

## **CS240 Operating Systems, Communications and Concurrency – Dermot Kelly**

### **Practical 2 Command Shell and Shell Scripts**

Last week's practical introduced the Unix environment through the Cygwin package for Windows. We covered some basic commands, how to open an editor to create files and programs and how to navigate the file system by creating and moving files between directories. We noted how to find and set your current directory as you move up and down the file name space from the root '/' to your cs240 directory '/home/username/cs240'.

There are two ways to interact with the operating system, either by using a windows oriented graphical user interface(GUI such as Gnome) or by using a command interpreter program called a shell. When navigating the file system, organising files and launching applications it is probably easier to use the mouse and GUI. The purpose of the GUI is to make the system easy to use, but the kinds of things you can do with it are limited to what appears in its menubars and drop down boxes.

When you interact with the operating system through the Command Shell, although typing text based commands is more cumbersome, it may be more powerful because you can tailor the commands, extend them and compose them with others to do custom tasks as well as being more in control of where the input and output for those tasks is sourced and directed.

The shell is a command interpreter program which has its own mini-language, allowing the user to execute commands, and redirect input and output to those commands and link commands together to create more complex functions. There are a few alternate shell programs, which may use different command syntax, but we are going to use a shell program called the "bash" shell which uses the "\$" prompt character where user input is expected.

We will look at how we can put a series of commands for the shell into an executable file known as a shell script, so that we can execute a complex series of commands simply by executing that single script file under the control of the shell.

The shell is an interactive user interface program which interprets your commands according to a specific syntax and communicates with the operating system to create processes to execute them.

Remember when using the Cygwin/Unix environment the online manual is available to explain the details of any unix commands and the parameters taken by them.

**man <command>**

You can exit the online manual by typing 'q' at any time.

**Write out the commands you use as you are going along if you want to make additional notes.**

#### **1. Start GNOME Desktop**

Start the GNOME Flashback desktop application from the Windows Start menu under the folder Cygwin-X.

When the GNOME desktop is ready, launch an XTerm application from the Applications/System Tools menu.

## **2. The Shell Program**

The shell program reads from a single input stream and writes to a single output stream (by default the input stream is the keyboard and the output stream is the xterminal window associated with the shell). If you have an xterminal window open and a \$ prompt displayed then a bash shell is running waiting for user keyboard input. Keyboard input is read and echoed to the Xterminal window by the bash program so you can see what you type.

The 'ps' command gives you a listing of running processes. Type ps

The list shows many processes associated with the graphical interface as well as your user processes.

Use the mouse scroll wheel to scroll the Xterminal window back to where the list of processes begins and you can see the header information for each column. Your list will look differently.

```
$ ps
  PID   PPID   PGID   WINPID  TTY      UID    STIME COMMAND
  1029     1    994    32280   ?        197609 Feb  3 /usr/bin/dbus-daemon
    812     1    712    29972   ?        197609 Feb  1 /usr/bin/dbus-daemon
   1730   1729   1730    21860  pty0     197609 17:44:21 /usr/bin/bash
```

Each process has a process id(PID), and a parent process id (PPID). Some processes are attached to Xterminals like process 1730 above which is attached to pty0. On the right, we see the executable commands which caused their creation.

Note that to execute a command, the shell 'forks' a separate child process, to execute that command, and then waits until the child process terminates. When the command is completed, the child process exits and the shell process resumes by redisplaying the "\$" prompt.

You can terminate a process at any time by sending it a terminate signal if you know its PID, for example the bash process in the list above has PID = 1730.

## **3. Terminating processes**

Find the PID of a bash process in your list of processes.

The type the command

```
kill -9 PID
```

where PID is the number of the process you want to terminate. The kill utility sends a signal to a process, the parameter signal -9 indicates that the receiving process should terminate. Don't do this willy nilly or your system will stop working.

You may notice your Xterminal window closed when you killed the bash process. If so, you can start a new Xterminal (as done in step 1 earlier).

You can also terminate a shell process by entering CTRL-D (i.e. press CTRL key and d together). This indicates an end of input condition on the input stream and so the command interpreter of the shell cannot continue. Try entering CTRL-D now. What happened? You can open a new shell terminal window under the applications/accessories menu using the GUI.

#### **4. Creating a Shell Script File**

A Unix shell script is simply a **list of one or more Unix shell commands** contained in a file which may be executed by a shell program. This saves us having to retype complex commands which we may create for routine tasks. A script can be used to customise an existing command or to create new more complex and more powerful commands by combining existing ones, using communication pipes, as needed.

Make **cs240** your current directory using `cd /home/username/cs240`  
Create a directory **p2** within **cs240** and make **p2** your current directory.

Put the command “**echo hello world**” into a file called **script1** **using an editor like gedit** and save it in your **p2** directory



Alternatively to using an editor, you can use the shell **echo** command and output redirection symbol as shown below:-

```
echo 'echo hello world' > script1
```

The **echo** command usually writes its parameter to the xterminal window. In this case however, we are telling the shell to redirect the output from the **echo** command to go to a file called **script1**. So a new file called **script1** will now exist with the line '**echo hello world**' inside it.

Open the **script1** file using **gedit**, **nano** or type **cat script1** to display the contents of this file in the shell's window. Note that the file can be used as the basis of a shell script because it contains a valid utility command (**echo**) and a parameter to be echoed (**hello world**).

What happens when you enter the command **script1** at the shell prompt \$?

You may get a command not found error because your current directory is not in the command search path of the shell. **\$PATH** is the name of the shell's environment variable where the search path is configured. Everytime you type a command at the shell, it searches the directories listed in **\$PATH** in the order in which they are listed. If your current directory is not listed, the shell will not search it for the commands in it. You can review the current path settings with the command:-

```
echo $PATH
```

If you get a command not found error then you need to specify more explicitly the name of the file you wish to run by supplying a full path name or a more qualified relative path name.

Type **./script1** at the prompt.

This tells the shell that you want to run the file **script1** located in the current directory.

So the shell now should have found your file but noticed that you do not have permission to execute the file **script1**. – Permission denied

## 5. File Access Permissions

In Unix a file can have 3 types of access namely read, write and execute (rwx).

If you type the command

```
ls -l
```

The long directory listing option -l will show you the access permissions of each file in your cs240 directory.

**script1** currently has r and w access only for its owner.

Give yourself execute permission on the file **script1** by typing

```
chmod +x script1
```

This changes the mode (chmod) in which you and your processes can access the file.

Type the command `ls -l` again to see the difference to the permissions.

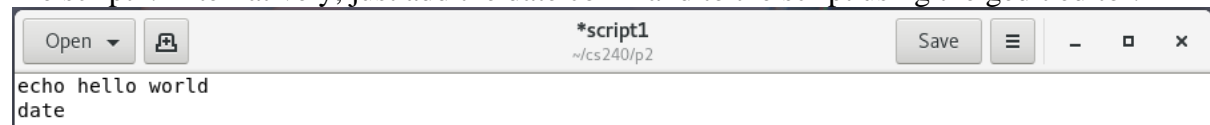
Now that you have execute permission, run it in the current shell by typing

```
./script1
```

The addition of further commands to the script will not cause the execute permission of the file script1 to change.

Add the **date** command to the file **script1** by typing `echo 'date' >> script1`

The >> symbol indicates to the shell that output of the echo command is to be **appended** to file script1. Alternatively, just add the date command to the script using the gedit editor.



Note, when your editor shows a \* next to the name of the file script1 it means you haven't saved the file.

Type `cat script1` to see the contents of it, when it has been saved.

Now run **script1** and note the change in output. `./script1`

## 6. Background Processes

The ampersand character & can be used to tell the shell to run commands concurrently with the shell program (i.e. not to wait until child processes terminate before accepting new commands). The commands execute in the background and status information about background processes (e.g. initialisation and termination) is passed to the parent shell and displayed on its current output stream. This allows us to have a number of programs running at once, and we still have the facility to enter Unix commands at the original shell prompt while they are running.

Open two new Xterminals by entering the following command in your existing terminal window:-

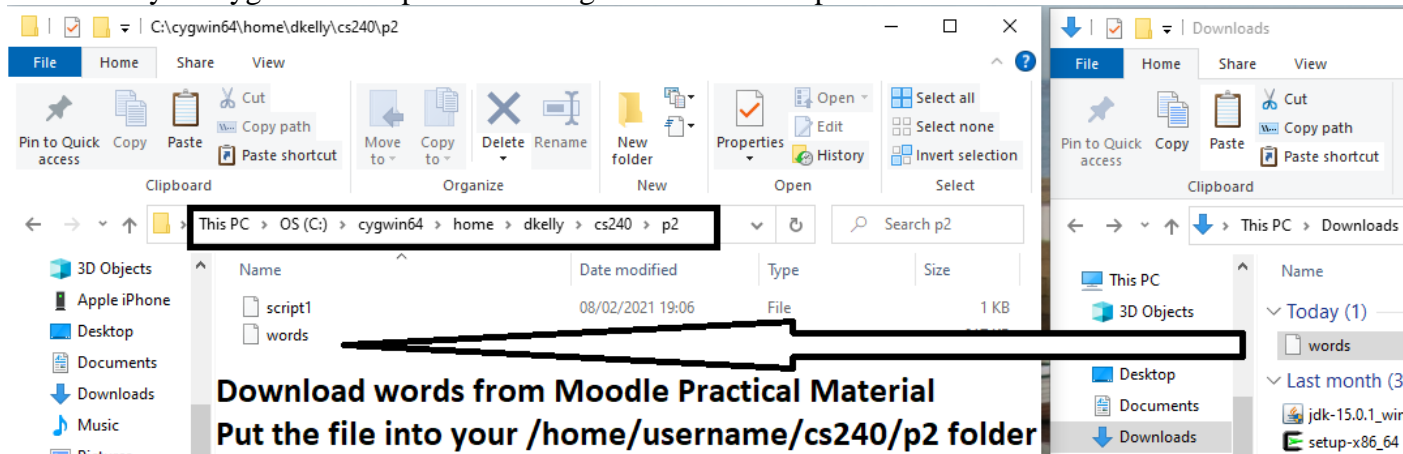
```
xterm & a new terminal window appears
```

```
xterm & a new terminal window appears
```

Note that the original bash shell prompt remains for us to continue launching other processes in the original window. Now close the original Xterm window by clicking the x in top right. Notice how by killing that bash shell, it also resulted in the child xterminals also being killed.

## 7. Install the words dictionary file

In order to do the next exercise, we need to download the words file from the moodle Practical Material folder. words is a standard text file on unix systems which simply consists of a list of dictionary words, one per line. Move words from your web browser downloads folder into your cygwin cs240/p2 folder using Windows file explorer as shown.



## 8. Using the grep string pattern matching utility

We are going to use the Unix grep utility to create a more complex shell script. The grep program searches for text patterns in named files or in the standard input if no files are given. The text patterns to match are indicated by regular expression arguments to grep. Without options, grep prints each line in the input stream that contains a text pattern that matches the given expression. Various options can alter this behaviour.

As an example, get a process listing use the command:-

```
ps
```

Now get a process listing showing only lines containing the text `pty0` with the command:-

```
ps | grep pty0
```

Make `/home/username/cs240` your current directory. The output of the second command below will depend on whether you correctly followed the instructions of last week's practical. Substitute your own username in place of `username` below.

What is the output of `grep quick /home/username/cs240/p2/words`

What is the output of `grep file p1/text/*`

```
quickness's
quicksand
quicksand's
quicksands
quicksilver
quicksilver's

dkelly@MaryEllen ~/cs240
$ grep file p1/text/*
p1/text/file1:This is file 1
p1/text/file2:This is file 2
p1/text/file3:This is file 3
p1/text/file4:This is file 4

dkelly@MaryEllen ~/cs240
$
```

The first parameter to `grep` is the regular expression or string pattern to be matched and the second parameter is the path to the file words, that is to be searched for that pattern.

Some elements which may be included in the regular expression are defined to have the following meaning: (They are just included here for later reference.)

^      beginning of a line  
\$      end of a line  
.      match any single character

So for example if you wanted to find all 6 letter words beginning with `b` at the start of the line and ending with `n` with nothing following on the line you would use the command:-

```
grep ^b....n$ /home/username/cs240/p2/words
```

Try it, using your own username.

Some valid flag options to the **grep** command are listed below. Use the **man** command for entire list.

-c      Produce a count of lines rather than the lines themselves.  
-l      List the names of the files that contain text patterns matching the expression.  
-n      Print the line number (from the input file) for each match.  
-v      Print lines that don't contain text matching the expression rather than lines that do.  
-i      Ignore case distinction in the search string

```
grep -c ^b....n$ /home/username/cs240/p2/words
```

gives 28 as the number of 6 letter words beginning with `b` and ending with `n`

## **9. I/O Redirection**

Make `p2` your current directory `cd /home/username/cs240/p2`

One of the main features of the Unix environment is the ease with which input/output to and from commands/processes may be redirected. The output from any command executed on the command line can simply be redirected to a file by placing `> filename` after the command. The following will put all words beginning with `q` into a file called `temp` instead of displaying them on the screen.

```
grep -i ^q /home/username/cs240/p2/words > temp
```

The `-i` flag tells `grep` to ignore upper/lowercase distinctions, the output of `grep` is to be redirected into a file called `temp`.

```
cat temp                      Display contents of temp to standard output.
```

## **10. Formatting and Controlling Output for User**

The **pr** utility is used to paginate or columnate files for printing.

```
pr -4 temp                      Print the lines of file temp in four columns to standard output
```

Try it. Use the mousewheel to scroll back the Xterm window to see three pages worth of words.

Notice a lot of blank lines at the end of the last output page which were generated by **pr** during processing.

## **11. Connecting processes using Pipes**

A pipe is a one way communication channel between two processes. The shell pipe symbol '|' indicates that the output stream of one command is to be used as an input stream to another command.

**more** is a utility that reads from a single input stream (e.g a file) and writes to a single output stream (e.g. the shell window). It only writes one screen at a time and then waits for the space bar to be pressed before displaying the next screen. Press 'q' to exit **more** at any time.

We join the output of the print in 4 columns utility to the input of the more utility with the command:-

```
pr -4 temp | more
```

We can now read temp easily in four columns.  
Press the space bar to move to the next page of output from this command.  
Notice the blank lines at the end of the output which were generated by **pr** during processing.

### **How could these blank lines be eliminated from the output of pr?**

Given that the regular expression for a blank line is ^\$, you could pipe the output from **pr** into **grep** to eliminate the blank lines and then pipe the resulting output into **more** using the command below:-

```
pr -4 temp | grep -v ^$ | more
```

The -v option tells grep to not print any lines that match the blank line expression.

## **12. Exercise**

Derive a command (using grep) to search the user word dictionary (/home/username/cs240/p2/words) for all words ending with the string 'hell'.  
Test your command and then, using an editor place the command (not the output) into a file called script2. Save the file in directory /home/username/cs240/p2  
Make script2 executable using `chmod +x script2`

Then execute the command  
`./script2 > scriptout`

The list of words ending with 'hell' is put into a file called scriptout in our current directory p2.

When done type  
`cat scriptout`  
to see the results.

### 13. Arguments to Shell Scripts

We can pass arguments to our shell scripts by using the shell variables \$1, \$2, ..., \$9 within our script. These variables are replaced by string arguments given to a shell script on the command line when it is invoked.

For example, using gedit create a file called `script3` in your **p2 directory** containing the text `echo $1`

Make `script3` executable  
`chmod +x script3`

Run it by typing `./script3 hello`

Now type `./script3 there`

Explain what you observe?

Whatever string value we give on the command line to `script3` is passed into the script as the value of variable \$1. So the command `echo $1` prints the value of the command line argument given to the script.

### 14. Script Programming Problem 1

The **du** utility (disk usage) gives the number of kilobytes contained in all files and directories in the specified path and operates recursively on directories contained within that path. See the online manual for details. It will provide a line of output containing the file name and other information for each file it comes across in the given search path.

Try `du -a *`

The parameter `*` indicates all files in the current directory and is the default. What does the `-a` flag do (look in the manual)?

```
dkelly@MaryEllen ~/cs240/p2
$ echo 'du -a /home | grep $1' >filefind

dkelly@MaryEllen ~/cs240/p2
$ chmod +x filefind

dkelly@MaryEllen ~/cs240/p2
$ ./filefind scr
0      /home/dkelly/.local/share/nautilus/scripts
1      /home/dkelly/cs240/p2/script1
1      /home/dkelly/cs240/p2/script2
1      /home/dkelly/cs240/p2/script3
1      /home/dkelly/cs240/p2/scriptout
1      /home/dkelly/p2/script1

dkelly@MaryEllen ~/cs240/p2
$ █
```

Use the **du** and **grep** utilities, pipes and shell variables to write a script called **filefind** which will search for a given file (supplied as a parameter and accessible as \$1 within the script) starting at the home directory. Use **du** to traverse the filesystem from the top of the home directory “`du -a /home`” and then pipe its input into **grep** to search for a given filename string and only output lines that contain that string.

Make **filefind** an executable script. When done try to find all files containing ‘scr’ in their filename. `./filefind scr`

So now you have made a new command by composing simpler commands together. This was the objective of the practical, to show you the power and flexibility of Unix I/O and to show how the command interpreter interface can sometimes be more powerful than a GUI.



## **15 Script Programming Problem 2**

Write a script called **xword** which scans the dictionary file to match a word outline (as in hangman style) and generates all matches of words of equal length to the search string and then prints the output to the screen in the number of

```
dkelly@MaryEllen ~/cs240/p2
$ echo 'grep ^$1$ /home/dkelly/cs240/p2/words | pr -2' >xword

dkelly@MaryEllen ~/cs240/p2
$ chmod +x xword

dkelly@MaryEllen ~/cs240/p2
$ ./xword b..m.. 5
```

2021-02-08 21:01

Page 1

balm's	berm's	blames	boomed	brim's
barman	bitmap	blimps	bowman	bummed
beam's	blamed	boom's	bowmen	bummer
beamed	blamer			

columns specified, without any blank lines. This is a useful script for solving crosswords.

Example: xword b..m.. 5 should give

2021-02-08 20:58

Page 1

balm's	berm's	blames	boomed	brim's
barman	bitmap	blimps	bowman	bummed
beam's	blamed	boom's	bowmen	bummer
beamed	blamer			

Try

```
xword b..... 5
```

The words should appear in five columns occupying about 3 pages.

## **16. Summary of things you should be able to understand and do at the end of this lab:-**

Know the advantage of a command line interface over a GUI

See what processes are running and how to terminate them

Understand how to run processes concurrently (in the background) with the shell using &

Understand how the shell finds commands in the filesystem using \$PATH

Understand the purpose of the grep, pr, more and cat command/utilities

Know how to create a shell script and make it executable and how to pass arguments to the script

Know how to redirect process input and output streams and how to create applications by composition of processes, i.e. joining processes with pipes

Be able to apply the above to write the shell scripts which solve the two problems at the end

## **17. What to submit on moodle**

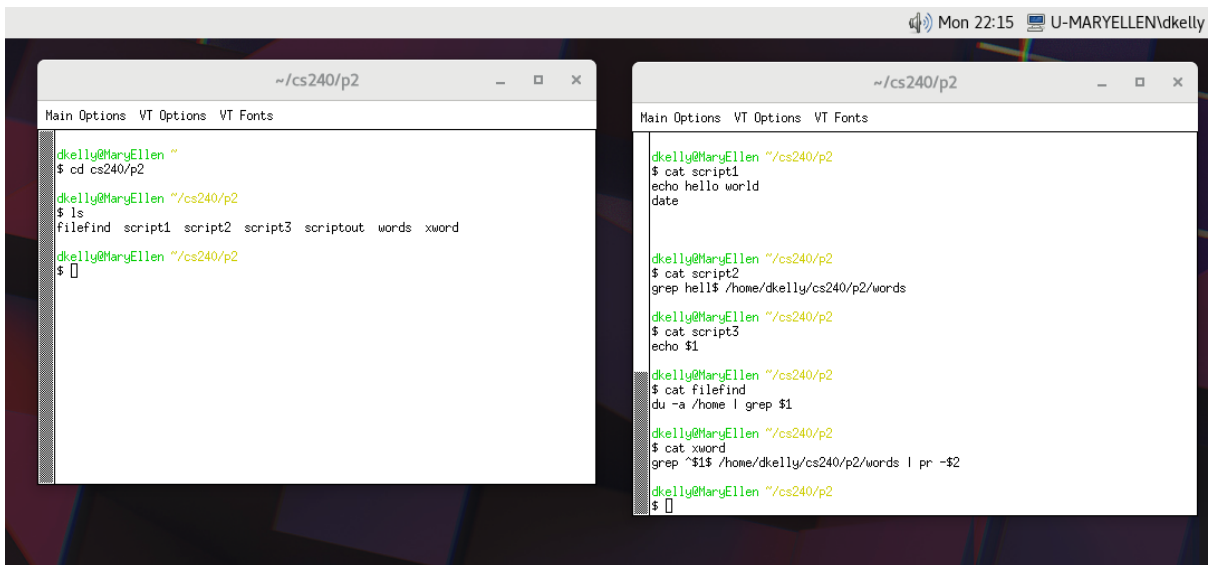
A single screenshot of your Gnome desktop showing two Xterminal windows. Save the screenshot as a fullsized .png file using Paint or equivalent and submit on moodle.

One window showing the list of files in /home/username/cs240/p2

The other showing the contents of the five scripts you created during this practical.

Your username in the top right.

It must be readable.



```
Mon 22:15 U-MARYELLEN\dkelly

~/cs240/p2
Main Options VT Options VT Fonts
dkelly@MaryEllen ~
$ cd cs240/p2
dkelly@MaryEllen ~/cs240/p2
$ ls
filefind script1 script2 script3 scriptout words xword
dkelly@MaryEllen ~/cs240/p2
$

~/cs240/p2
Main Options VT Options VT Fonts
dkelly@MaryEllen ~/cs240/p2
$ cat script1
echo hello world
date

dkelly@MaryEllen ~/cs240/p2
$ cat script2
grep hell$ /home/dkelly/cs240/p2/words

dkelly@MaryEllen ~/cs240/p2
$ cat script3
echo $1

dkelly@MaryEllen ~/cs240/p2
$ cat filefind
du -a /home | grep $1

dkelly@MaryEllen ~/cs240/p2
$ cat xword
grep ^1$ /home/dkelly/cs240/p2/words | pr -2

dkelly@MaryEllen ~/cs240/p2
$
```