# Intro to BASH

Group 1

First Year Bootcamp, 2016

## What are we bashing and why?

- No violence involved, bash is a program!

## Table of Contents

## What are we bashing and why?

- No violence involved, bash is a program!
- bash (the name is an acronym for Bourne-Again SHell, don't ask) is a powerful command line interpreter that is the default on most Linux distros and OS X.
- In other words, we're going back to the way people used to use computers in the old days, or the way that "hackers" use them on TV.
- Why? Bash allows you to:
  - Simplify tasks that you could possibly do in other ways. Want to rename 1000 data files? Bash makes it (relatively) easy!
  - Access powerful tools like ssh and git that you otherwise couldn't.

## Opening bash

- **OS X/Linux:** Open terminal, bash is default shell
- **Windows:** Go to Start → Git → Git Bash
- You can get a list of commands by typing *help* in the prompt and hitting enter.
- You can learn more about a command or program by typing *help command* or *man command* (Note: *man* doesn't work on windows, but you can always use google!)

## Table of Contents

## Where am I? (Directories)

- Just like when you use explorer or finder to navigate on your computer, bash sees your files as organized into directories (folders). Whenever you use bash, you're always in a directory (your working directory). To find out what directory you're in right now, try typing *pwd* (print working directory) in the prompt, and then hitting enter.
- Did it print a directory name? Good! Now try typing *ls* (list) into the prompt. This should list the directories and files that are within this directory.
- You can change to a new directory by typing *cd* (change directory) followed by the directory name. For example, try *cd* ∼. (∼ is a special character that refers to your home directory, usually */home/your-user-name/*.)
- Now try using *ls* again, and then using *cd* to enter one of the directories you see (perhaps Documents, Desktop, Downloads or similar, depending how your OS is set up).

## Arguments and flags for *ls*

- What if you want to list the contents of a directory other than your current one? In that case you can just give that directory's path as an argument to *ls*. For example: *ls /home* will tell you what's in the directory */home*, no matter where you are.
- There are various flags (usually a - followed by a single character, or a – followed by a word) you can pass to *ls*. For example, try *ls -l* (more information every item on its own line), *ls -a* (lists hidden files as well), or *ls –group-directories-first* (self explanatory).

**Arguments & flags**

Most commands we will talk about can take many different kinds of arguments and flags to alter their function, don't forget to use *man* and *help* to find out more about them! Try it now with *man ls*.

## Creating, moving, and destroying directories

- Maybe you want to create a new directory to house all your exciting new grad stuff, like the pictures that random tourists take of you. To do this, type *mkdir* (make directory) followed by the name for the directory, e.g. *mkdir exciting-grad-school-stuff*
- Maybe you changed your mind about what to call it though, so you want to change it to *boring-grad-school-stuff*, and also move it somewhere else. Not to worry! You can use *mv* (move) to rename the directory, or to move it to a new place. For example, you could use *mv exciting-grad-school-stuff ~/boring-grad-school-stuff* to move it to your home directory and rename it. Give it a try!
- There is also a command called *cp* (copy) that works like *mv* except it copies the file.
- Finally, maybe you think this is a silly directory and want to get rid of it. You can do that by using the command *rmdir* (remove directory), e.g. *rmdir ~/boring-grad-school-stuff*.

## Special directories

- There are two special directories you'll see listed if you type *ls -a*, . and .., which are used to refer to the current directory and its parent, respectively.
- For example, if you are in ~/*Documents/grad/* and type *cd ..* your working directory will change to the parent of your current directory, i.e. ~/*Documents*.

# Table of Contents

## What's all this stuff? (Files)

- Now let's look at the files within a directory. Many of the commands you've already learned still apply, *ls* will list them, and *mv* will move them.
- Let's create an empty file to play around with. You can do this by typing *touch emptyfile.txt* (in real life you'll usually be working with files you create in other programs, this way of creating them is just an example).
- Now let's try to put some text in the file and save it, just for fun. You can do this from the command line, but how exactly you do it will depend on your OS.
  - **OS X:** try *open emptyfile.txt*.
  - **GNOME-based linux distros:** try *gedit emptyfile.txt*.
  - **Windows:** try *notepad emptyfile.txt*

## Manipulating files, listing selectively

- Now that we put some text in the file, maybe we ought to rename it to *nonemptyfile.txt*. How do you think we do that?
- Yup, *mv emptyfile.txt nonemptyfile.txt*.

**Warning!**

*mv* overwrites any files with the same name(s) in its destination, so be careful when using it! If there was another file in this directory called nonemptyfile.txt, we would have overwritten it.

## Selectivity and globs

- Let's suppose you come back tomorrow and can't remember what you called this file. You can type *ls* to list everything in the directory and look through for it, but there might be a lot of other stuff. You can make commands be more selective by giving them some hints. For example, type *ls *.txt* to list all files in the current directory ending with a *.txt* extension. Type *ls *empty** to list files with *empty* in their name.

### Globs

The character * is called a glob, because it sticks together all the files that complete the rest of the pattern I guess, I don't know. Globs are often useful, e.g. you could move all the csv files in the current directory to new directory by typing *mv *.csv  /Documents/my-new-data-directory*

## Removing files

- Just like you can remove directories, you can remove files by using the command *rm* (remove). Try it now by typing *rm nonemptyfile.txt*.

**Warning!**

*rm* is VERY DANGEROUS, especially when used with globs and/or with certain flags (use *man rm* to find out more). It does not simply move a file to the trash, it deletes it completely. Double check what you type before you run it, and don't use *rm* if you're not sure what you're doing.

## Wrapping up

Bash contains or allows access to many powerful tools, and has many arcane (but useful!) features. Here are a few examples of the things you can do with it that we won't have time to explain to you. If you're interested in more info, talk to us and we'll be happy to provide it!

## Table of Contents

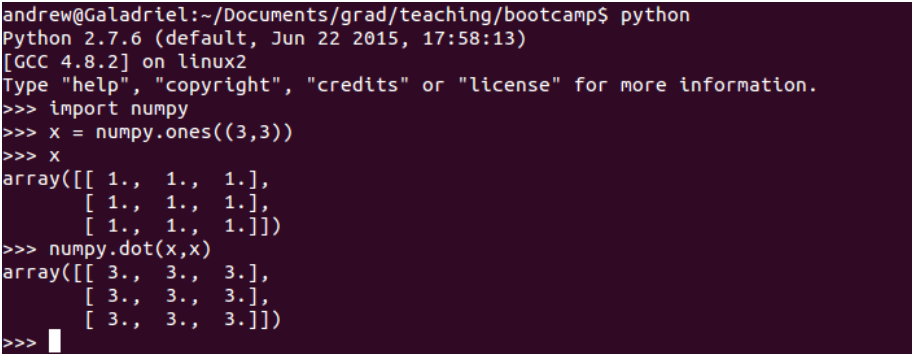
## Being lazy (scripts)

One of the main reasons bash is useful is that bash commands can be saved into scripts that can be reused. For example, here's a simple shell script I wrote to create anonymized data files by replacing participant identifiers in filenames with numbers starting from 0:

```
#!/bin/bash
i=0
for f in data_subject_*.json
do
  cp $f ../anonymized_data/data_subject_${i}.json
  i=$((i+1))
done
```

## Programming

The terminal gives you access to programming languages like python, both by running programs directly, and by using **interpreters**, programs that run in the terminal and allow you to run commands interactively, much as you would in Matlab or R.



```
andrew@Galadriel:~/Documents/grad/teaching/bootcamp$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> x = numpy.ones((3,3))
>>> x
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> numpy.dot(x,x)
array([[ 3.,  3.,  3.],
       [ 3.,  3.,  3.],
       [ 3.,  3.,  3.]])
>>> 
```

## Etc.

Also:

- Powerful commands for text manipulation (*sed, awk, grep*). Want to find all occurrences of a variable in many different code files and rename it? Want to extract lines from your datafiles that match a certain condition without having to read the files into R? Have a directory full of awful tab-separated datafiles and want to make them comma-separated? These tools can do it.
- Streams and piping: make commands work together, do file I/O, etc.
- Automation: more system specific, but *cron*, rc files, etc. allow for automation of things like backups, mounting filesystems when you start your computer, or sending reminders to subjects on a schedule.
- Text-editors like *vim*.
- Git version control – You'll learn about this at the later tutorial!

## Cheatsheet

| **Command** | **Effect** |
|---|---|
| *man command* or *help command* | Get manual/help for a command |
| *pwd* | Print working directory |
| *cd dir* | Change working directory to *dir* |
| *ls* | List directory contents |
| *mkdir dir* | Make a new directory called *dir* |
| *rmdir dir* | Remove the directory *dir* permanently |
| *mv source dest* | Move *source* file or folder to *dest* |
| *cp source dest* | Copy *source* file to *dest* (use -r for folders) |
| *rm file* | Remove *file* permanently |
| *ssh user@server* | Connect to *server* as *user* |
| *scp [[user1@]server1:]source [[user2@]server2:]dest* | Copy *source* from *server1* to *dest* on *server2* (if copying to/from your local computer, just omit the server and user parts in that file's path) |

(For a more comprehensive reference see
https://gist.github.com/LeCoupa/122b12050f5fb267e75f)

"That's all Folks!"