Quantitative Applications for Data Analysis: Linux command line I

Erik Spence

SciNet HPC Consortium

9 January 2024

Today's slides

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

`https://education.scinet.utoronto.ca`

You can also access the class web site directly, here:

`https://scinet.courses/1346`

Who are we?

We are Erik Spence and Alexey Fedoseev.

- We are Applications Analysts at SciNet (`https://www.scinethpc.ca`).
- SciNet is a High-Performance-Computing (HPC) consortium, one of six in Canada, run by the University of Toronto.
- These consortia run massively parallel computers, with tens of thousands of cores, to perform computations that couldn't be done otherwise.
- Our job at SciNet is to help users get their code to run on these machines.
- We also educate users on how to write fast, efficient code.

About this class

Some notes about this class:

- This class aims to be a graduate course on data analysis and research computing, using R and Python.
- We will meet for twelve weeks, two lectures per week, on Tuesdays, 10:00am - 12:00pm and Thursdays, 11:00am - 12:00pm.
- Classes will be held in AA207 on Tuesdays and IC120 on Thursdays.
- The class unbalanced scheduling is new for us. We are still trying to figure out how that will affect the material.
- This class can be taken for credit by UTSC graduate students (EES1137), as well as graduate students at other campuses.

About this class, continued

Some notes about this class:

- There will be 10 approximately-weekly homework assignments, probably assigned on Tuesdays, since those classes are two-hours long, and due one week later at midnight.
- The assignments are 100% of your grade.
- Late assignments will be accepted until one week after the deadline (at 10:00am), with a penalty of 0.5 points per day (out of 10).
- The assignments are submitted through the class website.
- Office hours will be held on Thursdays from 12:30pm - 2:00pm (after Thursday's lecture), in EV340.
- Please, please ask for help if you need it!
  - Post questions to the class forum.
  - Talk to us, or email us if you have questions: courses@scinet.utoronto.ca.

Ask questions!

SciNet certificates

In addition to official UofT class credit, SciNet also offers its own certificates.

- We offer certificates in High Performance Computing, Scientific Computing and Data Science.
- Each certificate requires 36 SciNet credits; specific classes qualify for specific certificates.
- This class qualifies for 28 credits toward the Data Science certificate, and 8 credits toward the Scientific Computing certificate.
- Visit the SciNet education website to see what other courses are available.

`https://scinet.courses`

Class expectations

Some details about the class:

- Prerequisites: minimal-to-no programming experience is sufficient. The goal is to start slowly so all will be on the same page.
- Software you will need:
  - a Terminal program (needed immediately). On Windows, we recommend
    - "git bash", which contains both the terminal and "git", which will be needed later.
    - if you're using the Windows Subsystem for Linux (WSL), the terminal will be built in.
  - On a Mac you may use "Terminal".
  - A text editor (needed Thursday): Sublime, VSCode, Brackets.
  - R, and various R libraries (needed by week 2).
  - git, for version control (needed by week 4?).
- Grading scheme: the final grade will be based on the homework assignments.
- Attendance is not mandatory, though encouraged. If you don't attend, listen to the recordings! The slides do NOT constitute all of the class material!

Class website

SciNet hosts its own web site for its classes.

- `https://scinet.courses`, or `https://education.scinet.utoronto.ca`
- We will not be using Quercus.
- Log in with your SciNet account, or temporary account.
- Click on "Quantitative Applications for Data Analysis".
- Or go directly to the class web site: `https://scinet.courses/1346`
- All assignments will be submitted through this website.
- Let us know if you do not yet have an account.

All work for the class will go through this web site, so it is important that you have access.

Student Code of Conduct

Some details about doing the assignments:

- You are welcome to discuss your assignments with each other.
- You are not welcome to copy each other's code.
- You are not welcome to copy code you find on the internet, without giving credit.

`http://tinyurl.com/UofTCodeOfConduct`

Class topics

Our adventure in data analysis will cover the following:

- Getting started with the Linux command line.
- Getting start with R.
- Vectors, arrays, data frames.
- Version control, modular programming.
- Statistics and machine learning.
- Visualization.
- Much of the above, using Python.
- Other topics.

This list is subject to change. If there's a particular topic that you'd like covered, let us know.

Before we start

This class is intended to be fairly interactive. Eventually you will need R on your computers. This week we will be using the Linux command line. Hopefully you've already got a terminal program installed.

Windows users: we strongly recommend downloading "git bash" and using that as your R interface.

`https://git-scm.com/downloads`

As mentioned, R will be needed, but not just yet. Please install it at your convenience.

Today's class

Today we will visit the following topics:

- Motivation for using the command line.
- The file system from the command line.
- Manipulating files from the command line.

The point of today's class is to give you a first taste of the Linux command line. Please stop me if you have a question.

The Truth about interfaces

Why are we looking at the command line interface?

- Nobody. Nobody. Nobody, uses a Graphical User Interface (GUI) for HPC (High-Performance Computing). Nobody.
- And this includes repetitive or large-scale data analysis, and the majority of programming environments.
- Who cares? Well, if you're going to do repetitive data analysis it's possible you might need to use HPC to get it done.
- Even if you don't, knowing how to use this infrastructure will allow you to be significantly more efficient, consistent and productive in the management and analysis of your data.

GUIs versus the command line

Graphical User Interfaces (GUIs) have many strengths.

- Very good at using existing functionality, existing controls.
- Programs tend to have lots of functionality built into them, but can only do what they've been programmed to do.
- Can't save a series of commands to replicate functionality.
- Easy to learn. Hard to use for big tasks.

The Command Line Interface (CLI) has a different approach.

- A blank canvas; you get to program what you want to do.
- Good at creating new things.
- Commands that do already exist are very good at doing *one* thing.
- Commands that you create can be saved and re-used.
- Hard to learn. Easy to use for big tasks.

Why are we learning this?

I thought this was a data analysis class. Why are we learning this?

- The goal of this class is to make you a more-productive researcher.
- To that end we are going to teach you more than just statistics and how to program. We're going to teach you programming best practices.
- It will be painful, because you will be learning new ways of doing things.
- But we can promise you that you'll be more productive if you adopt the practises that we are going to teach you.

Running code from the command line, instead of through a GUI, is a necessary part of improving your productivity.

"The" shell

Open a Terminal:

- Windows: start up "git bash" (or "MobaXterm").
- Mac: Applications/Utilities/Terminal (drag this to the dock).
- Linux: xterm, eterm, ...
- The terminal launches a shell. The shell is what you are actually interacting with when you type commands.
- The shell provides access to files, the network, and other programs.
  - You type in commands.
  - The shell interprets them.
  - Performs actions on its own, or launches other programs.
- The most commonly used shell in Linux is 'bash'.
- There are others; mostly the same but some syntax is different.

The command line prompt

Now that we've got a terminal open, what do we see? We see the command line prompt!

On "git bash", the prompt looks something like this:

```
ejspence@mycomp MINGW64 ~
```

Where 'ejspence' is my username, and 'mycomp' is the name of my computer. On a Mac my prompt might look like this:

```
mycomp:~ ejspence$
```

On a Linux machine, my prompt might look like this:

```
[ejspence@mycomp ~]$
```

All of these are customizable, which we won't be covering today. It doesn't matter what it looks like, so long as you're comfortable with the prompt.

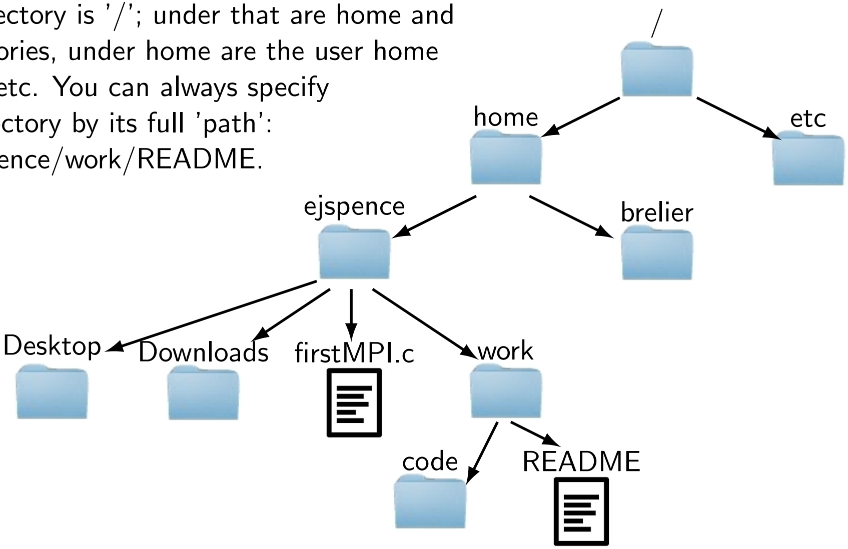
Basics: home sweet home

Where am I?

- Whenever you are using a shell you are located in some directory. You "are somewhere". This is called the "path".
- When you first launch a shell, you start in your "home directory", this is the top directory of all of your stuff.
- The home directory is /c/Users/username for "git bash", /Users/username on Macs, /home/username on Unix/Linux systems, /home/g/group/username on SciNet.
- If a path starts with a "/", it is a "full path" (or "absolute path"), otherwise it is a "relative path" (meaning the path relative to where you are currently located.
- The home directory is universally represented by the ~ symbol.
- Directories are sometimes called folders because of how they are represented in GUIs. We will call them directories.
- On Unix systems directories are listings of files, including other directories.

A typical Linux directory tree

The top directory is '/'; under that are home and other directories, under home are the user home directories, etc. You can always specify a file or directory by its full 'path': /home/ejspence/work/README.



Basics: the file system

I will be assuming I am on a "git bash" terminal, with a custom prompt. Your output will likely differ somewhat if you are on a different system.

```
[ejspence.mycomp]
[ejspence.mycomp] pwd
/c/Users/ejspence
[ejspence.mycomp] ls
Desktop LauncherFolder MyDocuments
[ejspence.mycomp] ls /c/Users
ejspence Public
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`arg`	mandatory argument
`[arg]`	optional argument

- 'pwd' stands for 'present working directory'. It will print the directory you are currently in. As mentioned on the last slide, you begin in your home directory.
- 'ls' stands for 'list'. If no argument is given it lists the contents of the current directory, otherwise it lists the contents of the argument. Some implementations of ls include colour.

Creating directories

```
[ejspence.mycomp] pwd
/c/Users/ejspence
```

```
[ejspence.mycomp] ls
Desktop LauncherFolder MyDocuments
```

```
[ejspence.mycomp] mkdir firstdir
```

```
[ejspence.mycomp] ls -F
Desktop LauncherFolder MyDocuments
firstdir/
```

```
[ejspence.mycomp] mkdir /c/Users/ejspence/2ndir
```

```
[ejspence.mycomp] ls -F
2ndir/ Desktop LauncherFolder MyDocuments
firstdir/
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`arg`	mandatory argument
`[arg]`	optional argument

- 'mkdir' stands for 'make directory'. It creates a new directory, putting it in the current directory unless a different path is specified.
- 'ls -F' lists the directory, as before, but labels directories with a '/'.

Moving between directories

```
[ejspence.mycomp] ls
2ndir Desktop LauncherFolder MyDocuments firstdir
[ejspence.mycomp] mkdir firstdir/temp
[ejspence.mycomp] cd firstdir
[ejspence.mycomp] pwd
/c/Users/ejspence/firstdir
[ejspence.mycomp] ls
temp
[ejspence.mycomp] cd temp
[ejspence.mycomp] pwd
/c/Users/ejspence/firstdir/temp
[ejspence.mycomp] cd ..
[ejspence.mycomp] pwd
/c/Users/ejspence/firstdir
[ejspence.mycomp] cd ~
[ejspence.mycomp] pwd
/c/Users/ejspence
```

Our commands

`pwd`	present working directory
`ls [dir]`	list the directory contents
`mkdir dir`	create a directory
`cd [dir]`	change directory
`arg`	mandatory argument
`[arg]`	optional argument

'cd' stands for 'change directory'. It moves you to the directory you specify. With no argument it moves you to the home directory.

Tips for getting around

Some common commands for moving around your directories:

- The directory above is represented by the '..' symbol; the current directory is represented by the '.' symbol:
  - 'cd ..' goes up a directory.
  - 'cd ../..' goes up two directories.
  - 'cd ../otherdir' goes up one directory and then down into 'otherdir'.
  - 'cd firstdir/seconddir/../..' goes nowhere.
  - 'cd ./././.' also goes nowhere.
- You can use absolute paths: 'cd /c/Users/ejspence/firstdir/temp'.
- ~ is the symbol for your home directory, on whatever system you are using. 'cd ~/work' goes to /c/Users/ejspence/work.
- 'cd' without any arguments goes to your home directory (~), from no matter where you are.
- 'cd -' goes back to the directory you were in previously.

Tips for using the command line

Some more helpful tips for using the command line:

- Use the 'tab' key, it will 'auto-complete' the available options based on what you've already typed,
  - start typing your command, and then hit 'tab'
  - the shell will fill in the rest, if there is only one option.
  - if nothing happens, there is either no option or more than one option.
  - hit the tab key twice, this will list all available options
  - continue typing to reduce the number of options, then hit tab again to fill in the rest.
- Use 'Ctrl-a' to go to the beginning of the command line, 'Ctrl-e' to go to the end of the line.
- Use the up arrow. This scrolls through the shell's 'history'.

History

```
[ejspence.mycomp] history
.
.
15 [2014-06-05 11:23:47] cd firstdir
16 [2014-06-05 11:23:49] pwd
17 [2014-06-05 11:23:50] ls
18 [2014-06-05 11:23:53] cd temp
19 [2014-06-05 11:23:55] pwd
20 [2014-06-05 11:23:58] cd ..
21 [2014-06-05 11:23:59] pwd
22 [2014-06-05 11:24:03] cd
23 [2014-06-05 11:24:05] pwd
24 [2014-06-05 11:24:11] history
[ejspence.mycomp]
```

Our commands

`echo arg`	echo the argument
`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
`arg`	mandatory argument
`[arg]`	optional argument

- The history command prints the commands that you've typed at the command line. "history 10" prints the last 10 commands.
- Use the up arrow to access the entries.

Cleaning up

```
[ejspence.mycomp] pwd
/c/Users/ejspence
[ejspence.mycomp]
[ejspence.mycomp] cd firstdir/temp
[ejspence.mycomp] ls
[ejspence.mycomp]
[ejspence.mycomp] cd ..
[ejspence.mycomp] ls
temp
[ejspence.mycomp] rmdir temp
[ejspence.mycomp]
[ejspence.mycomp] cd ..
[ejspence.mycomp]
[ejspence.mycomp] rmdir firstdir
[ejspence.mycomp] rmdir 2ndir
[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
`rmdir dir`	delete a directory
`arg`	mandatory argument
`[arg]`	optional argument

To delete a directory, use the 'rmdir' command. The directory must be empty.

Man pages

Know a command but aren't sure how to use the options? Use the man (manual) page!

- Most programs have a man page describing its use and all available options.
- These pages are good for finding out more about a command you already use, but are less good for learning new commands.
- Many programs have gazillions of options.
- No human being who has ever lived has know all the options for 'ls'.
- Over time you will find a few that you find useful for your favourite commands.
- Type 'q' to get out of the man page.
- Unfortunately, the 'man' command doesn't work with "git bash". Try adding the "--help" flag after a command to see the command-line options.

Man pages: help!

Use the man (manual) page for a list of all flags for a command.

```
[ejspence.mycomp] man ls
NAME
  ls - list directory contents
SYNOPSIS
  ls [OPTION]...  [FILE]...
DESCRIPTION
  List information about the FILEs (the current di-
  rectory by default).  Sort entries alphabetically
  if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory
for short options too.
-a, --all
  do not ignore entries starting with .
-A, --almost-all
  do not list implied .  and ..
...
```

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`arg`	mandatory argument
`[arg]`	optional argument

Not sure how to use the command? Not sure what options there are? Check the man page!

Type 'q' to get out of the man page.

Setting up for the rest of this class

```
[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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`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 download today.

Downloading today's data

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

```
[ejspence.mycomp] ls
```

```
[ejspence.mycomp] curl -O https://pages.scinet.utoronto.ca/~ejspence/UTSCfile
   %  Total     %  Received  %   Xferd  Average  Speed     Time
                                        Dload  Upload    Total
   0      0     0         0  0       0        0      0  --:--:--
 100    185   100       185  0       0      276      0  --:--:--
 100  15927   100     15927  0       0    16492      0  --:--:--
```

```
[ejspence.mycomp]
```

```
[ejspence.mycomp] ls
UTSCfile
```

```
[ejspence.mycomp]
```

The 'curl' command can download files from the internet:

- -O (capital O) tells curl to keep the filename.

Downloading today's data, continued

```
[ejspence.mycomp] ls
UTSCfile
```

```
[ejspence.mycomp] tar -z -x -f UTSCfile
```

```
[ejspence.mycomp]
```

```
[ejspence.mycomp] ls -F
UTSCfile data/
```

What happened?

- 'curl' downloads the (tarred) data file. A 'tar' file (also called a 'tarball') is a file in which has been bundled a number of other files, for easy of moving around.
- 'tar' handles a tar file,
  - -z means gunzip it.
  - -x means extract the contents of the file.
  - -f specifies which file you are applying this command to.

The data is now in the 'data' directory.

Wildcards

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

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0
[ejspence.mycomp]
[ejspence.mycomp] cd data
[ejspence.mycomp] ls
alexander Bert Frank_Richard gerdal jamesm
Lawrence THOMAS
[ejspence.mycomp] ls -d *er*
alexander Bert gerdal
[ejspence.mycomp] ls -d *e
Lawrence
```

With the '-d' flag 'ls' will only print the names of the directories, not the directory contents.

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
rmdir dir	delete a directory
man cmd	command's man page
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.

Manipulating files: copying

```
[ejspence.mycomp] ls
Bert          Lawrence alexander jamesm
Frank_Richard THOMAS   gerdal
```

```
[ejspence.mycomp] cd gerdal
```

```
[ejspence.mycomp] ls
.
Data0413 Data0468 Data0528 Data0558
```

```
[ejspence.mycomp] ls *27*
Data0227 Data0279
```

```
[ejspence.mycomp] cp Data0227 Data0227-new
```

```
[ejspence.mycomp] ls *27*
Data0227 Data0227-new Data0279
```

```
[ejspence.mycomp] cp Data0227 ..
```

```
[ejspence.mycomp] ls ..
Bert      Frank_Richard THOMAS    gerdal
Data0227  Lawrence      alexander jamesm
```

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`cp file1 file2`	copy a file
`arg`	mandatory argument
`[arg]`	optional argument

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

Wildcards can appear anywhere in the variable you are searching for. They don't need to come at the end.

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 is 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.

Manipulating files: moving

```
[ejspence.mycomp] pwd
/c/Users/ejspence/EES1137/assignment0/data/gerdal
```

```
[ejspence.mycomp] ls *27*
Data0227 Data0227-new Data0279
```

```
[ejspence.mycomp] mv Data0227-new new.txt
```

```
[ejspence.mycomp] ls *27*
Data0227 Data0279
```

```
[ejspence.mycomp] ls *txt
new.txt
```

```
[ejspence.mycomp] mv new.txt ../Data0227
```

```
[ejspence.mycomp] ls *txt
ls: *txt: No such file or directory
```

```
[ejspence.mycomp] cd ..
```

```
[ejspence.mycomp] ls *27*
Data0227
```

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`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/assignment0/data
[ejspence.mycomp] ls
Bert       Frank_Richard THOMAS     gerdal
Data0227   Lawrence       alexander jamesm
[ejspence.mycomp] ls *27*
Data0227
[ejspence.mycomp] rm Data0227
[ejspence.mycomp] ls *227*
ls: *227*: No such file or directory
[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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`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.

Looking inside files

```
[ejspence.mycomp] cd alexander
[ejspence.mycomp] pwd
/c/Users/ejspence/data/alexander
[ejspence.mycomp] cat data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
Year/month of birth: 1997/12
Sex: M
CI type: 20
Volume: 3
Range: 5
Discrimination:
[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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`cat file`	print the file contents
`arg`	mandatory argument
`[arg]`	optional argument

'cat' outputs the contents of the file.

Looking inside files, continued

```
[ejspence.mycomp] more data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
.
.
```

```
[ejspence.mycomp] less data_560.DATA
#
Reported: Sat May 7 10:50:03 2011
Subject: georgeSpice437
.
.
```

'more' (doesn't work in git bash), 'cat', and 'less' all output the contents of the file, but in different ways. Can you tell the differences? Type 'q' to get out of 'more' or 'less'.

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`cat file`	print the file contents
`more file`	scroll through file
`less file`	scroll through file
`arg`	mandatory argument
`[arg]`	optional argument

cat'ing files together

```
[ejspence.mycomp] ls *DATA
.
.
.
data_420.DATA data_502.DATA data_560.DATA
data_297.DATA data_357.DATA data_421.DATA
[ejspence.mycomp] cat *DATA > all-DATA
[ejspence.mycomp] ls *DATA
all-DATA data_297.DATA data_357.DATA
.
.
.
data_550.DATA data_292.DATA data_347.DATA
data_420.DATA data_502.DATA data_560.DATA
[ejspence.mycomp]
```

- 'cat' dumps the input to the screen.
- '>' redirects the input to a file, instead of the screen.

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
rmdir dir	delete a directory
man cmd	command's man page
cp file1 file2	copy a file
mv file1 file2	move/rename a file
rm file	delete a file
cat file	print the file contents
more file	scroll through file
less file	scroll through file
cmd > file	redirect output to file
arg	mandatory argument
[arg]	optional argument

cat’ing files together, continued

```
[ejspence.mycomp] less all-DATA
#
Reported: Wed Aug 17 13:56:38 2011
Subject: madonnaStarr178
Year/month of birth: 1995/02
Sex: N
CI type: 8
Volume: 7
Range: 3
Discrimination: 5
#
Reported: Thu May 19 09:08:14 2011
Subject: paulSpice199
Year/month of birth: 1994/01
Sex: M
CI type: 24
Volume: 4
:
```

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
rmdir dir	delete a directory
man cmd	command’s man page
cp file1 file2	copy a file
mv file1 file2	move/rename a file
rm file	delete a file
cat file	print the file contents
more file	scroll through file
less file	scroll through file
cmd > file	redirect output to file
arg	mandatory argument
[arg]	optional argument

Redirection fun

- cmd > file takes the output that would have gone to the screeen, creates a new file called file, and redirects (dumps) the output to the file. If the file already exists the previous content of the file is overwritten.
- cmd >> file takes the output that would have gone to the screen, and *appends* it to file. If the file doesn't already exist then it is created.

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`cat file`	print the file contents
`more file`	scroll through file
`less file`	scroll through file
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`arg`	mandatory argument
`[arg]`	optional argument

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
`rmdir dir`	delete a directory
`man cmd`	command's man page
`cp file1 file2`	copy a file
`mv file1 file2`	move/rename a file
`rm file`	delete a file
`cat file`	print the file contents
`more file`	scroll through file
`less file`	scroll through file
`cmd > file`	redirect output to file
`cmd >> file`	append output to file
`arg`	mandatory argument
`[arg]`	optional argument

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