Unix command line; editors
Tools for Reproducible Research

Karl Broman

Biostatistics & Medical Informatics, UW–Madison

biostat.wisc.edu/~kbroman
github.com/kbroman
@kwbroman
Course web: bit.ly/tools4rr

My goal in this lecture is to convince you that
(a) command-line-based tools are the things to focus on,
(b) you need to choose a powerful, universal text editor (you'll use it
a lot),
(c) you want to be comfortable and skilled with each.

For your work to be reproducible, it needs to be code-based; don't
touch that mouse!

Windows vs. Mac OSX vs. Linux

Remote vs. Not

The Windows operating system is not very programmer-friendly.

Mac OSX isn't either, but under the hood, it's just unix.

Don't touch the mouse! Open a terminal window and start typing.

I do most of my work directly on my desktop or laptop. You might prefer to work remotely on a server, instead. But I can't stand having any lag in looking at graphics.

If you're stuck with Windows...

Consider Cygwin (and perhaps Mintty)

Cygwin is an effort to get Unix command-line tools in Windows.

Mintty is a terminal emulator.

If you use a Mac...

Consider Homebrew and iTerm2

Homebrew is a packaging system; iTerm2 is a Terminal replacement

The command line is your friend

- ▶ Don't touch that mouse!
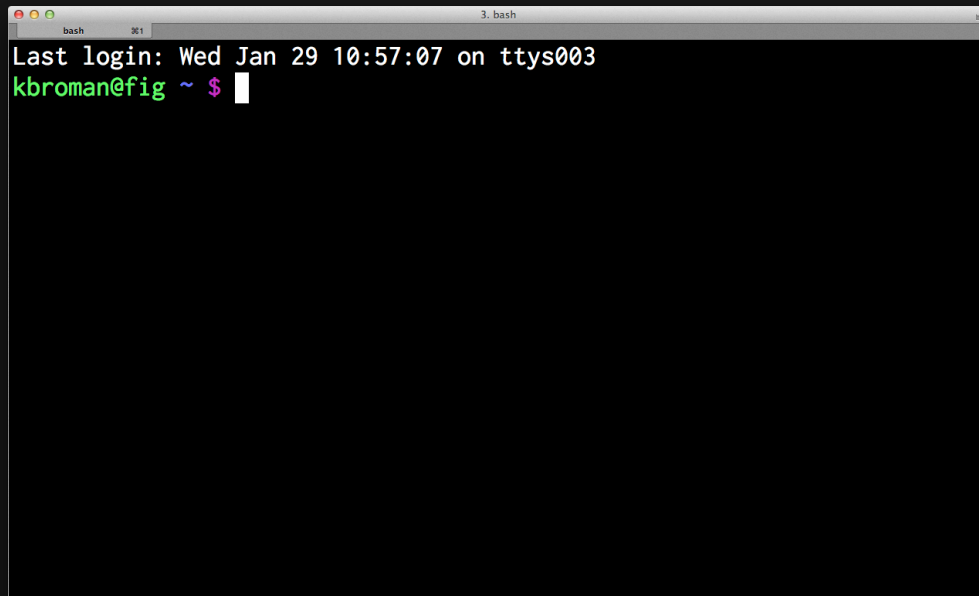
- ▶ Scriptable

- ▶ Flexible

In the long run, you'll be happier, having conquered the command line.

Pointing-and-clicking is not reproducible, and every time you take your hands off the keyboard, there's a loss of efficiency.

The command line allows you to piece together multiple tools and so do things that weren't anticipated by the developer of the GUI.

And it's only through scripts that you'll have truly reproducible analyses.

The shell

```
                                           3. bash
      bash        ⌘1
Last login: Wed Jan 29 10:57:07 on ttys003
kbroman@fig ~ $ ▊
```

Options: tcsh, bash, zsh

The shell is a program – an interface to the operating system.

There are a number to choose from. I use `bash`; I've heard great things about `zsh`.

Basics

- ▶ **Directory structure**
  Absolute vs. relative paths
  ```
  ls -l ~/..
  ```

- ▶ **Creating, removing, changing directories**
  ```
  mkdir
  rmdir
  cd
  cd -
  ```

- ▶ **Moving, copying, removing files**
  ```
  mv
  cp
  rm -i
  ```

7

This stuff is too boring to spend much time on.

But I should emphasize the importance of using relative paths (e.g., `../Figs/fig1.pdf`) in a project; reliance on absolute paths (e.g., `~/Projects/Blah/Figs/fig1.pdf`) make life difficult when you move the project to a different system.

~/.bash_profile

```
export PATH=.:/usr/local/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/lib

noclobber=1      # prevent overwriting of files
IGNOREEOF=1      # disable Ctrl-D as a way to exit
HISTCONTROL=ignoredups

alias cl='clear;cd'
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
alias ls='ls -GF'
alias 'l.'='ls -d .[a-zA-Z]*'
alias ll='ls -lh'
alias md='mkdir'
alias rd='rmdir'
alias rmb='rm .*~ *~ *.bak *.bk!'

alias Rb='R CMD build --force --resave-data'
alias Ri='R CMD INSTALL --library=/Users/kbroman/Rlibs'
alias Rc='R CMD check --library=/Users/kbroman/Rlibs'
alias Rcc='R CMD check --as-cran --library=/Users/kbroman/Rlibs'
```

Use the `.bash_profile` file to define various variables and aliases to make your life easier.

The most important variable is `PATH`: it defines the set of directories where the shell will look for executable programs.

If "." isn't part of your `PATH`, you'll need to type something like `./myscript.py` to execute a script in your working directory. So put "." in your `PATH`.

My `.bash_profile` file sources a `.bashrc` file; I don't quite understand when one is used versus the other. Google ".bashrc vs `.bash_profile`."

There are links to my `.bash_profile` and `.bashrc` files on the resources page at the course web site; some of it might just be total crap.

Redirection and pipes

```
$ locate article.cls > output.txt
$ locate book.cls >> output.txt
$ grep texmf output.txt

$ locate article.cls | grep texmf

$ locate article.cls | grep texmf | less

$ locate article.cls | wc -l
$ locate article.cls | grep texmf | wc -l

$ cat file1.txt file2.txt > combined.txt
```

Use > to redirect output "stdout" to a file.

Use >> to redirect output and append to the file.

Use < to have input "stdin" come from a file.

Use | to have the output of one command made the input to another.

A key design principle in Unix is the piecing together of small commands using this sort of technique. There are lots of little commands (often with short, cryptic names) that can be combined together with great flexibility.

Important tools mentioned here: `locate` (find files matching a pattern), `grep` (search for patterns in a file), `less` (look through long files a page at a time), `wc` (count the number of words, lines and/or characters in a file), `cat` (print contents of or concatenate text files)

Wild cards

```
$ grep blah *.txt

$ ls blah.???

$ ls [a-z]*

$ ls /usr/bin/[auz]*

$ ls /usr/bin/[auz]*.*
```

* stands for anything

? stands for a single character

Use [] to match some specific set or range of characters

Suspend/foreground/background

```
$ R CMD BATCH input.R output.txt &

$ R CMD BATCH input.R output.txt
[ctrl-Z]
$ bg

$ emacs afile.txt
[ctrl-Z]
$ fg
```

Use `&` to run a job in the background.

Use ctrl-Z to suspend the current job. Then use `bg` to then put it in the background or `fg` to bring it back to the foreground.

I use ctrl-Z and `bg` if I had forgotten to use `&`.

I use ctrl-Z with emacs sometimes, to do some command-line things without opening another shell/terminal; I'll then use `fg` to bring emacs back. Or I'll forget about it and muck a bunch of stuff up.

Other useful tools

```
$ find . -name *.py

$ ps ux

$ top

$ df -hk

$ du -h
$ du -hd2

$ ln -s ~/Projects/SomeFriend/Data
$ ln -s ~/Projects/SomeFriend/Data SomeFriend_Data

$ tar xzvf qtl_1.29-2.tar.gz
$ tar czvf blah.tgz Blah/
$ tar tzvf blah.tgz

$ wget http://kbroman.github.io/Tools4RR/pages/resources.html
```

`find` is like `locate`

`ps ux` to see what processes are running.

`top` gives an interactive view of what processes are running.

`df -hk` shows disk usage

`du -hd2` shows disk usage in a directory and its subdirectories; the `d2` bit says go no more than 2 levels down through the subdirectories.

`ln -s` makes a "soft link" to a file or directory. It acts like there's a copy, but it's not really copied.

`tar` is used to archive a bunch of files within a single file. `x` for extract, `c` for combine, `t` for test, `z` for compress/zip, `v` for verbose, `f` for "file name to follow."

`wget` (see also `curl`) for grabbing something from the web.

Further useful tools

```
$ whereis bash
$ type rm
$ type emacs

$ pwd

$ head afile.txt
$ tail afile.txt
$ head -n20 afile.txt

$ man head

$ kill 8453
$ kill -9 8453

$ history
$ !!
$ !-2
$ !503

$ ping www.google.com

$ ispell afile.txt
```

whereis for finding a program.
type for figuring out the location of a program or the definition of an
alias.

pwd – print working directory

head – print first few lines of a file
tail – print the last few lines of a file

man – view manual page

kill – kill a job

history – view command history
! – execute past commands

ping – see if you can connect to some server

ispell – spell checker

Moving around the command line

`ctrl-f, ctrl-b`	move forward and back
`ctrl-a, ctrl-e`	move to beginning and end of line
`ctrl-k, ctrl-u`	delete rest of line, or to the start
`ctrl-l`	clear the screen
`ctrl-c`	cancel what you've typed
`tab`	autocomplete command or file
`ctrl-p, ctrl-n`	forward and backward in history
`ctrl-r`	search for a previous command

These are mostly emacs-like key "bindings".

File modes

```
kbroman@fig ~/Teaching/Tools4RR/Lectures (master) $ ll
total 8
drwxr-xr-x  27 kbroman  staff    918B Jan 27 11:35 01_Intro/
drwxr-xr-x  30 kbroman  staff    1.0K Jan 29 11:38 02_Unix/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:28 03_KnitrMarkdown/
drwxr-xr-x  37 kbroman  staff    1.2K Jan 20 23:05 04_Git/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:28 05_Organization/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:28 06_EDA/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:28 07_ClearCode/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:28 08_Rpack/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 09_TestingDebugging/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 10_BigJobs/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 11_KnitrPapers/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 12_KnitrTalks/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 13_KnitrPosters/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 14_Python/
drwxr-xr-x   2 kbroman  staff     68B Jan 14 06:29 15_Licenses/
-rwxr-xr-x   1 kbroman  staff    488B Oct 23 16:18 createVersionWithNotes.rb*
kbroman@fig ~/Teaching/Tools4RR/Lectures (master) $
```

Note the mode, owner, and group for each file.

mode = read/write/executable for owner/group/everyone

r = readable; w = writable; x = executable (for a directory, enter-able)

File modes/owner/group

```
sudo chown kbroman .
chgrp -R staff .
chmod +x createVersionWithNotes.rb
chmod 755 02_Unix
chmod 644 02_Unix/02_unix.tex
chmod 700 Private_stuff
```

You don't usually need to change the owner or group assigned to a file or directory, but it's good to be aware of the possibility.

Groups are useful if you want a file accessible by some set of people but not everyone. You need a system admin to set up the group.

You often want to make scripts executable, or make files/directories unreadable or unwriteable.

For example, primary raw data files should not be writable. Large Excel-based data files often contain screwed up cells where someone was typing in some random spot without realizing it. I found myself doing that yesterday!

The octal codes (e.g, 755 and 644) are convenient, once you get the hang of them.

How to solve computing problems

- Try stuff!
- man pages and help files
- `blah -h` or `blah --help`
- Google
- Stackoverflow
- Google with `site:stackoverflow.com`
- email lists and google groups
- friends or colleagues
- Twitter
- Buy a book. Buy all of the books.

It's surprising how many computing problems can be solved by googling the error message.

You will run into crazy and mysterious errors. Will you give up, or figure them out?

I do tend to but all possible books on a topic that is of even passing interest to me. I read at least part of each of them.

Examples

- ► How do you suppress warnings in knitr?
- ► What symbol corresponds to the unicode `\u00B1`?
- ► What's the difference between `curl` and `wget`?
- ► What does "`502 Bad Gateway`" mean?
- ► "`To open gs you need to install X11`"
- ► `mclapply` isn't working in Windows
- ► How to ping a server in Python?
- ► `Font shape `EU1/pplx/m/n' undefined`
- ► `except KeyError, k: raise AttributeError, k`

These are examples of things you might search for.

If you don't understand an error message, start by pasting it into google.

Important principle

Learn to code by looking at good code.

Identify programmers that you respect (e.g., Hadley Wickham), and study what they do.

Choose a good editor

- ► Emacs

- ► VIM

- ► RStudio

- ► Textwrangler

- ► Notepad++

- ► Tinn-R

I use emacs; I should probably use vim.

RStudio is increasingly useful, but as a general editor (for things that aren't R), I think it's insufficient.

The choice of editor is very personal.

A good editor

- ▶ Doesn't require pointing-and-clicking

- ▶ Easy to get code between R and a script

- ▶ Syntax highlighting of code

- ▶ Automatic indentation

- ▶ Close parentheses/brackets/braces

- ▶ Browse code across files

- ▶ Integrated with other tools (e.g., version control)

I've not figured out how to explore code across a set of files in emacs; otherwise I'm very happy with it.

Don't forget to look at the resources page

`http://kbroman.github.io/Tools4RR/pages/resources.html`

If you find other useful resources, let me know.

When we get to git and GitHub, make a pull request!