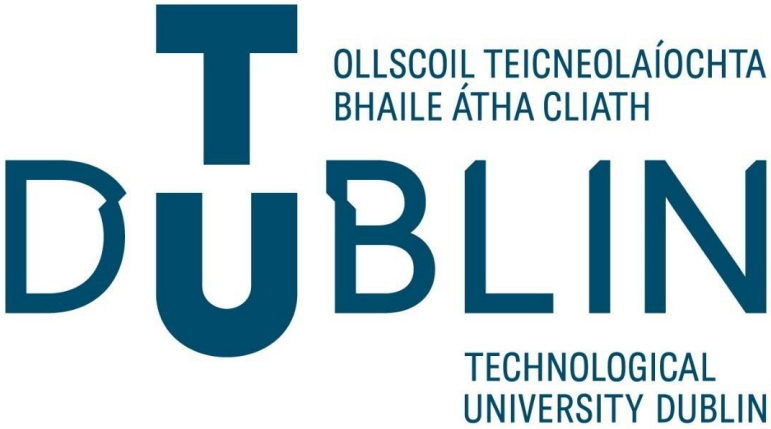


GETTING STARTED WITH THE SHELL

File Management

Create a directory in your code space called file-management.

Create a file called alice.txt in this folder.

Modify the permissions of this file that will give read write and execute permissions to everyone.

Run this command on the terminal and take a screenshot for your codespace github blog.

Now, modify the permissions of this file to give read/write permissions to the user, read permissions to the group and no permission to outside.

Run this command on the terminal and take a screenshot for your codespace github blog.

Create a directory inside file-management called level-1, create a file called alice-level1.txt inside this folder.

Next create another directory called level-2 inside level-1, create a file called alice-level2.txt

The directory structure should look like the following:

./file-management/

./level-1/alice-level1.txt

./level-2/alice-level2.txt

Go to your home director 'cd ~/'

What is the chmod command you need to use to modify the permissions to be read and write for everybody, for every file in file-management directory.

Note – you will need to use one command (use the man page and search for recursive)

Run this command on the terminal and take a screenshot for your codespace github blog

Using the Man pages

Every command you can run in the terminal comes with a built-in manual. If you want to find out more about how a command works just type

```
$ man <command>
```

(As usual, you need to remove the angular brackets from this command)

Create the following directory hierarchy in your home folder

`lab/sysDetails/reports`

We need to create 3 folders above, we could do this one at a time, making each directory, entering it, and then making the next, but the `mkdir` command allows us to make all directories in one go. Use the `man` page to find out how to do this (you are looking for the --**parents** option). Add the command to your github repository.

Environment variables

When you open your shell (BASH), it creates a bunch of variables containing information about your current user, your system, and things like the current directory of the shell. These variables can be used by you, or by the programmes you run to find out more information about its environment, therefore these are known as **environment variables**.

Environment variables in bash are uppercase names beginning with a **$ sign**. If we want to check the value of an environment variable, we use the `echo` command. The echo command will echo back

whatever we give it as input. The `$OSTYPE` environment variable contains basic information on the type of operating system you are running. Find out its value using echo

```
$ echo $OSTYPE
```

Use the echo command to find out the current version of **BASH** you are running (this uses the `$BASH_VERSION` environment variable).

Add a screenshot showing your $BASH_VERSION to your github repository

Find out which environment variable contains your user ID, and use echo to figure out the ID for your current user. Unlike your username, userID is numeric and is used by the system to identify you in logs etc.

Use a text editor (nano or vi) to create a file inside your newly-created reports folder called *bash_version.txt*. The file should contain the value of the $BASH_VERSION environment variable. Ensure it worked by outputting the contents of the file

```
$ cat bash_version.txt
```

If you are new to working on the command line, I suggest using nano; there is a guide on how to use nano available here.

Checking free memory

There is a saying in Linux that *everything is a file*. If you want to print out some text you save it to a special file that sends it on to the printer. If you want to send some text over the internet you write it to a special file that sends it out along the network. If you want to read some text coming in from the network you read it from a special file. This makes a programmer's life easy because if they know how to read from and write to a file, they can do pretty much anything they want!

On Windows, if you want to check how much RAM is available you need to open the Task Manager and check the Memory tab. On Linux, you can look it up in a special *file*. The /proc/meminfo file contains all the information you could ever need about system RAM. Read from that file using the cat command.

What is the total RAM allocated to the system?

How much of that RAM is available right now?

Watching a command in real-time

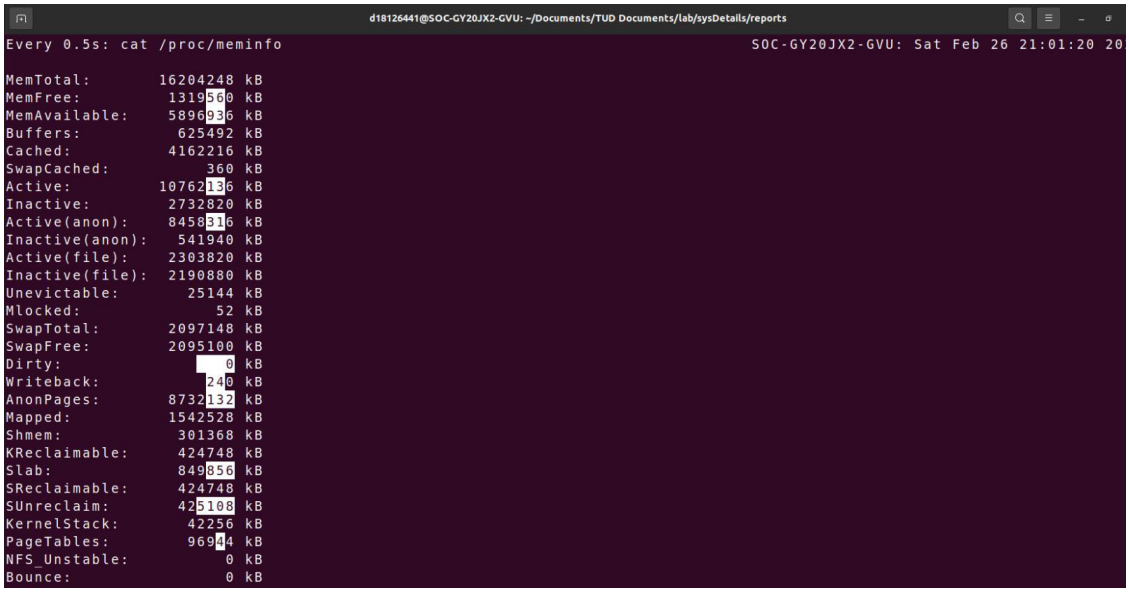
When we read from the `/proc/meminfo` file we get the information for the RAM at a snapshot in time. Often, we would like to monitor how the memory usage changes. Linux provides a watch command which will take any command as its input, and run it every 2 seconds, this will allow us to watch it change in real time.

```
$ watch -option1 -option2 <command>
```

We want to specify 2 options to the watch command. The first option lets us highlight the differences on the screen, this makes it easier to see values changing. The second option lets us set the interval. 2 seconds is a long time to wait. We want to make it wait 0.5 seconds instead. The command you want to watch is

```
cat /proc/meminfo
```

When you run the command, you should see something like this



While the watch command is running, we cannot do anything else in the terminal so click on the terminal menu at the top menu bar and open a new window. In our new terminal we are going to change into the directory we created for this lab and generate a random file.

Generating large files

The `dd` command lets us create files. In the command below, we are copying data from the input file (if) **/dev/urandom** (a *file* which gives us random bytes) and writing it to the output file (of) specified. We are using a block size (bs) of 1MB and writing 1000 blocks count. In short, the command below is generating a 1GB file consisting of random bytes

```
$ dd if=/dev/urandom of=random.bytes bs=1M count=1000
```

This command should take a while to complete. Keep an eye on the `watch` window of the terminal. Roughly how much memory did this command use?

When you are done, delete the file using the `rm` command (use the man command to look this up).

You can stop the watch command by clicking on the terminal window and pressing **ctrl+c**

Redirecting output

By default, the terminal will print its output for us to read. Sometimes, rather than having it printed out to the terminal, we would rather save it to a file. This is known as **redirecting output**. The **>** operator allows us to redirect the output of a command to a file.

We are going to create a file called *helloWorld.txt* and watch what happens to it as we use the redirection operator. First create an empty file using **`touch`**

```
$ touch helloWorld.txt
```

Then use the watch command like we did previously to continuously check the contents.

Run the following command, what happens to the file?

```
$ echo "Hello World" > helloWorld.txt
```

What happens if we do it again?

```
$ echo "Goodbye World" > helloWorld.txt
```

Check the contents of helloWorld.txt after running this command. State on your github repository what happens if you redirect to an existing file using >

We can also use the >> operator to redirect output. What is the difference between these two operators? Try it again to see if you can figure it out.

Creating the system report file

Finally, we are going to use everything we have learned to create a single system report file containing information on both the memory and CPU usage at a given point in time.

Use `cat` to output the contents of `/proc/meminfo` to a file called *systemStats.txt*

Use the `top` command to append the CPU usage to this file, make sure you do not overwrite the RAM details when you do this.

The top command will run in an infinite loop by default (like how we used watch earlier). This is useful when you want to watch the values, but makes it impossible to output to a file because the command never ends. We need to specify **batch mode** with n=1 to make sure that we can redirect the output

```
$ top -b -n 1
```

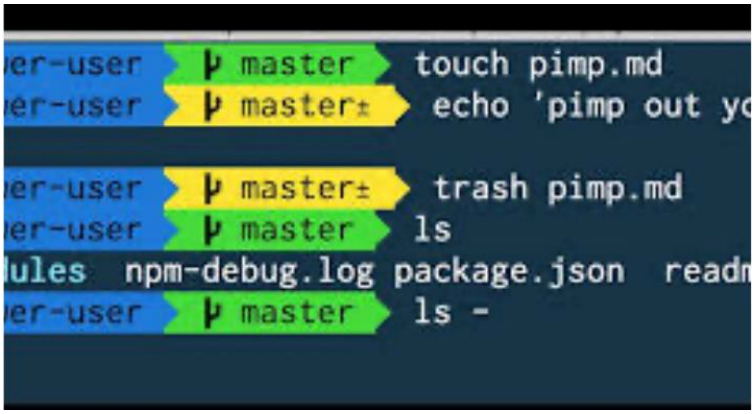
Finally, output the version of bash used to run these commands to the bottom of the file (it is an environment variable; you can use the `echo` command).

Add *systemStats.txt* as a downloadable file to your github repository.

Customizing the prompt

Now you know about environment variables you can customize your terminal.

Linux provides lots of options to change how your shell looks, you can use colour and personalise the message if you like. The shell prompt message is stored in the **$PS1** environment variable. The tutorial below will guide you through personalising your prompt.



https://www.howtogeek.com/307701/how-to-customize-and-colorize-your-bash-prompt/