Quantitative Applications for Data Analysis: Linux command line II

Erik Spence

SciNet HPC Consortium

9 January 2025

SciNet

Today's slides

Today's slides can be found here. Go to the "Quantitative Applications for Data Analysis" page, under Lectures, "Introduction to the Linux Shell II".

`https://scinet.courses/1376`

SciNet

Our commands so far

There are a couple of things to observe about the commands we've seen so far:

- The commands are designed to be fast and easy to use.
- The commands do, essentially, only one specific thing.
- The commands are pretty cryptic. Either you know them or you don't.
- Commands can take options. These are usually indicated with a '-something' flag (such as 'ls -F').

Our commands	
`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`arg`	mandatory argument
`[arg]`	optional argument

As you may have hoped, the purpose of this class is to teach you enough commands that you will be able to survive the Unix command line.

Setting up for today

```
[ejspence.mycomp] pwd
/c/Users/ejspence
[ejspence.mycomp]
[ejspence.mycomp] mkdir EES1137
[ejspence.mycomp]
[ejspence.mycomp] cd EES1137
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137
[ejspence.mycomp]
[ejspence.mycomp] mkdir assignment0
[ejspence.mycomp]
[ejspence.mycomp] cd assignment0
[ejspence.mycomp]
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`arg`	mandatory argument
`[arg]`	optional argument

Here we are creating a directory to hold your work for this class.

We create a directory, 'assignment0', to hold the files we'll create today.

SciNet

Our next command: echo

One of the simplest bash commands is 'echo'.

- The 'echo' command prints out whatever arguments it is given.
- This may seem silly, but combined with other commands it can be quite useful.

Don't forget to hit 'Enter' at the end of each line.

If you get an error message, it's likely you're running a different shell (csh, tcsh, zsh). Type 'bash' to start a bash shell, and try again.

```
[ejspence.mycomp]
[ejspence.mycomp] echo Hello
Hello
[ejspence.mycomp]
[ejspence.mycomp] echo Hello, world
Hello, world
[ejspence.mycomp]
[ejspence.mycomp] echo "Hello, adoring fans!"
Hello, adoring fans!
[ejspence.mycomp]
```

Text editors

It's time to write our first shell script. What is a script? A script is just a list of commands which you want the computer to run.

To write our script we need to use a text editor. NOT a word processor (WORD, for example). Good text editors include:

- Brackets (`http://brackets.io`)
- Sublime (`https://www.sublimetext.com`)
- VSCode (`https://code.visualstudio.com`)
- NetBeans (`https://netbeans.apache.org`)
- one of the command line text editors: nano, emacs, vi, vim, ...

We recommend against Notepad, Notepad++ (Windows) or TextEdit (Macs).

You will need a proper text editor for the rest of the semester.

SciNet

Our first shell script

Start your text editor.

- Create a new file called 'first.script.sh'.
- Save the file in the 'assignment0' directory.
- Put in the lines to the upper right.

The first line tells the computer to use 'bash' to interpret the commands.

The second is a 'comment'. Everything after the # is ignored by bash.

The other lines are like the commands from two slides ago. These are the commands to be executed.

DO NOT try to copy-and-paste code from PDF files! Bad things can happen!

```
#!/bin/bash
# first.script.sh
echo "Hello, world!"
```

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] ls
first.script.sh
[ejspence.mycomp]
```

Any commands which you can run on the command line can be put into the script.

File names

Some notes about file names.

- Do not try to name files the same names as built-in commands ('echo', 'pwd', 'cp').
- **Do not put spaces in your file names!**
- File name extensions do not matter in Linux systems.
- Periods in filenames are fine.

Note that Linux systems are case sensitive ("A" is not the same as "a"). Windows systems (git bash) may not respect this in general.

SciNet

Our first shell script, continued

Note that the code on the upper right is code as you would put it into your text editor (such as Sublime). The commands on the lower right are at the bash prompt, because there is a prompt.

The 'source' command tells the shell to run the commands in the script, one at a time.

```
#!/bin/bash
# first.script.sh
echo "Hello, world!"
```

```
[ejspence.mycomp]
[ejspence.mycomp] ls
first.script.sh
[ejspence.mycomp]
[ejspence.mycomp] source first.script.sh
Hello, world!
[ejspence.mycomp]
```

Assigning variables

We can create variables in bash.

- The '=' sign tells the shell to create a variable called 'myvar' and assign it the value "pants".
- There are no spaces around the '='.
- The variable's value is accessed using the $.
- When the $ is invoked, the shell finds the value of the variable and gives it to the command (echo) to process.
- There is nothing special about the variable "myvar", you can call variables just about anything, and have as many as you want.

```
[ejspence.mycomp]
[ejspence.mycomp] myvar="pants"
[ejspence.mycomp]
[ejspence.mycomp] echo Hello, world
Hello, world
[ejspence.mycomp]
[ejspence.mycomp] echo Hello, $myvar
Hello, pants
[ejspence.mycomp]
```

Our second shell script

```
#!/bin/bash
# second.script.sh
myvar="adoring fans!"
echo Hello, $myvar
```

Once again, we run the script using the 'source' command.

```
[ejspence.mycomp]
[ejspence.mycomp] ls
first.script.sh second.script.sh
[ejspence.mycomp]
[ejspence.mycomp] source second.script.sh
Hello, adoring fans!
[ejspence.mycomp]
```

Our third shell script: arguments

Suppose we'd like our script to behave slightly differently each time we run it. We don't want to have to rewrite the script for each possible case. How do pass information into the script, so we can slightly change its behaviour?

```
#!/bin/bash
# third.script.sh
anothervar="world"
echo Hello, $anothervar ${1}
```

- Information which is passed to a script is called an 'argument'.
- Any arguments which are given to a bash script are put into the variables ${1}, ${2}..., in order.
- The script can then access them and use them as needed.

Our third shell script: arguments, continued

```
#!/bin/bash
# third.script.sh
anothervar="world"
echo Hello, $anothervar ${1}
```

We run the script using the usual way.

- The arguments are accessed using ${1}, ${2}, etc.
- third.script.sh only uses the first argument; any other arguments are ignored.

We will use scripts in this manner the rest of the semester, to run data-analysis pipelines.

```
[ejspence.mycomp]
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] ls
first.script.sh second.script.sh third.script.sh
[ejspence.mycomp]
[ejspence.mycomp] source third.script.sh pants
Hello, world pants
[ejspence.mycomp]
[ejspence.mycomp] source third.script.sh wide web
Hello, world wide
[ejspence.mycomp]
[ejspence.mycomp] source second.script.sh wide web
Hello, adoring fans!
[ejspence.mycomp]
```

Manipulating files: copying

```
[ejspence.mycomp]
[ejspence.mycomp] ls
first.script.sh second.script.sh third.script.sh
[ejspence.mycomp]
[ejspence.mycomp] cp first.script.sh first-new
[ejspence.mycomp]
[ejspence.mycomp] ls
first-new first.script.sh second.script.sh
third.script.sh
[ejspence.mycomp]
[ejspence.mycomp] cp first-new ..
[ejspence.mycomp]
[ejspence.mycomp] ls ..
assignment0 first-new
[ejspence.mycomp]
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`arg`	mandatory argument
`[arg]`	optional argument

'cp' stands for 'copy'; it copies a file.

Manipulating files: moving

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp] ls
first-new first.script.sh second.script.sh
third.script.sh
[ejspence.mycomp]
[ejspence.mycomp] mv first-new new.txt
[ejspence.mycomp] ls
first.script.sh new.txt second.script.sh
third.script.sh
[ejspence.mycomp] mv new.txt ../first-new
[ejspence.mycomp] cd ..
[ejspence.mycomp] ls
assignment0 first-new
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`arg`	mandatory argument
`[arg]`	optional argument

- 'mv' stands for 'move'; it moves a file and/or renames it.
- mv can overwrite a file, so be careful when moving things!

Manipulating files: deleting

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137
[ejspence.mycomp]
[ejspence.mycomp] ls
assignment0 first-new
[ejspence.mycomp]
[ejspence.mycomp] rm first-new
[ejspence.mycomp] ls
assignment0
[ejspence.mycomp]
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`arg`	mandatory argument
`[arg]`	optional argument

- 'rm' stands for 'remove'; it deletes a file. It does not delete directories, by default.
- rm does not 'move the file to the Trash'. It deletes it; it's gone; it's not recoverable. Be sure before you use rm.

Wildcards

Wildcards (*) capture all possible combinations that fit a given description.

```
[ejspence.mycomp] cd assignment0
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp] ls
first.script.sh second.script.sh third.script.sh
[ejspence.mycomp] ls f*
first.script.sh
[ejspence.mycomp] ls *on*
second.script.sh
[ejspence.mycomp] ls *.pants
ls:  *.pants:  No such file or directory
[ejspence.mycomp] ls *.sh
first.script.sh second.script.sh third.script.sh
[ejspence.mycomp]
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`arg`	mandatory argument
`[arg]`	optional argument

The shell expands the wildcard into a list of all possible matches, and passes the list to the command.

SciNet

Head/Tail

```
[ejspence.mycomp]
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] head -2 first.script.sh
#!/bin/bash
# first.script.sh
[ejspence.mycomp]
[ejspence.mycomp] tail -3 third.script.sh
# third.script.sh
anothervar="world"
echo Hello, $anothervar ${1}
[ejspence.mycomp]
```

'head'/'tail' prints the first/last 10 lines of the input. Use "-n" to specify n lines.

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`arg`	mandatory argument
`[arg]`	optional argument

SciNet

Word count

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp] wc first.script.sh
3 6 51 first.script.sh
[ejspence.mycomp] wc -l first.script.sh
3 first.script.sh
[ejspence.mycomp] wc -w first.script.sh
6 first.script.sh
[ejspence.mycomp] wc -c first.script.sh
51 first.script.sh
[ejspence.mycomp] wc -w *
6 first.script.sh
8 second.script.sh
8 third.script.sh
22 total
```

'wc' stands for 'word count'. It counts the number of lines/words/characters in the input.

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`history [num]`	print the shell history
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file
`arg`	mandatory argument
`[arg]`	optional argument

SciNet

Searching files: grep

How do we search for character strings (words) within files?

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] grep ash first.script.sh
#!/bin/bash
[ejspence.mycomp] grep ello *
first.script.sh:echo "Hello, world!"
second.script.sh:echo Hello, $myvar
third.script.sh:echo Hello, $anothervar ${1}
[ejspence.mycomp]
```

grep prints the lines from the input that contain the search argument.

Our commands	
`man cmd`	command's man page
`rmdir dir`	delete a directory
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`wc file`	word count data of file
`grep arg file`	search for arg in file
`arg`	mandatory argument
`[arg]`	optional argument

If you forget the second argument grep will hang. Use Ctrl-C to cancel the command.

Pipelines of commands

How do we combine commands?

- Suppose we want to take the output of one command and use it as the input to another.
- Outputing one command straight into another is so common and useful that the shell has a special feature to do this, called the 'pipe'.
- The 'pipe' is the vertical line, found above your 'return' key.
- Note that the commands that are receiving information from the pipe do not take an file argument.

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] grep var *
second.script.sh:myvar="adoring fans!"
second.script.sh:echo Hello, $myvar
third.script.sh:anothervar="world"
third.script.sh:echo Hello, $anothervar ${1}
[ejspence.mycomp]
[ejspence.mycomp] grep var * | wc -l
4
[ejspence.mycomp]
[ejspence.mycomp] myvar="how long is this sentence?"
[ejspence.mycomp] echo $myvar | wc -c
27
[ejspence.mycomp]
```

The sort command

The sort command can take a number of important flags:

- -n: sort by number (not lexicographic; $10 < 30$ without -n).
- -k [num]: sort by the k'th column.
- -r: reverses order.
- -t [sym]: specify a different column separator.

```
[ejspence.mycomp] wc -c * | sort -n -k 1 -r
201 total
78 third.script.sh
72 second.script.sh
51 first.script.sh
[ejspence.mycomp]
```

More commands

`curl url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`arg`	mandatory argument
`[arg]`	optional argument

Cutting up the output

How do keep just part of the output?

```
[ejspence.mycomp]
[ejspence.mycomp] grep var * | head -1
second.script.sh:myvar="adoring fans!"
[ejspence.mycomp]
[ejspence.mycomp] grep var * | head -1 | cut -c -8
second.s
[ejspence.mycomp]
[ejspence.mycomp] grep var * | head -1 | cut -c 10-
ript.sh:myvar="adoring fans!"
[ejspence.mycomp]
[ejspence.mycomp] grep var * | head -1 | cut -c 2-5
econ
[ejspence.mycomp]
```

More commands

`curl url`	downloads the url
`tar file`	handles tar files
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`sort file`	sorts the lines of file
`source file`	run the cmds in file
`grep arg file`	search for arg in file
`cut flags output`	cut part of output
`arg`	mandatory argument
`[arg]`	optional argument

- "-c" tells cut to cut characters.
- "-8" means keep up-to-and-including character eight.
- "10-" means keep 10 and higher.

Saving information

If I need to save something, I use variables.

```
[ejspence.mycomp] echo "How many words are in this sentence?" | wc -w
7
[ejspence.mycomp]
[ejspence.mycomp] i=$( echo "How many words are in this sentence?" | wc -w )
[ejspence.mycomp]
[ejspence.mycomp] echo i
i
[ejspence.mycomp] echo $i
7
[ejspence.mycomp]
[ejspence.mycomp] echo "The value of my variable is $i."
The value of my variable is 7.
[ejspence.mycomp]
```

Enough to get started

- These commands are enough to get started with using the command line.
- As you have seen, Unix commands are simple, and are designed to do one specific thing.
- By combining these commands together we will be able to do more interesting things.
- If there is functionality that you think ought to exist, it probably does. Ask someone what the command is, or google it.

SciNet

Shell-command cheat sheet

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`man cmd`	command's man page
`echo arg`	echo the argument
`source file`	run the cmds in file
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`wc file`	word count data of file
`grep arg file`	search for arg in file
`cmd1 | cmd2`	pipe cmd1 output to cmd2
`arg`	mandatory argument
`[arg]`	optional argument

`rmdir dir`	delete a directory
`history [num]`	print the shell history
`file file`	type of file
`more file`	scroll through file
`less file`	scroll through file
`cat file`	print the file contents
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`cmd < file`	use file as input to cmd
`head file`	print first 10 lines of file
`tail file`	print last 10 lines of file
`curl url`	downloads the url
`tar file`	handles tar files
`sort file`	sorts the lines of file
`cut flags output`	cut part of output
`for..do..done`	for loop in bash
`if..then..fi`	if statement in bash
`logout`	close the terminal session

SciNet