FIT2070 Operating Systems

Semester 2 2016

Practical 1: Getting started with Unix and C

Part I: Pre-lab Preparation

1. Let us get started

The purpose of this lab is to ensure that you have a basic familiarity with the UNIX system and some general-purpose UNIX commands (or help you review some UNIX commands if you already know) as well as how to compile C and Java programming.

A utility similar to the SSH Secure Shell Client is needed for you to log into the UNIX server and we will use the PUTTY free program for this purpose. If this is not on the machine, you many need to download from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

As an alternative, the following Unix server is setup for you.
Server name: ra-clay1.its.monash.edu.au or use the IP address: 130.194.1.2
Username/Password: your Authcate name and the password.
This server can be accessed from outside Monash too through the Internet.

2. The Unix manual pages

The UNIX manual pages (also called **man pages**) provide information about most of the UNIX commands. To find and display manual pages, the man command is used. To display the man page for the **man** command, type the command:

```
$  man  man
```

(In our tutorials the **$** at the start of a command line is the shell prompt. Please do not type that as part of your command. Your shell prompt can be different depending on the shell that you use.)

The **man** command enters into a pager mode, and will only display the first 20 or so lines of the man page. You can use a number of commands in the pager to scroll the text:

Press	**[enter]**	to scroll forward to the next line of text
Press	**[space]**	to scroll forward to the next page of text
Press	**h**	to display a list of the commands that the man pager accepts
Press	**q**	to quit and return back to the shell prompt

Find out how to scroll backwards a page and then scroll back up to the top of the **man** page.

It is important to realise that the **man** pages do not contain information about all of the programs installed on the system. You should also realise that the content of many of the **man** pages is aimed at experienced users, and if you are new to UNIX you may find this a little overwhelming.

Use the **man** command to find more information about the following commands:

```
ls
more
cat
vim
```

3. When a command has to be interrupted?

Suppose that you inadvertently pressed **[*Enter*]** just after entering **cat**:

```
$  cat  [Enter]
```

There is no action here; the command simply waits for you to enter something. You can use:

[*Ctrl-d*]

to get back to the shell prompt. To interrupt a running program you can also use one of the following:

[*Ctrl-c*]

4. Try some Unix commands

UNIX commands are essentially **executable files** representing programs – mainly written in **C**. These files are stored in certain directories such as **/bin**. UNIX commands are in lowercase (remember that case is significant in UNIX). The syntax for UNIX commands is invariably of the form:

```
$  command  -option1  -option2...  other-arguments
```

Let's try out some UNIX commands.

Note: *You can press the UP arrow key to repeat a previous command, which is especially handy for repeating long command lines.*

Before trying out some commands, create a directory named **fit2070** under your home directory and then change to this **fit2070** directory, using the following commands:

```
$  mkdir  fit2070
$  cd  fit2070
```

Try to understand the following commands:

who
w
users
finger

These commands can be used to find out who the users are as UNIX is a system used by multiple users.
What are the differences between these commands (check their man pages)?

who am i

Sometimes, you may forget your username, especially when you have a number of accounts on this server. The additional words used with who are known as arguments.

tty

Know your terminal name (a hardware device is also a file in the UNIX file system.)

whatis

Sometimes, you would like to know just what a command does and not get into its syntax. For example, what does the **cp** command do? The **whatis** command provides a one-line answer:

```
$  whatis  cp
cp      - copy files and directories
```

Once you have identified the command you need, you can use **man** command to get further details.

whereis

This command can be used to locate the executable file represented by a command (as mentioned above, UNIX commands are essentially executable files representing programs). Try this:

```
$  whereis  pwd
```

apropos

This command performs a keyword look-up to locate commands. For example, if you wonder what the command to copy a file is, the following could be typed**:**

```
$  apropos  "copy files"
```

man -k

This is an alternative to **apropos**. You can try this to verify:

```
$  man  -k  "copy file"
```

Answer the following questions:

Which command converts an image to jpeg format (jpeg is a popular data compression method)?

Which command displays the status of disk space (the number of free disk blocks) on a file system?

Which command displays the type of a file (ascii text, executable, etc.)?

script

This command lets you "record" your login session in a file. All the commands, their outputs and the error messages (if any) are stored in the file for later viewing. If you are doing some important work, and wish to keep a log of all your activities, then you should invoke this command immediately after you log in:

```
$  script
Script  started,  file  is  typescript
```

The (possibly new) prompt returns, and from now on all your keystrokes that you enter here as well as the outputs of the commands that you entered get recorded in the log file named **typescript**. After your recording is over, you can terminate the session by entering:

```
$  exit
Script done, file is typescript
```

You can now view this file with either of the following commands.

```
$ cat typescript
$ more typescript
```

Please note that the command script **overwrites** any previous typescript that may exist. If you want to append to it, or use a different log file name, as shown below.

```
script  -a
script  logfile
```

clear

Clear your screen.

date

Display the system date.

uname -n	shows your machine name.
uname -r	shows the operating system's version number.
uname	shows the operating system.

cal	display the calendar.
cal 7 2001	
cal 7 2999	
cal 7 2011	

wc

This command counts the number of lines, words, and characters in a file. Try the following command and explain the output: **$ wc /etc/passwd**

Generally the **passwd** file, which is a text file, contains information about all the users registered on the system. You can use the command **cat /etc/passwd** to read it. For security reasons, the password file may be stored some where else too depending on the version of Unix/Linux. For example, on **ra-clay1** machine, the **passwd** file is located in **/var/db**.

grep

This command searches a file for a pattern and displays all the lines (of the file) that contain the given pattern. Try the following command line:

```
$ grep <your username> /etc/passwd (or /var/db/passwd)
```

For example, if your username is **abc** then the command line is**:**

```
$ grep abc /etc/passwd
```

Are you able to find out how many registered users (on the system) have "Andrew" as part of their name?
Hint: Check out the **-c** option of **grep**.

5. Combining commands

So far you have been executing commands separately. UNIX allows you to specify more than one command in the same command line. Each command has to be separated from the other by a **semicolon** (**;**):

```
$ who; date; cal
```

You can even redirect the output of these commands to a single file.

```
$ (who; date; cal) > newlist
```

You can then view the content of this file with the **cat** command**:**

```
$ cat newlist
```

6. Some more commands

info

Displays the information page for a given command; similar to a **man** page.

```
$ info pwd
```

more

Displays the contents of a file on screen, one screenful at a time. Press the **SPACE** key to get the next screen**.**

```
$ man man > test.txt
$ more test.txt
```

A similar command is **less**. Can you find the difference between **more** and **less**?

touch

Creates a new, empty file.

```
$ touch test2.txt
$ cat test2.txt
```

zcat
Displays the contents of a compressed file on the screen.

echo
Prints the text on the screen. Will be useful in shell programming (scripting).

```
$echo "Hello World"
```

history
Displays all of the stored commands in the history list.

head
Displays the first few lines of a text file (10 lines by default).

```
$  head  test.txt
```

A similar command is **tail**, which displays the last few lines of a text file (10 lines by default).

free
Display the amount of free and used memory (physical and virtual), with basic information about how that memory is being used.

jobs
Lists jobs (processes) started in the current shell environment.

sleep
Causes the shell to sleep for a specified number of seconds.

```
$ sleep 5
```

7. The Unix files

All files in a Unix file system are treated equally – that is, the system does not distinguish between a text file, a directory file or other file types. It is up to the user to know the type of file they are using, which can result in operations being attempted on incompatible file types (e.g. printing executable output files to printers). The command **file** can be used to give a fairly reliable description of the contents of a file. Try the following commands and compare the output.

```
$  file  /bin/tcsh
$  file  /bin
$  file  /etc/passwd
```

8. Commands used to handle files and directories

The Unix operating system provides a number of commands to create, modify, traverse and view file systems:

cd	**change into another directory**
pwd	**print the present (working) directory**
ls	**list a directory's contents**

mkdir	**make a directory**
rmdir	**remove a directory**
rm	**remove a file or a number of files**
mv	**rename/move a directory or file**
cp	**copy a file**
df	**display information about free space on the available file systems**
du	**display disk usage information (for files or directories)**

Make sure you know what each of the commands does and how to use each command.

The **cd** command without any argument takes you back to your home directory from anywhere.

The **ls** with option **l**:

```
ls  -l
```

will display more information about files. For example, you can see the file size.

Make sure you know how to use the special characters "*****" and "**?**" with these commands.
Make sure you know how to use the "**..**", "**.**", and "**~**" short cuts with these commands.

*****	(asterisk)	represents all ordinary files in a directory
?		represents a single character
~	(tilde)	represents your home directory

Try the following (make sure you understand what's happening and the outputs):

```
$    man  man  >  f01.txt
$    cp  f01.txt  f02.txt
$    cp  f01.txt  f001.txt
$    cp  f01.txt  f002.txt
$    cp  f01.txt  testf01.txt
$    cp  f01.txt  testf02.txt
$    cp  f01.txt  f1.txt
$    cp  f01.txt  f2.txt


$    ls
$    ls  f*.txt
$    ls  f?.txt
$    ls  f??.txt
$    ls  f???.txt
$    ls  test*.txt
$    ls  ??.txt
$    ls  ???.txt
$    ls  ~
```

After you have a good understanding of the above, remove all of these text files using the following:

```
$  rm   f*.txt  t*.txt
```

9. The Unix file attributes

Each file in the Unix file system has a number of attributes associated with it. Some attributes that are associated with a file are:

name
size
permissions/protection
time/date of modification
owner
group

To investigate file attributes, change your working directory to your home directory. Get a full directory listing by typing the command **`ls -al`**. Make sure you know what each piece of information means.

Try these too:

a) Under the directory **`fit2070`** make a subdirectory called **`testdir`**.

b) Create an ordinary file called **`file.txt`** under **`testdir.`**

c) Check the current file attributes (especially the permissions) of **`file.txt`**.

d) Use the following commands to change the permissions of **`file.txt`**. Verify the effect of each command using **`ls -l file.txt`**. Make sure you understand the permissions represented by each octal number:

```
chmod       000   file.txt
chmod       777   file.txt
chmod       666   file.txt
chmod       444   file.txt
chmod       664   file.txt
chmod       600   file.txt
chmod       466   file.txt
chmod       251   file.txt
chmod       111   file.txt
chmod       700   file.txt
```

10. Use `find` to locate files

One of the powerful tools of the Unix system, **`find`** recursively examines a directory tree to look for files either by name or by matching one or more file attributes. Use the **`man`** page to know the options for the **`find`** command.

a) Under the **root** directory, find all files that are more than 2 months (60 days) old.

b) Under the **root** directory, find all files that has two character file names.

c) Under the **root** directory, find all files that has 2-uppercase character file names (e.g., AB, BG).

11. Shell

Pre-Processing of Commands by Bourne Shell

Life cycle of a shell can be described as follows:

- Shell places the prompt on the user terminal and goes to sleep.
- User types a command line consisting of one or more commands. These commands may be separated by the following symbols: **;** (sequentially execute), **||** (otherwise execute), **&&** (if ok then execute).
- When the user presses **[*Enter*]**, the shell begins processing the command line.
- First step in this pre-processing is to parse the first command. If there are more than one command in a line they will be processed and executed after the first command has finished execution. However, the decision will be based on the separator (**;**, **||**, or **&&**) you use between the commands.
- The command is broken into its constituent words. End of the words are usually identified by the spaces, tabs and special symbols. The command line parser is often not as nice as those used by the programming languages. As a result, sometimes your command may not be understood if there is an extra space or you miss a space between the words.
- Next, the shell replaces variables by their values. These variables are shown in the command by preceding them with a **$**.
- Command *substitution* is done next. A command substitution is indicated by enclosing the command in a pair of *back-quotes* (` `). This quote is usually found on the left end of the top row on your keyboard.
- Shell then performs redirection of the standard input, standard output and standard error output, if requested.
- Wild-cards are expanded next.
- Finally the command is ready to execute. The shell searches for an executable file whose name matches the command name.
- While the command executes, the shell waits.
- When the execution finishes, the shell displays next prompt on the terminal. A new cycle begins.

In your home directory, do the following:

First invoke the Bourne shell by typing **sh** at the command prompt. Use the following command to create a file**.**

```
$  cat > chap01
#include <stdio.h>

main()
{
   printf("You are the best in the class.\n")
}
```

Use ***Ctrl-D*** to finish the session with the **cat** command.

Note 1: This is a simple C program which prints a message. The only surprise for **Java** students must be the first line:

```
#include  <stdio.h>
```

Actually the include here is somewhat similar to the **import** mechanism used in **Java**.

Note 2: The above **C** program has an error. There should be a **;** at the end of the longest line. This error is deliberate.

Some of you may wish to use a text editor (such as **pico**, **vim** or **joe**) to create a text file**.**

Part II: Scripting Exercises

Purpose of this Lab Exercise

- To understand the nature and purpose of the Unix shell scripts.
- To understand the advantages that the shell programs provide in aiding common tasks.
- To practice and use some simple methods for customising the actions of the scripts based on the user input.

Task 1

The shell script, shown below, is from a file called **helpme**. When executed the script asks the user to specify the topic (a keyword) on which the help is required. Internally it uses a standard Unix command to list each manual page that has the specified keyword occurring in it. The command **man -k** is an alternative to the command **apropos.**

```
#!/bin/sh
echo -e "What do you need help on:\c"
read topic
man  -k $topic | tee file1
```

Enter the script in a file called **helpme**. Use the command **chmod** to assign *execute* permission to you. Use the command **ls -l** to verify the permissions.

Run the shell script by entering the following keywords, respectively:
jpeg
telnet
login
chat

Also try other keywords (made up by you) on which you might require help.

What does the command **tee** do in this script?

Note the use of shell variable, **topic**, to assign a value to it and to later retrieve the value from it. How will you change the script if the topic can be a phrase made of multiple words such as:
remote login
disk usage
file type

Task 2

Experience shows that the list generated for most key topics is long and unmanageable (especially when you only provide one word instead of a phrase)! Fortunately, **apropos** as well as **man -k** lists each command with a 1-line description of the command. We can refine the list by searching for lines containing (or not containing) secondary keywords and clues.

In this task we will try to seek one more keyword to reduce the size of the list. Take note of the use of the case construct.

Append the following script to the **helpme** file (add after the last statement) you created in **Task 1**.

```
echo  -e  "\n\n Is the list of man pages too long?"
read  YN
case  $YN  in
[yY]*) echo  -e  "Please suggest another keyword:\c"
       read topic
       cat file1 | grep $topic | tee file2
       rm  file1
       mv  file2  file1
      ;;
[nN]*) echo  -e  "Good. Bye for now.\n"
      ;;
esac
```

Make sure that you understand the purpose of each line in this script.
Run this script by first entering the keyword *remote* and then the keyword *login*.
Enter **y** to answer the question **"Is the list of man pages too long?"**

Run this script again to work on the following input pairs:
jpeg (first keyword), *compress* (second keyword)
memory (first keyword), *statistics* (second keyword)
cpu (first keyword), *time* (second keyword)

Has the content of the file **file1** changed (compared it with that from **Task 1**)?

Run this script by entering other keywords made up by you.

Task 3

In this task we will repeat the list shortening step (as was completed in **Task 2**) two times. If the list is not adequately short by the end of the second step, the user will be asked to remove some **man** pages based on a keyword that they do not want to read about. The script below is a major re-write of the **helpme** file. Some of you would like to copy your current version of **helpme** to another file before replacing the whole text of file **helpme** with the text (script) below.

Note that the command **grep** has been used with some options specified. How do they help?

```
#!/bin/sh
echo –e  "What do you need help on:\c"
read topic
man  -k  $topic | tee file1
#
RepeatsLeft="2"
while [ $RepeatsLeft -ne "0" ]
   do
     RepeatsLeft=`expr  $RepeatsLeft  -  1`
     echo –e "\n\n Is the list of man pages too long (y/n)?"
     read YN
     #
     case  $YN  in
     [yY]*) echo –e "Please suggest another keyword:\c"
            read topic
            cat file1 | grep -i $topic | tee file2
            rm file1
            mv  file2  file1
          ;;
     [nN]*) echo –e "Good. Bye for now.\n" exit 0
          ;;
     esac
done
echo –e "Provide a keyword that you wish to exclude: \c"
read AntiKey
cat file1 | grep -iv $AntiKey > file2
more file2
```

Explain the functionality of the following statements in the above code:

```
RepeatsLeft=`expr  $RepeatsLeft  -  1`
cat  file1  |  grep  -iv  $AntiKey  >  file2
```

Run the script by entering the following keywords (Enter **y** to answer the question **"Is the list of man pages too long?"**):

file (first keyword)
compress (second keyword)
zip (third keyword)
bzip2 (last keyword)

Have you seen a shorter list after each extra keyword has been entered?

Task 4

In this step we will develop a new script program to display the **man** pages selected in the previous task. We call this script file **showme**. But first you need to modify the last statement (**more file2**) in the script **helpme** so that it now becomes:

```
./showme  file2
```

Also replace the command line **echo –e "Good. Bye for now\n"** with

```
./showme  file1
```

The following script **showme** uses the command **cut** to extract the first column of a formatted output generated by **man -k $topic**. You should not run **showme** independently. The script **showme** is launched when you run **helpme**. The script for **showme** is as below:

```
#!/bin/sh
# get the command names
# They are in the first column before a tab or a blank character of
a formatted output
#
     LIST=`cut  -f1  $1  |  cut  -d  "  "  -f1`
#
# You can use echo $LIST to view the value of LIST here
#
for R in $LIST
do
     echo  Show $R?
     read  YN
     case  $YN  in
           [Yy]*)  man  $R
           ;;
           *)
           ;;
     esac
done
```

Research the **cut** command. Give examples to demonstrate how it works.

Shell Functions

A shell function executes a group of statements enclosed within curly braces. It optionally returns value with the return statement. Unlike in **C**, a shell function definition uses a null argument list, but requires **( )**:

```
function_name( ) {
    statements
    return value (this is optional)
}
```

The function can be invoked by its name (*without* the parentheses), optionally followed by arguments. The value returned is numeric and represents the success or failure of the function.

Enter the following shell function named **info** onto your shell (the **$** is the shell prompt, the **>** is automatically generated):

```
$  info()  {
  >  echo  -e  "The  current  directory  is:  \c";  pwd
  >  echo  -e  "The  current  users  are:  \c";  users
  >  echo  -e  "Today  is  `date`"
  >  }
```

To run this function simply use the function name:

```
$  info
The current directory is: /home/jojo/fit2070
The current users are: jojo john mary
Today  is  Fri  Aug   8  15:25:33  EST  2015
```

Shell functions can be created at the command prompt (like above). Shell functions can also be defined at the beginning of a shell script using them or at least preceding the function calls. This is because shell statements are executed in the interpretive mode.

The following shell script (named **compile.sh**) compiles a **C** or **Java** program from the current directory that was last modified. First the file name is assigned to the variable file and then compiles the program. The input to the script is either "**c**" or "**j**".

```
#!  /bin/sh
if  [  $#  -eq  1  ]  ;  then
     if  [  $1  =  "j" ]  ;  then
          file=`ls -t *.java | head -1`
          javac $file
     elif  [  $1  =  "c" ]  ;  then
          file=`ls  -t  *.c  |  head  -1`
          gcc  $file  &&  ./a.out

     else echo "Invalid file type"
     fi
else echo -e "Usage: $0 file_type\nValid file types are c and j"
fi
```

We could define two functions within the script to make the procedure clearer:

```
#!  /bin/sh

compj()
{
        file = `ls -t *.java | head -1`
        javac $file
}

compc()
{
        file=`ls -t *.c | head -1`
        gcc $file && ./a.out
}

if  [  $#  -eq  1  ]  ;  then
        if [ $1 = "j" ] ; then
                compj
        elif [ $1 = "c" ] ; then
                compc
        else echo "Invalid file type"
        fi
else echo -e "Usage: $0 file_type\nValid file types are c and j"
fi
```

Save the above shell script as **compile2.sh**. Create the following small **C** program named **hello.c**. Run the script **compile2.sh** to test the defined functions.

```
#include<stdio.h>

main()
{
        printf("Hello  World!\n");
}
```

Create a small **Java** program to test **compile2.sh**.