Linux Lab: Exercises

Cody Harrington
CompSoc

May 23, 2013

1 Getting started

1.1 Logging in

Before you log in, you are able to choose which desktop environment you want to use. The default desktop environment is called Cinnamon.

On another note, you can also choose to log in with one of the virtual consoles by pressing [Ctrl] + [Alt] and one of the keys [F2] to [F6]. This will give you just a command line interface with no graphical windows; press [Ctrl] + [Alt] + [F1] to return back to the X Window system. You can be logged in multiple times this way.

You will need access to the terminal for the majority of this lab. In Cinnamon, you can find it down on the taskbar as a small black box with a couple of white characters on it.

1.2 Changing your password

You can change your password by using the `passwd` command. It will ask you for your current password, and will then ask you to enter a new password twice. Note: this *only* changes the password for your Linux account. To change passwords for both Windows and Linux, you'll need to go to `https://ras.canterbury.ac.nz`.

1.3 Getting acquainted with the University system

Your personal files can be found in your home directory, which is located in /home/cosc/student/*usercode*. You can see this by using the `pwd` command, which means "print working directory", just be careful not to confuse `pwd` with `passwd`. For convenience, you don't have to type out this path all of the time to get back to your home directory; you can use ˜or $HOME instead, so `cd ~` or `cd $HOME` will take you back to your home directory.

2 Terminal basics

Firstly, we're going to get acquainted with moving around the system.

You can use the `ls` command to list the files and other directories in the directory you are in. `ls` has many different options you can pass to it; for example `ls -l` gives you a more detailed listing, while `ls -a` will tell show hidden files. Take a look at the Unix Programmer's Manual (a.k.a manpages) with the `man` command, i.e. `man ls` for the man page on the `ls` command.

You use `cd` to change to a different directory, and `cd ..` to go up one directory (i.e. if you are in /home/cosc/student, `cd ..` will take you back up into /home/cosc). You can also chain this to go several levels up, so `cd ../../..` will send you up 3 directories for example. Just like .. means the parent directory one level above, you can also use . as the current directory, so `ls .` will print the contents of the directory you are in.

Exercises

1. Make sure you are in your home directory.

2. We want to create a new folder to work within, then change to that directory. The `mkdir` command will do this.

   ```
   cosc937:~$ mkdir linuxlab
   cosc937:~$ cd linuxlab
   cosc937:linuxlab$
   ```

3. We will write a simple sentence to a file using `echo` and redirection.

   ```
   cosc937:linuxlab$ echo "This is a simple sentence">ex1
   ```

   Note that we are running the command `echo "This is a simple sentence"`, and telling the terminal that we want store the output of this command in a file called ex1 with the redirection operator >. Note that you also don't need a file extension such as .txt like on Windows. So that should now show up if you run `ls`:

   ```
   cosc937:linuxlab$ ls
   ex1
   ```

   And we can see that it says our sentence

   ```
   cosc937:linuxlab$ cat ex1
   This is a simple sentence
   ```

4. Now we're going to write random data into a file using /dev/urandom, a special device file that the OS generates random data into. Use the `head` command to read the start of a file, and the double-arrow redirection >> to add this data to the end of the previous file we created. We want 10 bytes. *Hint: check the man pages*

   If you done it correctly, when you `cat` the file like before, you should see some strange characters on the end.

5. This time we want to read the words "simple sentence" from the file, only by using `head`, a pipe and the `tail` command, which reads from the end of a file.

   Firstly we're going to see how many bytes the file is using `wc`, and we can subtract 10 from this since we added 10 extra bytes of junk on the end.

   ```
   cosc937:linuxlab$ wc -c ex1
   36 ex1
   ```

   Taking away 10 makes 26 bytes. I should point out that a newline character is automatically appended to the end of a sentence when you echo it into a file, so for the full sentence without the newline character, that is only 25 bytes.

   Now we use the pipe operator | to take the output of `head` and put it into the `tail` command. A character is 1 byte in size (some extended character sets have bigger character sizes, but for now lets assume they're only one byte). There are 15 characters in "simple sentence", so we want to read the last 15 bytes of the first 25 bytes of the file. If you run it correctly, your output should look like this:

   ```
   simple sentencecosc937:linuxlab$
   ```

And, to read in the newline character as well, we simply read 26 bytes off the front, and 16 from the end of that.

```
simple sentence
cosc937:linuxlab$
```

To review the commands used so far:

```
man # View the manual pages
passwd # Change your password
pwd # Print working directory
ls # List contents of directory
cd # Change directory
mkdir # Make directory
echo # Output text input
cat # Display the contents of a file
head # Read the start of a file
tail # Read the end of a file
wc # Word count
```

3 Scripting

I have written a multitude of scripts to help make using Linux easier. My scripts involve enabling the internet on a machine or setting file permissions. We need to create a scripts folder and add it to our $PATH variable so that when we type a script in to the terminal, it runs.

You can enable internet on your computer by going to `http://ienabler.canterbury.ac.nz:259` or by using `telnet` to connect to ienabler.canterbury.ac.nz on port 259.

Exercises

1. First we want to create a folder to store our scripts in
2. Next we want to create a file containing the line `telnet ienabler.canterbury.ac.nz 259` named 'ie'.
3. We need to make sure that our file is executable:

   ```
   cosc12019:scripts$ chmod +x ie
   ```

   `chmod` changes the mode of a file, and the +x means we want to add executable permissions for us only and no other user.
4. Lets take a look at our $PATH environment variable. $PATH is a list of paths, in order, separated by colons, that the OS checks for a filename matching the command you entered into the terminal.

   We'll use the command `export` to store the path to our scripts folder in a variable, and then prepend this variable to $PATH. You can prepend to the $PATH with the command `export PATH=$SCRIPTS:$PATH`.

   We should now be able to `cd` to the scripts directory, or run a script from anywhere.

   ```
   cosc937:~$ ie
   Trying 132.181.3.230...
   Connected to ienabler.canterbury.ac.nz.
   Escape character is '^]'.
   Check Point FireWall-1 Client Authentication Server running on ucfw1
   User:
   ```

5. However, this will only work while this terminal is open. As soon as we close it, it will be reset. Instead, we can add these lines to a special startup script called `.profile` so when we log on, our script folder will be added to the $PATH instantly. This is located in the home directory.

   So, add the `export` lines which modified you path, to the end of the file `~/.profile`

6. Voila! You should be able to run any file that is in your scripts folder and set to executable, from any location in the terminal. You should also be able to use the 'ie' script to enable internet on all ports on the computer.

To review the commands used so far:

```
telnet # An old, outdated, insecure way of connecting remotely
chmod # Change mode-changes file permissions
export # Save an environment variable
```

4 The COSC webserver

There is a web daemon constantly running which will run webpage(s) for you if you have it set up correctly. Here's how we can set it up.

Exercises

1. This web daemon looks for a directory called `public_html` in your home directory. Create it.
2. Create a simple HTML file inside `public_html` called `index.html`
3. To be able to view your webpage, you need to set read and execute permissions for other on the `public_html` directory and all of its contents.
4. You can view the page at `http://studweb.cosc.canterbury.ac.nz/~usercode/index.html` where `usercode` is replaced with your own usercode.

5 Mail from the terminal

The university has a mail server with which, you can use to send email, right from your terminal. You can even attach and send files, set priority, and most other things you can do with sending an email.

```
cssecs1:~$ mail cah143@uclive.ac.nz
Subject: Test mail
This is the message I have typed
```

This is relatively simple. It will ask you for a subject, and when you have given that, you type your message, and press [Ctrl] + [D] to finish it (and automatically send it).

6 Secure shell, secure copy

In this section we will be mainly covering remote terminal connections using `ssh` (Secure SHell). We'll also look at generating SSH keys, and using `scp` (Secure CoPy).

We're going to take a look at the COSC linux server which can be ssh'd to from outside of the university, which has the address `linux.cosc.canterbury.ac.nz`. Note: this address is actually an alias, so when you connect to it, you are instead connected to `cssecs1.canterbury.ac.nz`. You can use either address to connect.

Exercises

1. The syntax of `ssh` is `ssh usercode@address`.

   SSH into the COSC Linux server (the password is your password on the university machines). When you type in your password, you'll see that no typing comes up on screen at all–not even '*' characters. This is because standard output (stdout) is redirected to special device file /dev/null as a security feature, which is a file that contains nothing which acts as like a *bit-bucket*[1] or black hole for when you just need to dispose of some data.

2. Once you've ssh'd in, you'll notice that the computer is cssecs1 instead of cosc10060. Now we will generate our SSH keys (you can skip this step if you're already generated them). Run `ssh-keygen` and follow the instructions to generate new keys.

3. The syntax of SCP is `scp file usercode@address:dest` to copy to a remote machine, or `scp usercode@address:file_path dest` to copy from a remote machine.

   Use SCP to copy `~/linuxlab/ex1` to `~/linuxlab/ex2`. Because you're logged in to the one account, this will act like a normal copy, however, it still copies across the network.

4. Type `exit` to close the session once you're done.

That covers connecting to the linux.cosc server, and generating SSH keys. You can use SSH and SCP to work on or with files on the uni machines, from home. Just `ssh username@linux.cosc.canterbury.ac.nz` as mentioned before.

This time we're going to have a little playful fun by messing up a person's terminal while they're using it.

Exercises

1. Firstly, we want to write a little script to disable messaging so we have some way of defending ourselves against this kind of heckle. Save `mesg n` to a file somewhere and make it executable. We'll use it later.

2. SSH into a machine that someone else is using. A computer's address is formatted `cosc$NUMBER.canterbury.ac.nz`, where you replace $NUMBER with the number on the sticker which should be stuck on the front of the computer case. For example, I'm on computer 10060, and I want to connect to computer 10038, so I'd ssh to cosc10038.canterbury.ac.nz.

   Note: We don't want to do this on the linux.cosc.canterbury.ac.nz server because that will disrupt staff, and one day you might come back to find you can't log in anymore :(.

3. To display who is logged on, we use `who` or `w`

   ```
   cosc10038:~$ w
    15:24:53 up 29 min,  4 users,  load average: 1.26, 0.86, 0.51
   USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
   abc123   :0        14:56   ?xdm?   2:52   0.27s /usr/bin/gnome-session --sessio
   abc123   pts/0     14:57   27:43   0.00s  0.00s /bin/bash
   abc123   pts/1     14:57    3:53   0.02s  0.02s bash
   cah143   pts/2     15:24   11.00s  0.09s  0.00s w
   ```

   We can see that I am logged on, on terminal pts/2, while some other user (whose usercode I have replaced with abc123) is logged on pts/0, pts/1 and :0, which is the local address of the main Xorg server display.

[1] http://www.catb.org/jargon/html/B/bit-bucket.html

4. We can use the `finger` command to find out what name belongs to a usercode. If we take a look at the output on my usercode:

   ```
   cosc10038:~$ finger cah143
   Login: cah143          Name: Cody Alan Harrington
   Directory: /home/cosc/student/cah143 Shell: /bin/bash
   On since Wed May 15 15:24 (NZST) on pts/2 from cosc10060.cosc.canterbury.ac.nz
      13 seconds idle
        (messages off)
   Mail forwarded to cah143@uclive.ac.nz
   No mail.
   No Plan.
   cosc10038:~$
   ```

5. We are going to use the `wall` command (write all) to send text to all terminal sessions. Run `wall`, type whatever message you like, and press [Ctrl] + [D] to send it. As soon as you press [Ctrl] + [D], it will send the message to all terminals, including your own. Note: This does not work if they don't have a terminal window open.

6. The more astute of you may have realised that you could pipe data into `wall`, and that there is a special device file called /dev/urandom that generates a constant stream of random data, and there is a program `cat` that outputs the contents of a file. Doing this will spew random junk into all terminals!

7. However, if you gave that a try, you will have noticed there are a couple of problems with this approach:

   - It only sends a finite amount because of a set maximum message size
   - If you're bombarding all terminal sessions, then you're bombarding yourself too.

   There is a `write` command which will allow you to send a message to an individual user (see `man write`). Brilliant! This will stop your own terminal from being bombarded with junk! But there is still the problem with the maximum message size.

   Using the `while` command in BASH, use `write` instead of `wall` above, and find a way to generate a constant stream of messages to the terminal. Is there a way you could prevent the process from being hung up (terminated)?

8. That file we created earlier. Our savior. The command in it, `mesg n` disables messaging to you, which, by running it, (should) prevent people from harassing you with junk.

To review the commands used so far:

```
ssh # Secure shell. Connect to a remote machine.
scp # Secure copy. Copy a file via SSH
ssh-keygen # Generate new SSH keys
exit # Close the terminal session
mesg # Enable/disable messaging
who # Display logged on users
w # Display logged on users
finger # Get information on a user
wall # Write all. Write a message to all terminal sessions
write # Write a message to an individual user
```

7 Python, Java, C and Bash

It's pretty important to know how to build/run your code from the terminal, so here's a section dedicated to it. I've written a set of Fizz buzz[2] programs in Python, Java, C and BASH to play around with running/compiling code. You can take a look at the source code for any of these files using the text editor `nano`.

Exercises

1. We need to get hold of those files. Go back to the 'linuxlab' directory we made earlier.
2. `wget` is a command used to download files from the web. Use this to fetch the files, located at `http://studweb.cosc.canterbury.ac.nz/~cah143/FizzBuzz.tar.gz`
3. Now that we've fetched it, we need to extract the files. This type of file with the extension `.tar.gz` is called a 'tarball', uses two programs `tar` (store files into one archive) and `gzip` (compress a file or files), to create it. To extract our files, we can simply use `tar`. Use the manpage for `tar` to extract the files.
4. The Python interpreter allows you to not only run your Python files, but also start up a session of the interpreter and even just run single Python statements, redirecting to stdout. Running `FizzBuzz.py` is easy enough:

   ```
   cosc10020:linuxlab$ python FizzBuzz.py
   ```

   Starting the Python interpreter is even easier:

   ```
   cosc10020:linuxlab$ python
   Python 2.7.3 (default, Aug  9 2012, 17:23:57)
   [GCC 4.7.1 20120720 (Red Hat 4.7.1-5)] on linux2
   Type "help", "copyright", "credits" or "license" for more information.
   >>> print("Hello world!")
   Hello world!
   >>>
   ```

   and [Ctrl] + [D] will exit. Note: By default, this brings up Python 2.7. You can get to Python 3 by typing `python3` instead of `python`.

   Finally, to run simple Python commands and have them output to the terminal without entering the interpreter, use the `-c` switch:

   ```
   cosc10020:linuxlab$ python -c "print('Hello world! Frobnicate the foobar')"
   Hello world! Frobnicate the foobar
   cosc10020:linuxlab$
   ```

5. Java is a compiled language, and the resulting compiled code is executed by the Java Virtual Machine (JVM). Don't worry if you don't know any Java; I've written the code, so all you need to do is compile it.

   We can compile our FizzBuzz.java using the Java compiler, `javac`, which will create a file called `FizzBuzz.class`:

[2] http://en.wikipedia.org/wiki/Fizz_buzz

```
cosc10020:linuxlab$ javac FizzBuzz.java
cosc10020:linuxlab$ ls
FizzBuzz.c FizzBuzz.java  FizzBuzz.sh
FizzBuzz.class FizzBuzz.py    FizzBuzz.tar.gz
```

This `FizzBuzz.class` contains the compiled Java code, called bytecode, that the JVM executes. Now we run Java to execute the class:

```
cosc10020:linuxlab$ java FizzBuzz
```

Note that there is no file called `FizzBuzz`. However. the JVM executes the file `FizzBuzz.class`. Try to imagine that the JVM is executing the compiled class as opposed to just executing the file.

When you have multiple .java files, you would compile them into multiple class files, then put them into a .jar file with the command `jar`. This is like `tar`, but for Java. You would then run `java -jar SomeFile.jar` to execute it.

```
# As an example
javac *.java
jar cvf SomeFile.jar *.class
java -jar SomeFile.jar
```

6. Now to compile C code. We use GCC, which can stand for GNU C compiler, or the GNU Compiler Collection. Firstly, the simplest way to compile:

```
cosc10020:linuxlab$ gcc -o FizzBuzz FizzBuzz.c
cosc10020:linuxlab$ ./FizzBuzz
```

This compiles the C file `FizzBuzz.c` into the executable `FizzBuzz`, and then we run the file However, GCC is a lot more powerful than that, and contains warning generators to make sure that your program is written well. *C does exactly what you tell it, and nothing more.* This is how I compile my C programs:

```
gcc -g -ansi -Wall -Wextra -Werror -pedantic -pedantic-errors -o FizzBuzz FizzBuzz.c
```

Here is what these extra compilation flags mean:
`-g`
Add debug data, so you see where your program crashed
`-ansi`
Check that your program is ANSI-C compliant
`-Wall -Wextra -pedantic`
These add more warning checkers
`-Werror -pedantic-errors`
These turn any warnings into errors so that your program does not compile.

Compiling C++ code is pretty much the same. Instead of `gcc` however, you use `g++`.

To review the commands used so far:

```
wget # Get a file from a web address
python # Begin Python 2.7 interpreter
python3 # Begin Python 3 interpreter
javac # Java compiler
java # Java Virtual Machine
gcc # C compiler
g++ # C++ compiler
```

8 Extra: Mounting your Windows P: drive

Note: the exercises in this section cannot be performed on the Linux accounts at UC, because the command, `mount` needs higher privileges. However, this can be done one your laptop while connected to UCWireless.

Firstly, create the directory where you want to mount your P: drive. We'll make a directory under /mnt called win.

The command to mount the drive is:

```
mount -t cifs -o username=$usercode,password=$password,domain=uocnt
//ucfile$i/Homes$j/$usercode /mnt/win
```

Where you replace $usercode with your usercode and$password with your password. Your P: drive is located in either ucfile5, ucfile6 or ucfile7, and in one of those directories, it is in either Homes1 or Homes2. You can find out by booting in Windows and checking the properties of your :P drive, or you could guess by trial and error.

`-t cifs` specifies the file system type, `-o username=$usercode,password=$password,domain=uocnt` specifies the options to pass to the mount, `//ucfile$i/Homes$j/$usercode` specifies the network drive to mount locally, and `/mnt/win` specifies the directory under which to mount it under.

You could also add your details to a file and restrict the permissions to that file:

```
echo username=cah143,password=$password,domain=uocnt > deets
chmod og-rwx deets
chmod u-rw deets
mount -t cifs -o $(cat deets) //ucfile$i/Homes$j/$usercode /mnt/win
```

The syntax `$()` replaces `$()` with its output, so `$(cat deets)` would be replaced with the contents of the file `deets`

To unmount, you would simply just need to type `umount /mnt/win`.