Introduction to Computational BioStatistics with R: Linux command line I

Erik Spence

SciNet HPC Consortium

8 September 2020

Today's slides

To find today's slides, go to the "Introduction to Computational BioStatistics with R" page, on the right, under "Lectures", "Intro to Linux Shell I".

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

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

`https://scinet.courses/546`

Who are we?

We are Erik Spence and Marcelo Ponce.

- 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.
- We will meet for twelve weeks, two lectures per week, on Tuesdays and Thursdays, starting September 8, 12:00pm - 1:00pm.
- Class is held in online only.
- This class can be taken for graduate credit by students in IMS, and other departments (MSC1090H).

About this class, continued

Some notes about this class:

- There will be 10, approximately-weekly, homework assignments, usually assigned on Thursdays, due one week later at midnight, and worth 100% of your final mark.
- Late assignments will be accepted until one week after the deadline (at 12:00pm), with a penalty of 0.5 points per day (out of 10).
- The assignments are submitted through the class web site.
- Office hours will be held on Mondays from 2:00pm - 3:00pm, online.
- Class tutorials will also be held on Wednesdays and Fridays, from 12:00pm - 2:00pm. These will be run by the class TAs.
- Please, please ask for help if you need it! Talk to us, or email us if you have questions: courses@scinet.utoronto.ca.

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.

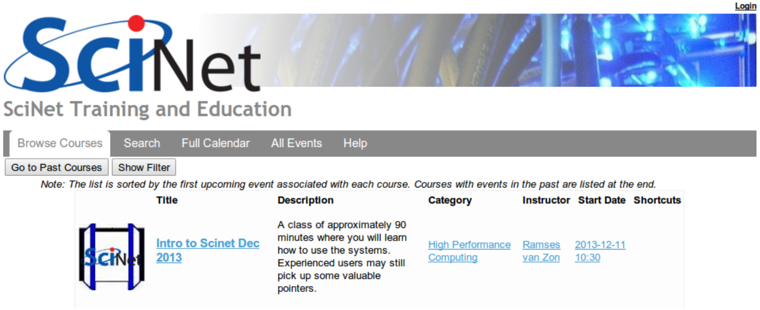
`https://scinet.courses`

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.
  - Alternatively on a Mac you may use "Terminal".
  - A text editor (needed Thursday): Atom, Brackets, Sublime.
  - 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 (100%).
- Attendance is not mandatory, though encouraged.

Class website



Login

SciNet Training and Education

Note: The list is sorted by the first upcoming event associated with each course. Courses with events in the past are listed at the end.

	Title	Description	Category	Instructor	Start Date	Shortcuts
	Intro to Scinet Dec 2013	A class of approximately 90 minutes where you will learn how to use the systems. Experienced users may still pick up some valuable pointers.	High Performance Computing	Ramses van Zon	2013-12-11 10:30	

`https://scinet.courses`

- Log in with your SciNet account, or temporary account.
- Click on "Introduction to Computational BioStatistics with R".
- All assignments will be submitted through this website.
- Let us know if you do not yet have an account.

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.
- 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.

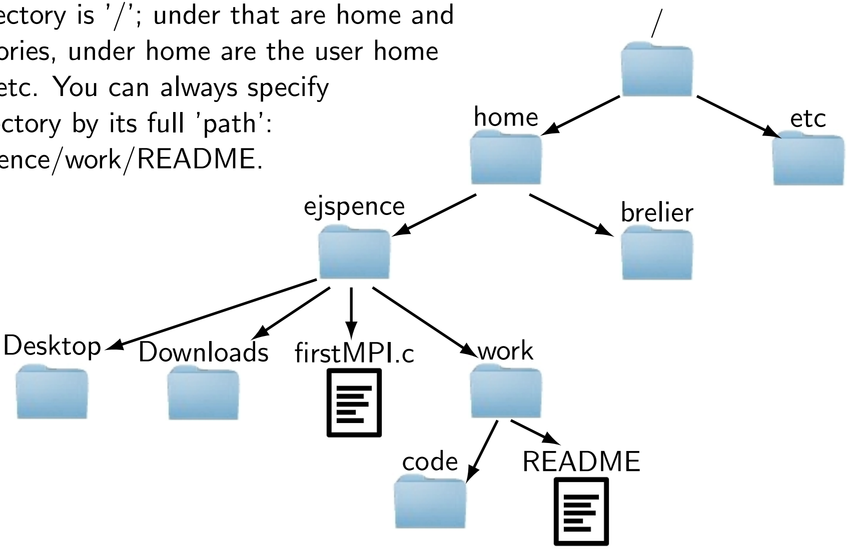
Basics: home sweet home

Where am I?

- When you 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.
- 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 ~/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'.
- Do not put spaces in your files names, nor any other special characters.

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.
- 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
`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.

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
`man cmd`	command's man page
`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.