting with the specified *`string`*. For example if your most recent history is

```
75 javac Simple.java
76 java Simple > out.txt
77 more out.txt
```

you can re-issue the `javac Simple.java` command by typing

```
!75
```

or

```
!javac
```

To re-issue your previous command, you need only type

```
!!
```