Linux Terminal

Naavin Ravinthran
SIG Cybersecurity – Monash University (Malaysia)

1 Meme

Note to self, delete this later and find more appropriate meme to lighten mood.

Figure 1: A humourous mug.

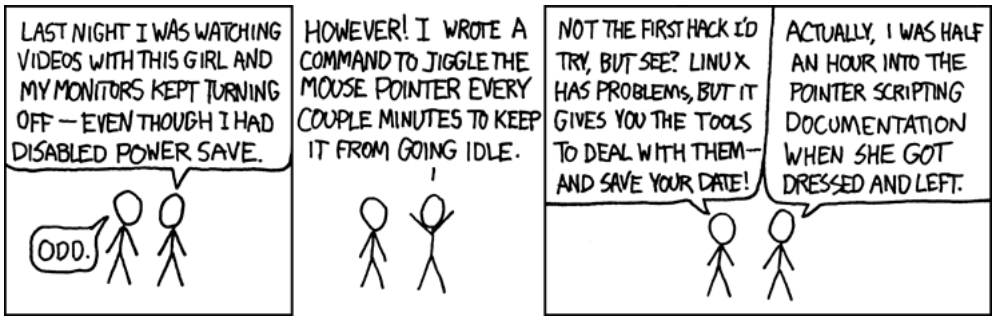

Figure 2: XKCD Comic #196

2 Acknowledgement

A lot of the introductory sections use content from Digitalocean's Introduction to the Linux Terminal[1]. Also of note is Greg's Wiki Bash tutorial.

3 Terminal Emulator

From DigitalOcean, A terminal emulator is a program that allows the use of the terminal in a graphical environment.

4 Shell

From DigitalOcean, In a Linux system, the shell is a command-line interface that interprets a user's commands and script files, and tells the server's operating system what to do with them. There are several shells that are widely used, such as Bourne shell (sh) and C shell (csh). If you're on Windows, I suggest getting Windows Terminal from the Microsoft Store, as well as getting WSL2 (sort of like a Linux virtual machine) installed as well, prefarably something Debian-based like Ubuntu.

5 Navigating

Usually, when you open your terminal, you'll be started off in your home directory. Sometimes your prompt will use a shorthand symbol "~" for your home directory. You can use pwd to print your working directory, i.e: The directory you are currently on.

You can find the File System Hierarchy Standard here[2]. Most of the time, you'll just be working in the "/home/<user>" directory (Similar to the "C:/Users/<username>" directory[3], maybe going into "/mnt" if you have a pen drive, going into "/var/"

[0] XKCD Comic source: `https://xkcd.com/196/`

[1] `https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal`

[2] `https://refspecs.linuxfoundation.org/fhs.shtml`

[3] I generally use the term "directory" over "folder"

for application specific configuration files (The directory "/var/log/" has lots of logs, for example your web server logs for in "/var/log/apache2/access.log", or "/var/www/html" or similar is often the directory for the files in a web server.)

5.1 Listing files and folders

You can list file with the ls command.

```
$ ls
dir1   file1   file2
```

You can view more options with −−help, generally, a lot of programs have this option or something similar to it. Of interest is the "ls −al" command, which gives a slightly more comprehensive listing.

```
$ ls -al
total 4
drwxr-xr-x  3 b1temy b1temy 120 Apr 22 11:10 .
drwxrwxrwt 26 root   root   740 Apr 22 11:10 ..
drwxr-xr-x  2 b1temy b1temy  40 Apr 21 19:39 dir1
-rw-r--r--  1 b1temy b1temy   0 Apr 21 19:39 file1
-rw-r--r--  1 b1temy b1temy   0 Apr 21 19:39 file2
-rw-r--r--  1 b1temy b1temy  30 Apr 22 11:10 .hidden_file
```

The single "." refers to the *current directory*, and two dots refer to the *parent directory*. So if you were in a directory "C:/Users/User/Documents/Homework", its parent directory would be "C:/Users/User/Documents". If you're wondering what the use of a dot for current directory is, I suppose one use I can think of is when you need to pass to a program a directory, and you want to use the current directory that you are in.

The ".hidden_file" is a file that starts with a dot. Files and directories starting with a fullstop are usually hidden by default; it is *not* a security featuere! The purpose is simply not to clutter up your display. Usually used for configuration files/directories of applications.

You can navigate with the cd <directory> (**c**hange **d**irectory). You can "cd .." or "cd dir1" or even specify an absolute directory like cd /home/<user>.

PS: You can also do "ls <directory>" to list files in that directory.

6 Arguments and Environment Variables

You have already seen the arguments from "−−help".

Make a python file using "vim" or "vi", hit the "i" key, and enter the following. Then, when you're done, hit "Escape" on your keyboards, then type ":wq" and hit Return (Enter). [4]

Unfortunately, you'll probably have to write it manually instead of copy pasting.

```
#!/usr/bin/env python
import sys
import os
```

[4] If anyone is interested in an informal Vim workshop, I wouldn't mind.

```
if __name__ == '__main__':
    print(f"Arguments: {sys.argv}")
    print(f"Environment Variables: {os.environ}")
```

You can then run it with python <filename>.txt. Before we look at the output, let's see another way to execute it.

```
$ ls -al file.py
$ chmod +x file.py
$ ls -al file.py
$ ./file.py
```

The reason this works is because of the shebang [5] at the start of the python file, which tells the shell how to execute it. if you've ever wondered why that was there, now you know.[6].

Anyway, you might see something like this:-

```
$ python args_envs.py
Arguments: ['args_envs.py']
Environment Variables: environ({<...>}
```

Ignore the environment varibles for now. Note how the arguments variable is a python list of strings, with the first one being the name of the program. Now try several of these commands.

```
$ python args_envs.py arg1
Arguments: ['args_envs.py', 'arg1']
<...>
$ python args_envs.py arg1 arg2
Arguments: ['args_envs.py', 'arg1', 'arg2']
<...>
$ python args_envs.py "arg1 arg2"
Arguments: ['args_envs.py', 'arg1 arg2']
<...>
```

Note how we added the quotation marks to make it a single string in the list. Python programs (And other languages like C or C++ as well) can use these to figure out the options from the user.[7]

As for the environment variables, you can set one by putting "KEY=VAL " before executing the program. This isn't as commonly done, but it is still sometimes used. Additionally, the PATH environment variable is used so that you can execute programs from other directories (e.g: "/usr/bin").

```
$ TEST_VAR=TEST_VAL python args_envs.py arg1
Arguments: ['args_envs.py', 'arg1']
Environment Variables: environ({<...>, 'TEST_VAR': '
    TEST_VAL'})
```

[5] `https://linuxhandbook.com/shebang/`

[6] Sometimes you might see #!/usr/bin/python instead, which directly tells you to execute it with the python stored in the "/usr/bin/python" path. The reason we use "env" is that running this will use the currently configured python command instead of fixing the one in that directory. If you've used python virtual environments before, that would be one example where you would want to execute the virtual environment's python executable instead of the system version.

[7] You might want to look into the argparse module in the python standard library.

7 Printing out file contents

There are a ton of built-in commands, most of which are either built-into bash (See: "man bash", which is sort of like a terminal-based manual), or are in a directory such as "/usr/bin". One such command is "less". Use it like so.

```
$ less .hidden_file
Hello, this is a hidden file!
```

For longer files, you may need to scroll using the arrow keys. You can hit the "q" key to exit. Some resources online may also point you towards using the "cat" command, which stands for "concatenate". Its meant to be used to concatenating and printing out the contents of multiple files (e.g: "cat file1 file2"), but it can be used with just a single file too.

8 Linux I/O Redirection

Occasionally, you may want to output the output of the file to a file. You can do this by placing "> outputfile.txt" at the end of your command.

9 Command Injection

Consider this program that will take in your name, and print it out.

```
#!/usr/bin/env python
import sys

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print(f"Usage: ./{sys.argv[0]} <your_name>")
    else:
        print(f"Hello there {sys.argv[1]}!")
```

So you can run it like so.

```
$ ./hello.py "My Name"
Hello there My Name!
```

But imagine that we had another program calling this with user input, feeding directly to the terminal, like so.

```
#!/usr/bin/env python
import os
import sys
if __name__ == '__main__':
    if len(sys.argv) > 1:
        os.system(f"python hello.py {sys.argv[1]}")
```

Using any sort of system call is bad practice, and is usually avoided and API calls or a library is used instead. Also, this isn't really a realistic example, but imagine perhaps maybe you're running a PHP server instead that wants to call the python file, and you're rather lazy to figure out how to call a python file from php, and instead using a similar command like "shell_execute".

```
$ python bad.py "test;cat .hidden_file"
Hello there test!
Hello, this is a hidden file!
```

Now, you can see that even though the input was supposed to be just for the name to be passed into for "hello.py", because I put a semicolon, it was able to execute a different command directly after that!

10 Common files of interest

1. /etc/passwd
2. /home/<user>/.bash_history
3. /home/<user>/.bashrc

"/etc/passwd" stores the usernames of the users of the computer, as well as a *hashed+salted* version of their password. There are tools such as John the Ripper to brute-force the passwords and try to crack it.

"bash_history" is simply a history of all the previous commands ran by the user. It is useful for seeing what the last commands of the user were. For example, maybe a developer put his API Key in the environment variable when starting a web service.

API_KEY=344AAB9758BB0D018B93739E7893FB3A node .

And now you have their API key. I have actually seen this file in the Monash provided VMs before, though they didn't really contain anything interesting besides setup for the machine, which is why I guess they left it there.

"bashrc" is executed every starting time you start up your terminal. [8]

Note that for the latter two, they are exclusive to the bash shell. Other shells will have their own files for this.

11 Example Programs

There is a user-contributed list of resources for a ton of different areas of computer science called the awesome list. They have a section on Security tools. They're not all exclusively terminal-based, some use a GUI, some run on Windows, some are even web based, and some are libraries for programming languages. But it is still very likely you'll come across a terminal-only tool at some point or another.

12 Gotchas

12.1 Stopping a program

Ctrl+C

[8] I personally like the use the "fortune" program to give a nice message every time I start up my terminal, as well as perhaps piping it through "cowsay" and/or "lolcat" for fun!

12.2 Special characters

Occasionally, you may want to use some reserved keywords of the shell, like the & operator, or the wildcard ? operator. [9].

Perhaps you found an API[10] like this Cat Facts[11] API that you want to use. After reading the documentation, you decide to test it with the following.

```
$ curl -L https://cat-fact.herokuapp.com/facts/random?
    animal_type=cat&amount=1
[1] 13971
zsh: no matches found: https://cat-fact.herokuapp.com/
    facts/random?animal_type=cat
[1]  + 13971 exit 1     curl -L https://cat-fact.
    herokuapp.com/facts/random?animal_type=cat
```

The "-L" is just to follow redirects. It's a habit of mine to use it. You can find a full list of arguments to use using curl --help all, since it's quite long, you might want to pipe it through less, such as curl --help all | less.

Note if you're using this on bash or another shell, you may get some results, but there will probably also be some errors.

The reason is that there are several reserved keywords being used, such as the & opeartor, and the = operator and the ? operator. You could manually escape it with \&. But you can also just wrap it with quotation marks.

```
$ curl -L "https://cat-fact.herokuapp.com/facts/random?
    animal_type=cat&amount=1"
{"status":{"verified":true,"sentCount":1},"_id":"58
    e008800aac31001185ed07","user":"58
    e007480aac31001185ecef","text":"Wikipedia has a
    recording of a cat meowing, because why not?","__v"
    :0,"source":"user","updatedAt":"2020-08-23T20
    :20:01.611Z","type":"cat","createdAt":"2018-03-06T21
    :20:03.505Z","deleted":false,"used":false}
```

If you're wondering why your terminal didn't end up on a new line, it's because the output of the program didn't include a newline. My shell (zsh) set up my system to handle for this by printing a inverse+bold "%" symbol to indicate that there isn't a newline. But that's not too important.

13 Notes

13.1 Shell scripting

Occasionally you might find some "sh" files, which are files that basically execute the commands you have learnt.

[9] See: "Special Parameters" after running man bash for a full list and description of what it does.

[10] For a list of free public APIs, see `https://github.com/public-apis/public-apis`

[11] `https://alexwohlbruck.github.io/cat-facts/`

13.2 Windows

Windows used to (and still does) have this DOS-like terminal, "cmd". You could do some basic scripting in the ".bat" files, which you might have heard of and were popular in the late 2000s. There are many differences, such as the "dir" command instead of "ls".

A few years ago, Microsoft really started pushing Powershell. While I initially despised it for its weird syntax of commands ("cmdlets") with a "Verb-Noun" naming scheme, and hating the default blue terminal, it does I suppose have its uses. You can learn more about powershell in Microsoft's website here [12].

I have seen some CTFs and some malware hiding in obfuscated Powershell scripts before.

13.3 Macs

I have no tried, but I believe that Macs use the "zsh" shell, and it should be somewhat similar to what was learnt above. Mac is also what is called a "UNIX-like" operating system, which means it conforms to certain specifications, and Linux distros are also UNIX-like. Note that Macs don't come with the GNU core utilities[13] and instead use their own versions, so there might be some differences.

13.4 FreeBSD

FreeBSD and its derivatives also are UNIX-like operating systems, and so it should also be similar. Note that FreeBSD also doesn't come with GNU core utilities.

13.5 Additional Info

There are a ton more commands and programs out there. You can read through the manual "man bash" and/or "info coreutils",

[12] `https://docs.microsoft.com/en-us/learn/modules/introduction-to-powershell/`
[13] Enter: info coreutils to see what this encompasses