THE UNIVERSITY OF SYDNEY

INFO1103 Tutorial 1

Introduction to Unix

Using Ed

Familiarise yourself with Ed, which is accessible from https://edstem.com.au. The Ed learning platform has been created to help empower students and staff, incorporating a modern design that is simple and easy to use.

The discussion system within Ed should be used for any questions or posts that relate to the content of this course. Before asking a new question, look through the other questions as someone else may have already asked a similar question. This will maximise the efficiency of your tutors and fellow students answering questions.

Introduction to Unix

What is Unix

Unix is a family of operating systems that derive from the original AT&T Unix developed in the 1970s at Bell Labs research centre. Operating systems manage the computer's hardware and software, providing a common interface that applications can use.

There are several Unix derivatives with the most popular being BSD and Linux of which there are many distribution including: Mac OS X, FreeBSD (BSD) and Ubuntu, Debian, Fedora (Linux).

The Unix philosophy

The Unix philosophy was originated by Ken Thompson during his time at Bell Labs that aims to develop small, well defined tools that work together and remains the single most important revolution in the history of software systems. Doug McIllroy, the inventor of the Unix pipe, summarises it as:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Kernel and shell

Unix consists of a kernel and many programs. The kernel is the master control program that handles low level tasks, for example: starting and stopping programs, managing the file systems, providing access to connected peripherals and so on.

The programs make up a set of tools, where each tool has a limited well defined function and all tools can communicate with each other through the filesystem. These tools can be combined to perform increasingly complex tasks using the shell.

The shell is a user interface for Unix that interprets and executes the commands typed by the user. Shell can also be used to navigate the file system and modify its contents, as well as combine Unix programs using piping and redirection.

Files and processes

Everything in Unix is either a file or a process. File is any collection of data, e.g. an image, text document or a program, while a process is a program currently being executed. Each process is identified by a unique PID (process ID) chosen by the operating system.

Files can be created in many ways. For example, a user can create a file by typing text into a text editor and then saving the text in a file on the disk. The user might choose to save these kinds of files into a designated 'documents' folder for easy access.

In addition, system files are also created by the operating system and by programs running in the background. System files are generally not accessible to the user since tampering with these files can cause certain programs to cease functioning properly.

Files do not have to physically exist on a storage volume; some may exist completely in memory, in transmission across the network, or generated on the fly by the operating system or a program. For example, let's inspect the contents of the $/proc/uptime$ file a few times:

```
$ cat /proc/uptime
76530.83 305381.40

$ cat /proc/uptime
76531.94 305385.82
```

We can see that the contents of this file is changing every time we read it. This is because the operating system is generating the contents of the file on the fly. As such, the $/proc/uptime$ file and others inside the special $/proc$ directory do not occupy space on the hard disk, but instead are special files whose contents are determined by the operating system.

Other special device files worth mentioning live in the $/dev$ directory. These files are interfaces to a physical device on the system. Software can read and write data to these files as if they were ordinary files as a way of interacting with the computer's hardware.

Terminal basics

The terminal is the main way of interacting with the Unix environment as opposed to the graphical user interface GUI in operating systems like Windows. At first, the terminal may seem unusual and inconvenient but you will soon learn that it is much more powerful and offers greater flexibility.

Keyboard shortcuts

Here are several keyboard shortcuts that are helpful to know when using the terminal

```
ctrl + c   terminates the current running program
ctrl + d   sends end of file EOF to the running program
ctrl + a   moves the cursor to the beginning of the prompt
ctrl + e   moves the cursor to the end of the prompt
```

Wildcards and tab completion

Wildcards and tab completion allow you to reduce the amount of keyboard input for commands.

The wildcard $*$ is a special character interpreted by the shell that is replaced with the names of matching files as if the user had typed the filenames explicitly. For example, suppose we are in a directory containing the files $a.txt$, $b.txt$ and $c.txt$. The following three commands are equivalent:

```
$ cat a.txt b.txt c.txt
$ cat *.txt
$ cat *
```

When typing out a command, your shell also provides a tab completion feature where commands and filenames are guessed from the first few letters that you have typed. For example, at the prompt type “ec” and then press the tab key, your shell will guess that you want to type “echo”, completing the name of the command for you.

In addition, you can use tab completion to complete the path of a particular file or folder you want to reference. Often there are times when a file is hidden deep within many folders away from you.

Instead of typing out each intermediate directory, which you may not know the names of anyway, your shell can assist you by expanding the path to the file with the closest matching filename, stopping mid path if there is more than one possible match. Simply type a few more letters, enough to resolve any ambiguity and then press the tab key again to continue completing the path.

Getting help

The **man** command allows you to read what is referred to as the manual pages which contains help and documentation for Unix commands, library functions, system calls and so on.

```
$ man <command>
```

Use the arrow keys to move up and down and press the q key to quit, similar to $less$.

The **whatis** command displays a short description of the command that is specified

```
$ whatis <command>
```

Navigating the filesystem

The Unix file system consists of a set of organised files and folders. Everything in the file system is stored within a folder, which can also be stored within other folders, resulting in a tree hierarchy.

The location in the file system of a particular file or folder is represented by its $path$. The path of the root directory, by which all files and folders are contained within, is represented by a single forward slash $/$. For example: the folder bin resides inside the root directory has the path $/bin$.

Similarly, the path $/usr/bin/zsh$ refers to the executable file zsh inside the usr folder, which itself is inside the bin folder, which itself is inside the root directory.

Directory shorthands

The shorthands below are used extensively to make navigating the directory faster and simpler.

```
/   root directory
~   home directory
..  parent directory
.   current directory
-   previous directory
```

Absolute and relative paths

An absolute path always starts with a forward slash. Files and folders are referenced relative to the root of the filesystem. A relative path does not start with a forward slash and instead references files and folders relative to the current working directory. For example, compare the two paths below:

```
scott at Retina in ~
$ ls /Users/scott/Documents/2016

scott at Retina in ~
$ ls Documents/2016
```

The **pwd** command outputs the absolute path of the current working directory.

```
$ pwd
```

The **cd** command allows you to navigate into another folder.

```
$ cd /         go to the root directory
$ cd ~         go to your home directory
$ cd ..        go to the parent directory
$ cd <path>    go to the directory at the specified path
```

The **ls** command outputs a list of files and folders in the specified directory.

```
$ ls           list contents of the current directory
$ ls /         list contents of the root directory
$ ls <path>    list contents of the given path
$ ls -a        list contents and include hidden files and folders
$ ls -l        list contents and include detailed information
```

Creating files and folders

The **touch** command allows you to create files or update file timestamps if the file exists.

```
$ touch document    creates a file in the current directory
$ touch notes.txt   creates a file with the .txt extension
$ touch ~/numbers   creates a file in your home directory
```

The **mkdir** command allows you to create folders.

```
$ mkdir folder              creates a folder in the current directory
$ mkdir -p ~/photos/2016    creates intermediate folders as required
```

Manipulating files and folders

The **cp** command allows you to make a copy of one or more files or folders.

```
$ cp /usr/share/dict/words ~/            copy file to home directory
$ cp /usr/share/dict/words ~/documents   copy file to documents folder
$ cp <file1> <file2> ... <destination>   copy group of files to destination
$ cp -r <folder> <destination>           copy folder and its contents
```

The **mv** command allows you to rename a file or folder or move it into another folder.

```
$ mv DS1234.jpg photo.jpg   renames the file to photo.jpg
$ mv photo.jpg ~/pictures   moves the file into the pictures folder
```

The **rm** command is one of the most dangerous commands as it removes the given files and folders, and instead of placing these folders in a Recycle Bin it removes them permanently.

```
$ rm photo.jpg        deletes photo.jpg
$ rm one two three    deletes files one, two and three
$ rm -rf ~/pictures   deletes the folder and all of its contents
```

Working with text and files

The **echo** command displays the arguments given to it, including any environment variables.

```
$ echo Hello there   outputs "Hello there"
$ echo $PATH         outputs folders to search through for executable files
```

The **printf** command displays the arguments given to it, and is capable of processing format strings.

```
$ printf "Hello there\n"      outputs "Hello there"
$ printf "Hello, %s" Alice    outputs "Hello, Alice"
```

The **cat** command outputs the contents of each file given to it.

```
$ cat /usr/share/dict/words         outputs a list of words
$ cat /proc/cpuinfo /proc/meminfo   outputs cpu and memory information
```

The **less** command is the opposite of more and is an advanced text viewer that allows you to navigate through the text using the following commands.

```
up arrow, down arrow   up or down, one line at a time
page up, page down     up or down, one page at a time
/<pattern>             find pattern in the file
n, N                   next or previous result
g, G                   start or end of file
h                      help
q                      quit
```

Pipes and redirects

Pipes and redirects allow to combine Unix commands and small programs to produce elegant functional solutions. Redirection uses the following symbols: $<$, $>$, $>>$, and | is used for piping.

$>$ will redirect the output of one program to a file, rather than the terminal. If no file with the specified name exists, a new file will be created. Otherwise, any content of the existing file will be overwritten by the output of the program. This is something you need to be careful about.

```
$ cat /usr/share/dict/words > ~/words
```

$>>$ is very similar to $>$, however if the specified file already exists then it will append the output of the program to that file without overwriting any existing data.

```
$ echo "# List of words" > ~/words
$ cat /usr/share/dict/words >> ~/words
```

$<$ will send the contents of a file as standard input to the program.

```
$ tr a-z A-Z < /usr/share/dict/words
```

You can combine input and output redirection, like so:

```
$ tr a-z A-Z < /usr/share/dict/words > ~/WORDS
```

| will send the output of one program to the input of another. Below, the output of the $grep$ command is sent as input to the wc command, that outputs the number of words starting with the letter a.

```
$ grep "^a" /usr/share/dict/words | wc -l
```

The pipe is powerful because it allows us to combine the functionality of multiple programs in order to solve a problem. Here is a more complicated one liner:

```
$ grep "^a" /usr/share/dict/words | sort -r | less
```

It works like this:

- Find all words starting with the letter a
- Sort these words in reverse alphabetical order
- Opens the result in the $less$ program for viewing.

Exercises

Using your knowledge of Unix, complete the following tasks and write down the commands for each step.

- Output "Hello world" to the terminal
- Output your current working directory
- List the contents of your home directory including any hidden files
- Create the folder sun in your home directory
- Navigate into the sun folder you just created
- Create a file with the text "Modern Unix" in a file called $unix.txt$
- Append the text "ZFS + DTrace + Zones + KVM" into $unix.txt$
- Rename $unix.txt$ to $solaris.txt$
- Copy $solaris.txt$ to $illumos.txt$ in the same directory
- List the contents of the sun directory
- Delete the $solaris.txt$ and $illumos.txt$ files
- Use the $less$ command to view the $/usr/share/dict/words$ file
  - Navigate through the file
  - Search the file for the word $zythum$
  - Jump to the end of the file.
  - Jump to the start of the file.
  - Quit less