Activity 1 - Bash Primer

Daniel Schoonwinkel – E&E Postgraduate Networking Course 2019

April 30, 2018

1 Introduction

BASH (Bourne Again Shell) in some form or other installed in most Linux / Unix distributions. Manipulating files and variables using command line arguments and scripts is a valuable skill, looks great on your CV and potentially save you time and repetitive donkey work. BASH is especially useful in scenarios where you only have ssh remote access and no GUI interface (think servers, datacenters, remote IoT, etc.).

In this part of the networking course we will learn to use many frequently used commands and write a few basic shell scripts.

2 Practical examples

In this section you will see examples of how command can be used. Run all of the commands on your own computer to get an idea of the output and what each command does.

2.1 Basic commands

To get a list of files in a directory, get a Terminal (default Ctrl+Alt+T on Ubuntu), type

```
ls
```

and Enter. What do you see?

To get information about how to use a command, type the command and –help (or -h for some commands), like so:

```
ls −−help
```

This gives usage instructions and other commandline arguments that can be used after the command. For example:

```
ls −a −l
```

The above command can also be written as:

```
ls −al
```

To get even more information about how a specific command works type:

```
man ls
```

This brings up the manual of given command. Use arrow keys to scroll up and down. Type q to quit.

Try all of these command on your own:

```
ls, find, uname −a, ps −e, df, ifconfig, history
```

2.1.1 File structure

In the Unix file structure, all entries are treated as files (including folders). This could sound strange, but it actually simplifies handing folders and files, because they are treated the same. Furthermore, useful system runtime values can also be stored as files (more on that later).

Navigating the file system is achieved with the change directory (cd) command. Run the command:

```
cd ∼/
```

This will change the directory to the current user's home directory. The ∼ symbol represents the home directory.

Run the following command:

```
pwd
```

What do you see? The output is the **absolute path** to the current user's home directory. You can do a `ls` to get the contents of the directory.

Each `/` represents another folder away from the root of the directory structure. Doing this:

```
cd /
```

will take you to the root (somewhat similar to the C:
directory of Windows.) Doing a `pwd` will also confirm that you are in the highest directory. All **absolute paths** are calculated from here.

Doing a

```
cd /home/yourusername
```

will use the **absolute path** to get to your home directory.

Because absolute paths can become quite cumbersome, we more frequently use relative paths, for example (go back to root (`/` directory first)

```
cd home/
cd yourusername/
pwd
```

Note how both `cd` commands do not have a `/` before them. More useful relative path symbols are the following:

```
cd .
cd ..
```

The first refers to the current directory and the second refers to the directory on up on the hierarchy. If you keep on repeating the second command, you will end up back at the root (`/`) directory. Do this now.

2.2 File and directory manipulation

Git is a version control tool originally created by Linus Torvalds. More details on how Git can be used later, for now, we are going to use it to get the Activity files from the GitHub.com repository. Navigate to your ∼/`Downloads` directory and clone the repository:

```
git clone https://github.com/dschoonwinkel/networkingpgcourse.git
```

In this directory you will see 3 files and the Latex folder (in the Latex folder is the source documentation for this tutorial that you are reading).

The python_populator.py file is a Python script that writes text to "some_text.txt". You can run the python script with

```
python python_populator.py
```

This will start writing to "some_text.txt". You can stop programs on Linux by pressing Ctrl + C. Now use

```
cat some_text.txt
```

to display the contents of the text file. You can use `cat` on any type of text file (even Python scripts and C++ source files) to quickly show the contents of that file.

Now, open another Terminal window, and show them side by side. Start the python script in the one window, and run the `cat` command above repeatedly on the other. You should see that the contents of the text file is growing. You can confirm this by running `ls -l` repeatedly. If you want to see the results of a program repeatedly, you can use this command:

```
watch −n0 cat some_text.txt
```

`watch` calls a command repeatedly at an interval specified (`-n0`, i.e. as frequently as possible). This is very useful for checking the frequently changing status of something. I use it a lot with `ifconfig` for example.

Now stop your program with Ctrl+C. If you wanted to start your python script in the background, i.e. not active in your current terminal allowing you to continue using the terminal, you can accomplish that with the `&` symbol:

```
python python_populator.py &
```

It still seems as if the program is active in the terminal, but if you press Enter, you will see the terminal is ready to accept a command (shown by the $ at the end of the line). One the script prints something out again, the $ disappears, but can be seen again after you press Enter.

This is because of stdout, etc... kill, killall, ps -e | grep, fg, bg, stdout redirection etc.

2.3 References

List of sources:

- http://tldp.org/LDP/abs/html/writingscripts.html