Week 02
More about the command line and shell scripting

Melvyn Ian Drag

January 30, 2020

Abstract

For tonight's class we'll learn more about the Linux command line and shell scripting.

Working on the Linux Command Line

You know some commands already:

1. cat
2. ls
3. cd
4. cp
5. cut
6. cut -c
7. cut -b
8. echo
9. touch
10. **What else?**

Here are some more commands that are useful:

1. sort
2. uniq
3. tee
4. history
5. grep

these commands are particularly useful in the context of **pipes**. Pipes are one of the defining features of Unix systems, they are one of the things that make unix great. A major tenet of Unix philosophy is that programs should be very simple and do one thing - just one simple thing - but do it incredibly efficiently and always get the correct result. Then these simple programs can be strung together to make more interesting programs that are fast and correct, because the tiny programs (like cat and sort and ls and cd, etc.) are fast and correct.

Pipes

Consider the following file:

```
10
1
2
3
4
8
11
91
871
78
11
-10
11
200
8000
10000
-2
-3
```

There are many things you can do with this file. By the way, I have it saved in my current working directory as 'numbers.txt' - you should do the same. Then type the following commands and note the output for each one. Use the output to think about what the various commands do.

```
melvyn@laptop$ cat numbers.txt | sort
melvyn@laptop$ cat numbers.txt | sort -n
melvyn@laptop$ cat numbers.txt | sort -g
melvyn@laptop$ cat numbers.txt | sort | uniq
melvyn@laptop$ cat numbers.txt | uniq # note that this does not work.
```

So you see the idea of pipes? We used a few linux commands together, separated by a '|' to achieve a cool result. I showed you above a few ways to sort, and then the correct and the wrong way to find the uniq numbers in a file.

Let's continue with this discussion of pipes:

```
melvyn@laptop$ echo hello | cut -b1-2
melvyn@laptop$ echo hello | cut -c1-4
melvyn@laptop$ echo "hello world" | cut -d" " -f1
melvyn@laptop$ te how cut with d and f is finicky
melvyn@laptop$ echo "hello world" | cut -d" " -f2 # many spaces between hello and world.
melvyn@laptop$ default cut uses TAB ( \t ) as delimiter.
melvyn@laptop$ cat file | md5sum
melvyn@laptop$ md5sum file
melvyn@laptop$ cat file | sha256
melvyn@laptop$ history | grep "that cool command I can only remember part of" # I do this at least 50
    times a day
```

...this bit about 'cut' is super important.... let's play with it more to make sure you understand it... heck... to make sure *I* understand it.

Spend 5 minutes playing with cut and pipes to extract various bits of information

stdin, stdout, stderr

Intro

What a great time to be in a class. In the next few minutes you're going to hear about what all the Linux people know.

So far I've shown you stdout, but I never told you what it was.

```
melvyn@laptop$ # the > is for stdout
melvyn@laptop$ echo "hello" > hello.txt
melvyn@laptop$ cat hello.txt
hello
```

The greater than sign indicated stdout (standard out). It takes the output of a command and routes it somewhere. In this case, the standard out of 'echo' was routed to a file called 'hello.txt'.

Another interesting thing to note is that if you repeatedly use this operator, it does not append to the output file, it overwrites it.

```
melvyn@laptop$ # the > is for stdout
melvyn@laptop$ echo "hello" > hello.txt
melvyn@laptop$ echo "hello" > hello.txt
melvyn@laptop$ echo "hello" > hello.txt
melvyn@laptop$ echo "hello" > hello.txt
melvyn@laptop$ wc -l hello.txt
```

What do you expect? 4 lines or 1 line? Try it for yourself! You will find that there is only one line. If you want to append output and not overwrite, you have to use this operator:

```
melvyn@laptop$ # the > is for stdout
melvyn@laptop$ echo "hello" >> output.txt
melvyn@laptop$ echo "hello" >> output.txt
melvyn@laptop$ echo "hello" >> output.txt
melvyn@laptop$ echo "hello" >> output.txt
```

```
melvyn@laptop$ wc -l hello.txt
```

Now you will see there are four lines!

Details

All linux processes have at least 3 "file descriptors" or communication channels with the outside world. Does anyone not know what they are? Then Ill tell you. You have standard in, standard out, and standard error. We've already begun using them in this class briefly with the > operator.

Open vim and create a file. Type a couple of words in it, we're going to use that file for the next exercise.

```
melvyn@laptop$ wc fileThatExists > log
```

You see nothing. Running cat log will show you the output of the wc tool

```
melvyn@laptop$ wc filethatdoesntexist > log
```

You see an error in the shell, cat log shows nothing. There was no std out of wc, but there was err output

```
melvyn@laptop$ wc filethatdoesntexist 1>log 2>errLog
```

Now there is no error on the screen, but cat errLog shows the error from wc.

```
melvyn@laptop$ wc filethatdoesntexist >log 2>errLog
```

You can put the 1 or not, it assumes the 1 is for stdout, 2 is for stderr.

A common "idiom" you'll see is

```
melvyn@laptop$ wc file 1>place 2>&1
```

that redirects both stdout and stderr to the same location.

You can also write

```
melvyn@laptop$ wc filethatdoesntexist &>place
```

to redirect both to 'place'.

As an aside look what vim does to the files. Not essential stuff, but this is just to deepen your knowledge of vim and how much attention to detail is required in programming. What if you are working on a project and there is a datafile that shouldn't have newlines. Then you peek in the file with vim and absent mindedly close it with :wq. You could cause an error that could take days to debug!!

```
wc errLog
0 blah blah
```

There are no newlines

Not open errLog with vim and quit immediately with :q

```
melvyn@laptop$ wc errLog
0 blah blah
```

Now open errLog with vim and close with wq. Then

```
melvyn@laptop$ wc errLog
1 blah blah
```

Now there is a newline! Writing with vim - vim always puts a newline at the end of the file.

Now lets move on to standard in. We've already seen alot of it. Pipes use standard in!! You can always pipe the stdout of one process to the stdin of another!

```
username@computer$ cat file | wc
```

The stdin of wc picks up the stdout of file! Processes can take inputs and give outputs.

There are various ways to feed a process's standardin. Some tools allow you to just give it as a positional argument, like wc.

```
student@computer$ wc file
```

'file' goes to the stdin of wc.

You can also be more explicit. We saw that stdout is "1" for some mysterious reason, and stderr is "2" for an equally mysterious reason. Well, not a mystery, this is just boring technical stuff. Some programmer way back when made the decision to associate these numbers with the indicated channels, and that's the way it is! Why do we type "cp" to copy a file??? Just another boring technical reason, that's how the computer program was written , right?? Well, stdin is 0. Also, output is associated with > and », input is associated with <

So we can also type 'wc < file' or 'wc 0< file'

I don't know if I've ever written this syntax, maybe once or twice. I'm an application and operating system developer mainly. It certainly isn't in my day to day work, but it's good to know in case you come across it in a script, an exam, or have to teach it!

Of course, you can mix these things

```
melvyn@laptop$ wc 0< file > log`
```

Also, don't forget about "append" vs write when it comes to the stderr/stdout stuff

```
melvyn@laptop$ wc filethatdoesntexist 2>> err
melvyn@laptop$ wc filethatdoesntexist 2>> err
melvyn@laptop$ wc filethatdoesntexist 2>> err
melvyn@laptop$ wc filethatdoesntexist 2>> err
melvyn@laptop$ cat err
```

You'll see alot of error messages logged in "err"

An aside, maybe you're interested? How to pipe standard error? `https://stackoverflow.com/questions/1507816/with-bash-how-can-i-pipe-standard-error-into-another-process`

A common thing you'll see and maybe want to do is make your processes shoosh! If a process is running and giving alot of output you don't want you can throw away the output channels.

```
melvyn@laptop$ wc f 2>/dev/null
```

A common place to send unwanted output is to /dev/null, it just writes your output to the ether and you don't see it and it isn't logged anywhere.

The above command should give an error, but you won't see it, its just gone. There are many times you'll want to do this. I won't dream up some big situation right now to illustrate this to you, just know you'll see this all over the place in bash scripts and there will come a time, if there hasn't already, where you'll just want to throw away either stdout or stderr and never hear about it.

9:00 More vim commands to make your work better.

- h, j, k, l
- I prefer to just use the arrow keys
- modes i, esc
- quit with :q. To save and quit you use :wq or :x. If you want to know the difference, google the diff between :wq and :x. there is a slight difference, but it doesnt matter.
- dd to delete a line
- y is copy.
- p is paste after the cursor
- shift p is paste before the cursor
- u is undo
- CTRL + r is to redo.

Here is some more functionality that is so powerful and unique to vim that it will make you feel scared. You've never had this power before and this is going to make you think vim is hard. It's not. You can ignore the following for now, but within 2 weeks when you've mastered the above you'll be ready to appreciate how great the below commands are. I'm just showing you now to plan the seed so that it can start to develop in your subconscious.

- w to go forward a word
- e to go to the end of the next word
- b to go to the previous word beginning.
- to go forward 5 words, 5w
- To go back 5 words, 5b.

Review return codes in Bash

Return value of last command is : $?. All these linux commands I've shown you return information to the terminal. They tell the terminal if the command was successful or not. To check the return code you type 'echo $?'.

```
melvyn@machine$ echo "hello world"
hello world
melvyn@machine$? echo $?
0
melvyn@machine$ ehco "aksljdg"
# error
melvyn@machine$ echo $?
127
#the 127 indicates an error. There was an error because ehco is not a command,
echo is.
```