ILLINOIS

Introduction to the Shell I

Lecture 1

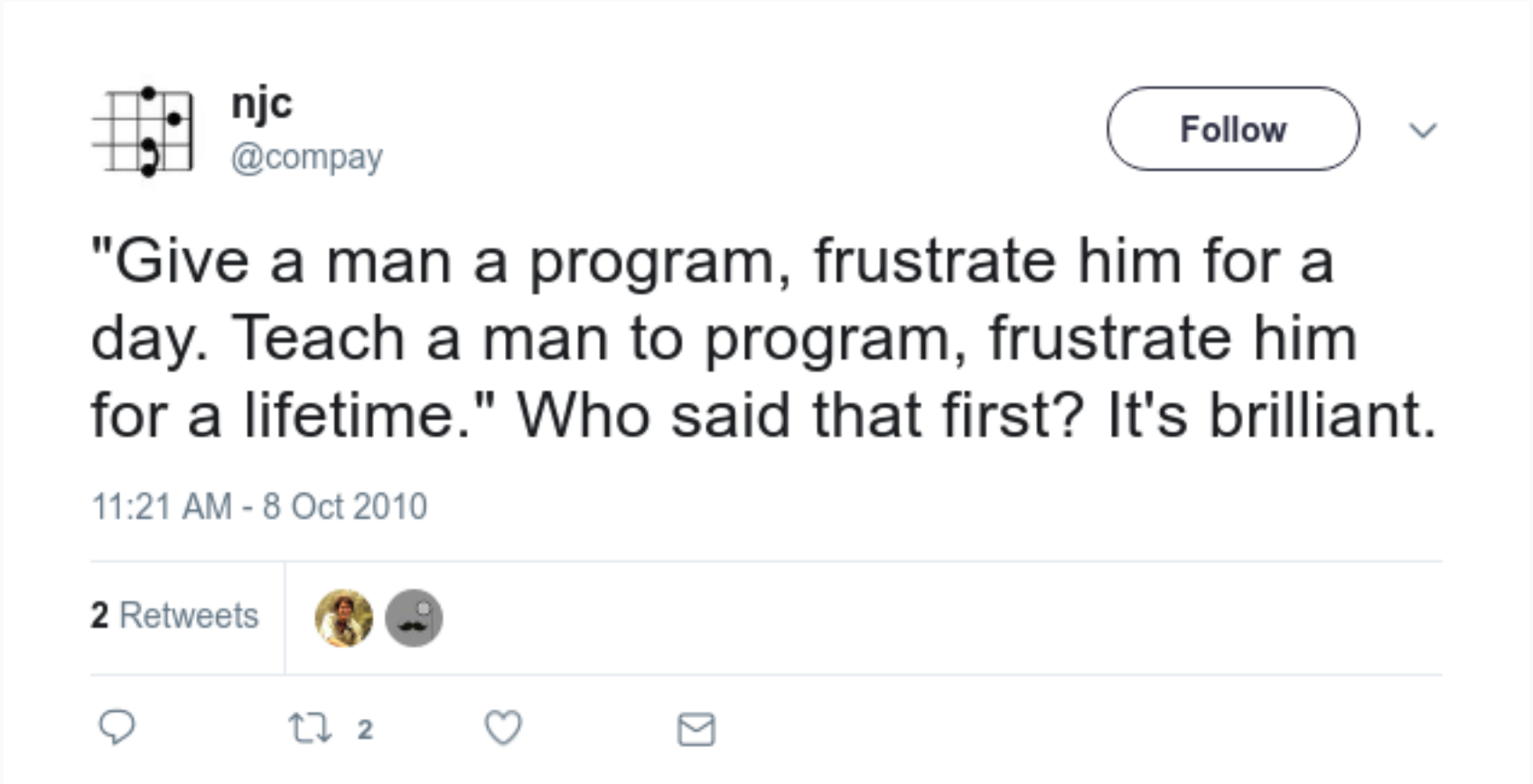
Dirk Eddelbuettel

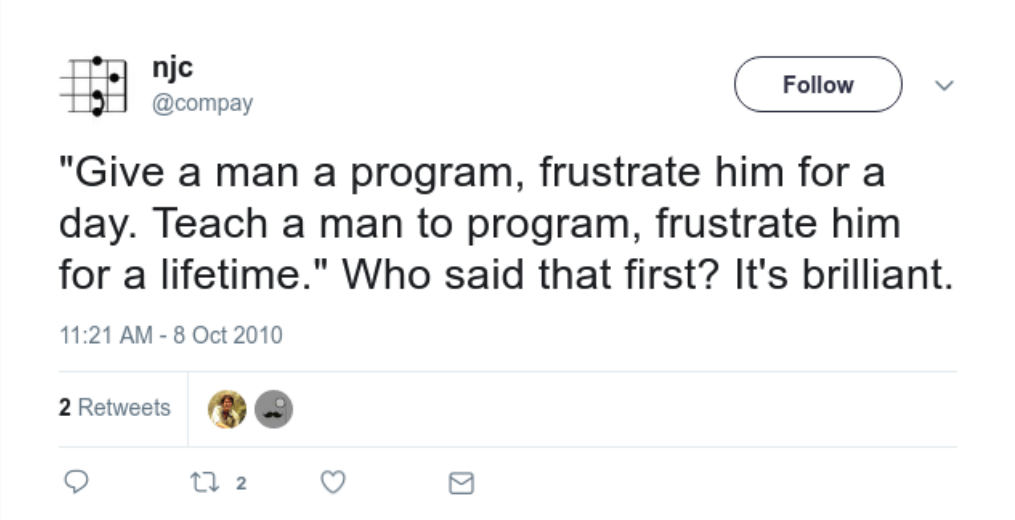
STAT 430: Data Science Programming Methods (Fall 2019)
Department of Statistics, University of Illinois

Brief Course Introduction:

Data Science

Programming Methods


njc
@compay
Follow
"Give a man a program, frustrate him for a day. Teach a man to program, frustrate him for a lifetime." Who said that first? It's brilliant.
11:21 AM - 8 Oct 2010
2 Retweets
2



Original source of the quote unknown. This is the oldest tweet I could find.

In short, the course is not about "getting better at R".

It is about getting familiar with programming for *data science* to get a lot of different things done – that are hard or impossible without *programming methods*.

Even if programming is occassionally frustrating :)

In a nutshell

- shell, git and sql *foundations* we need
- *core focus* on R programming concepts and techniques
  - language essentialis and types, flow control, loops, ...
  - data input / output, data wrangling
  - visualization and reporting
- reproducibility and best practices:
  - git for version control,
  - markdown for documentation and dissemination,
  - docker for deployment
  - plus some C++ and Rcpp

STAT430.COM WEBSITE: YOUR MAIN SOURCE

ILLINOIS

DSPM :: STAT430 - Data Science Programming Methods - Fall 2019 - Google Chrome

DSPM :: STAT430 - Data Sc

https://stat430.com

ILLINOIS

Search...

- General Overview
- Syllabus
- Lectures
- Schedule
- Resources
- Frequently Asked Questions
- Changelog

Built by Dirk Eddelbuettel using Hugo and Learn, and hosted in this GitHub org.

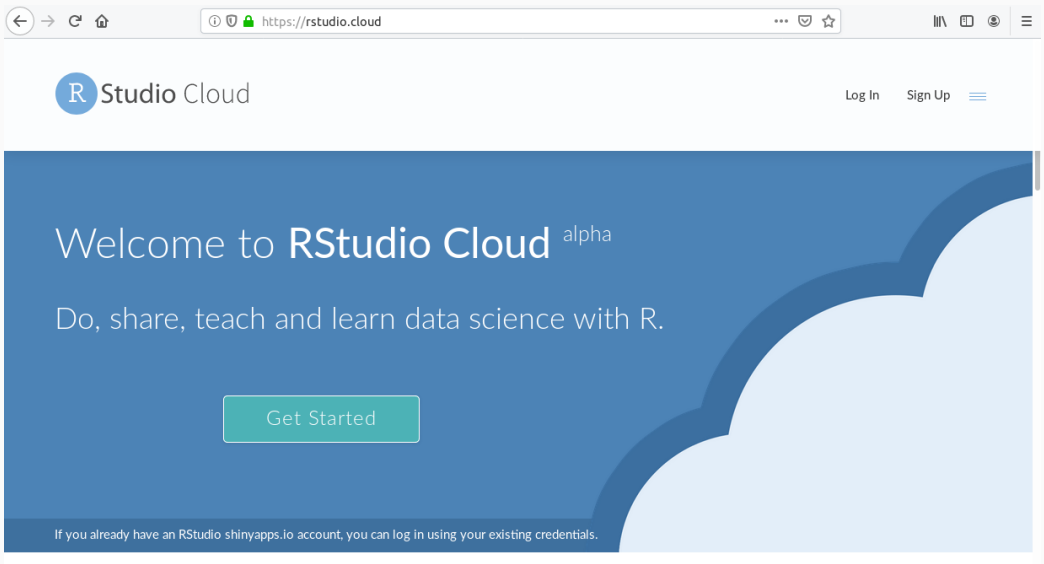
STAT 430: Topics in Applied Statistics

Fall 2019: Data Science Programming Methods

Dirk Eddelbuettel, Department of Statistics, University of Illinois

Image by Negative Space Ku via pexels.com under a 'no attribution required' and 'free to use' license.

Some Requirement: RStudio Cloud

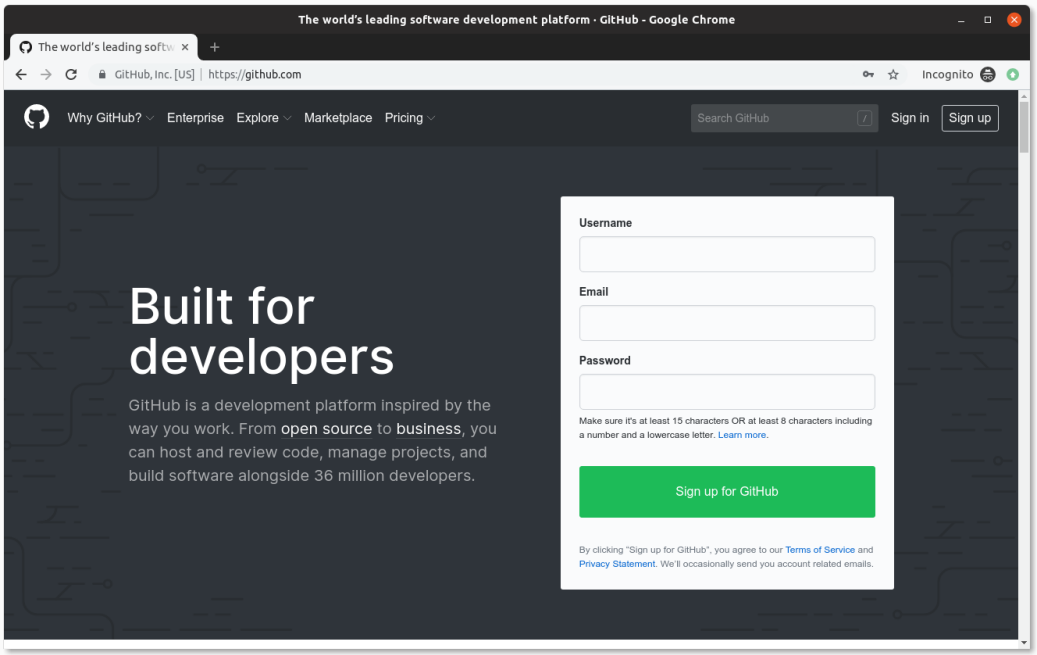


Yes, the actual URL is rstudio.cloud

Use this link to sign up for *our STAT430 DSPM Fall 2019* workspace.

See video 1 for a first introduction about how to log in, create a project and more, and video 3 for the workspace.

Some Requirements: GitHub



Get a GitHub account

Best to use your netid email (you get some .edu benefits).

Then tell us the pair of <netid, github_id> so that we can add you to the course project.

Use this Google Form with your netid to add your Google account.

Shell

Why do we start with the shell ?

- Common backbone of *all* computing
  - On a super-computer: often via a remote connection
  - Cloud computing: same, though you may also see GUIs
  - Workstation: can always see a shell if you know where to look
  - Desktop and laptop: same, generally present or installable
  - Phone: can have a shell
- It is everywhere and underpins many computing concepts
- Its usage concepts translate into almost everything

Why do we do this?

- Key insight: Everything (well, mostly) is a file
- Most direct way to interact with a computer
- Very useful for debugging when (not *if*) problems occur
- Learn to manipulate files *efficiently* and *effectively*

Four key things computers do:

- run programs
- store (and/or retrieve) data
- interact with other computers
- interact with us

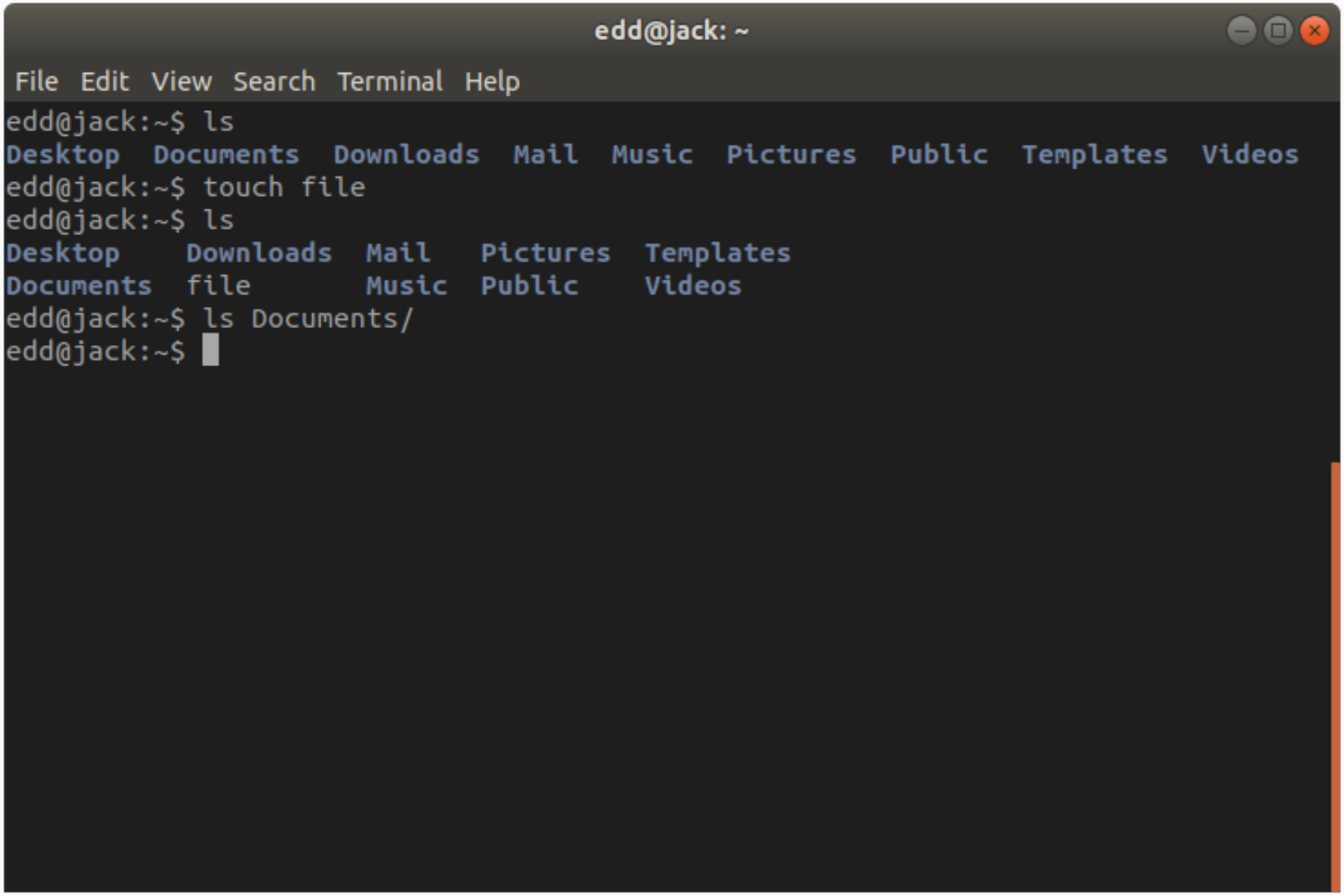
The shell is a common theme and connection.

Read-Evaluate-Print

- Read a command
- Evaluate it (and maybe execute something)
- Print the result

Also called REPL with *L* for loop as the REP parts are repeated.

Basic Interface



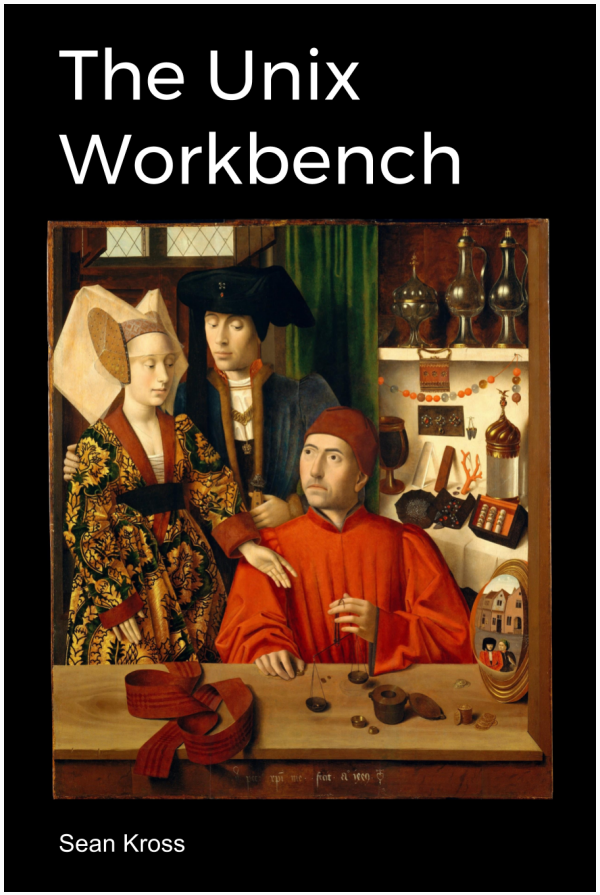
Shell and Bash are (almost) everywhere

- macOS: open terminal app
- Linux: open a terminal or xterm or ...
- Windows: always a bit more work, but tools exist
  - install gitbash via Software Carpentry installer from `http://installation.software-carpentry.org/`
  - video describing the steps linked on that page, or at `https://www.youtube.com/watch?v=339AEqk9c-8`
  - this will be helpful for the `git` section too
  - the `nano` editor may be useful too
  - or *maybe* try WSL / WSL2 (no support from us)
- ... but it easier for us as we discuss on the next slide
- *Nice to have* shell on your computer but course has fallback

Or just go to **https://rstudio.cloud**

- use this link to sign up and login via webportal
  - you can use an existing Google/Gmail or GitHub account
  - you should be able to use Google Auth with your NetId (new)
  - or you can always create a new account
- start a new project
- open console prompt and you have shell access
- see the example in video1

We will use it for most of the course.



- The Unix Workbench is a free / open source book
- From an R User / Data Scientist, source available
- Does not perfectly map out topics and order but helpful
- At https://seankross.com/the-unix-workbench/
- Also at http://leanpub.com for purchase and a Coursera course

A few key commands

Ten Useful Commands and Constructs

ILLINOIS

- `ls` list files (and / or directories)
- `cat` concatenate files or just display them
- `pwd` print working directory ("where am I")
- `cd [dir]` change directory (or back to home)
- `c1 | c2`, `> file`, `< file` redirection
- `*` wildcard character (for listing or file ops)
- `cp` copy one or more file(s) or directories
- `rm` remove (ie delete) one or more files and directories
- `mv` move file(s) or directories
- `grep` pattern matching

Source: Abrahams and Larson, *Unix for the Impatient*, 2nd ed, Section 1.6.3.

`ls` : list files

- Useful options
  - `ls -F` to list types
  - `ls -a` to list *all* files and directories (even if starting with a dot)
  - `ls -l` to list in *long* format
  - `ls -h` to list in *human* units
  - `ls -r` to *reverse* sort order
  - `ls -t` to sort by *time* (rather than name)
  - `ls -S` to sort by *size* (rather than name)
  - `ls -R` to *recurse* and report whole directory tree
- Options can be combined: `ls -alrth`

`cat` : concatenate files

- Simplest:
  - `cat filename`
  - This just displays to screen, or 'stdout'
  - Useful with redirection (later)
- Several files:
  - `cat fileA fileB fileC`
  - "conCATenates" all of them
- Related to output of one command can be input of next (later)

pwd : print working directory

- A text terminal or console is (generally) not graphical
- We sometimes need to "know where we are"
- **pwd** displays (or 'prints') the current working directory
- Related to notion of directories as a graph structure, or "tree"

`cd` : change directory

- important notion of *relative* and *absolute* paths
  - `cd ../work` changes 'up and then into `work/`' relative to the current directory
  - `cd /tmp` changes from anywhere to the *absolute* path for directory `tmp` which itself hangs off the root directory `/`
- `..` denotes the directory above us ("parent directory"),
- `.` is the current directory, useful e.g. when copying (cf below)

`cd` : change directory

- this seemingly small difference is of outmost importance
  - an *absolute* target is fixed and constant
  - ie `cd /tmp` will always switch to the `/tmp` dir
  - a *relative* target changes depending on your current location
  - `cd ../..` changes up two levels, but where you end depends on where you started from
- helpful shortcuts:
  - `cd` returns you to 'home' for which ~ is a shortcut
  - `cd -` changes back to the last directory you were in

Exercise: Starting from `/Users/amanda/data/`, which of the following commands could Amanda use to navigate to her home directory, which is `/Users/amanda` ?

1. `cd .`
2. `cd /`
3. `cd /home/amanda`
4. `cd ../..`
5. `cd ~`
6. `cd home`
7. `cd ~/data/..`
8. `cd`
9. `cd ..`

Input, output and redirection for composition

- Output of one program can be input of the next
- That allows *sequences* or pipes to be composed
- The `>` symbol directs output to a file
- The `<` symbols takes input from a file
- The `|` ("pipe") combines
- Example: `ls -l *txt | head`
- Lists all txt files, and then shows first ten
- Use pipe to paginate help, ie `ls --help | less`
- Also: `cmd > file` *redirects* output from `cmd` into `file`
- Note that a single `>` *overwrites* whereas `>>` *appends*
- And: `cmd < file` *takes input* from `file` for `cmd`

`*` as wildcard

- Commands often work on files, and we list those files
- Easier to 'group' several:
  - instead of `ls file1 file2 file3`
  - We just do `ls file*`
- The `*` wildcard matches 'zero or more characters
- Can combine: `ls abc*_2018-*.txt` matches files
  - starting with `abc` but
  - also containing `_2018-`
  - and ending in `.txt`
- Another wildcard is `?` which matches any *single* character
- And another is `[abc]` to list permitted characters

`cp`: 'copy' files

- absolute or relative addressing, use of wildcard common
- when copying or moving several files, the last argument must be a directory:
  - `cp a.txt b.txt c.txt some/dir/`
  - equivalent: `cp [abc].txt some/dir/` (not a wildcard but an efficient enumeration)
  - *maybe* equivalent: `cp *.txt some/dir` (depends if other files matched)
- options `-f` to 'force' and `-v` for 'verbose' mode, many more options

`rm`: 'remove' files

- very similar to `cp`: address one or multiple files via wildcards
- `rm somefile.txt someother.csv` removes these two files
- wildcards help and be combined with paths:
  - `rm /tmp/testFile*.log` removes all matching files in `/tmp`
- `rm -r` recurses into directories: use it to delete whole trees (*careful*); `rm -f` forces operations (*careful*)
- Generally speaking, a removed file *cannot* be recovered
- good habit:
  - "dry run" of `ls some file here`
  - once correct file shown replace `ls` with `rm`
  - Hint: cursor-up accesses the previous command, history is searchable and more

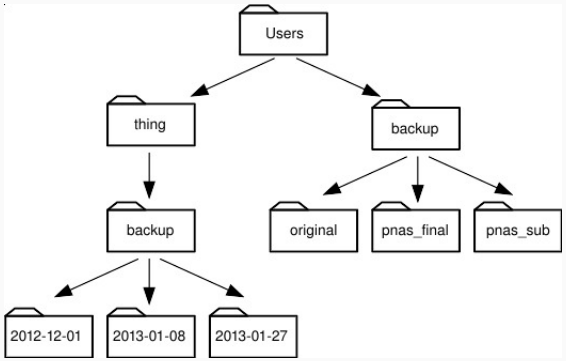
`mv`: 'move' files

- `mv` combines `cp` and `rm`
- used to move content or rename
- ie rename via `mv file.txt oldfile.txt`

`mkdir` and `rmdir`: 'make directory' and 'remove directory'

- same use of absolute and relative paths
- `mkdir ../some/new/dir` creates `dir` in `../some/new` (which must exist)
- `rmdir sub/dir` removes directory `dir` from within `sub/`
- as with other commands, multiple arguments work: `mkdir a b c` creates three directories
- `mkdir -p some/nested/directory` will make non-existing directories `some/` and `nested/` if needed

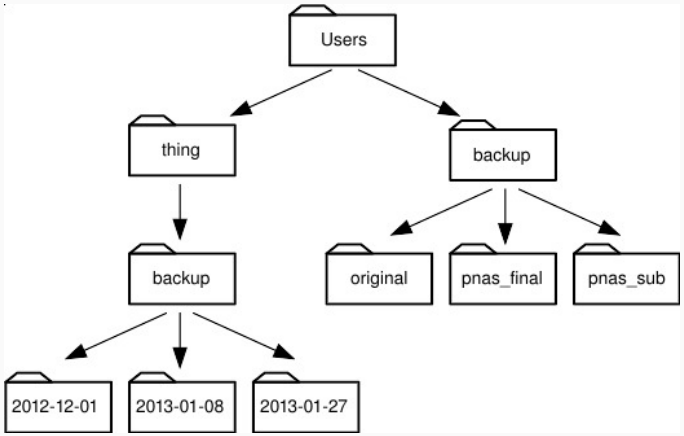
Exercise: Using the filesystem diagram below, if `pwd` displays `/Users/thing`, what will `ls -F ../backup` display?



Source: Software Carpentry shell lecture

1. `../backup: No such file or dir.`
2. `2012-12-01 2013-01-08 2013-01-27`
3. `2012-12-01/ 2013-01-08/ 2013-01-27/`
4. `original/ pnas_final/ pnas_sub/`

Exercise: Using the filesystem diagram below, if `pwd` displays `/Users/backup`, and `-r` tells `ls` to display in reverse order, what command will display `pnas_sub/ pnas_final/ original/`:



Source: Software Carpentry shell lecture

1. `ls pwd`
2. `ls -r -F`
3. `ls -r -F /Users/backup`
4. `Either 2. or 3. above, but not 1.`

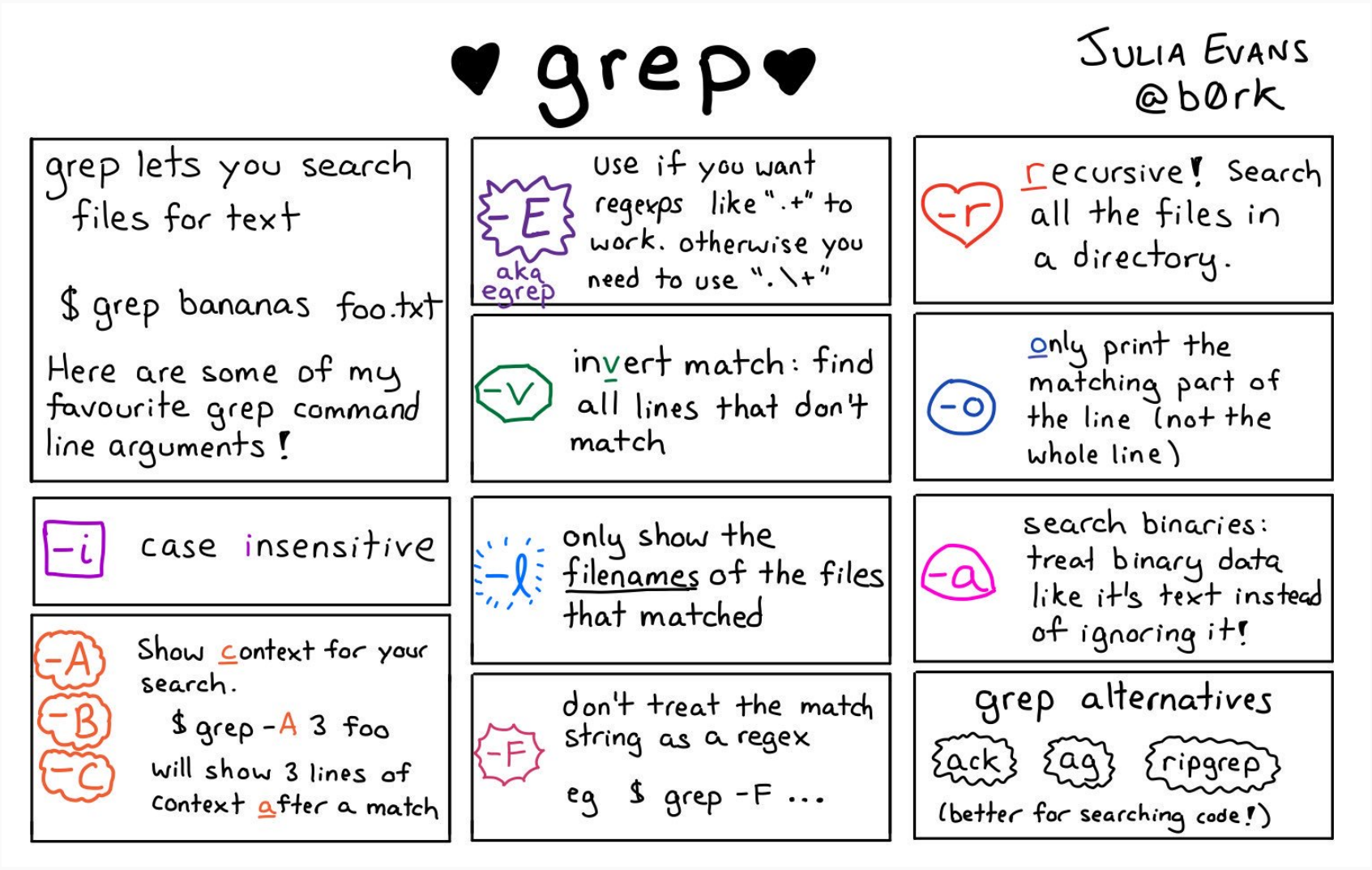
`grep` and regular expressions

- `grep` matches an *expression* in one or more files
- Example (try this from `gapminder/files_unfiltered`):
  - `grep 1992 Oceania/*csv`
  - `grep 1993 Oceania/*csv`
- Key insight: we can run both *at once*:
  - `grep '199[23]' Oceania/*csv`

Note: This jumps ahead a little but the second video shows how to get the data sets.

`grep` and regular expressions

- Useful options for `grep`:
  - `-l` list files
  - `-v` invert selection, ie *not* matching
  - `-q` quiet mode (in scripts)
  - `-C` display more 'context'
- And way more, see next slide



Source: Julia Evans, https://twitter.com/b0rk/status/993862211964735488

Regular expressions

- `grep` uses regular expression: a generalization of `*` wildcards
- Regular expressions can be tricky at first
- They don't have to be if you patiently decompose them
- There is a "grammar" (== comp.sci. speak for it is regular)
- Best may be to find a good tutorial and come back to it

Some miscellaneous commands

- `touch file.txt` creates a (zero-length, ie empy) file
- `head file.txt` displays the first few lines (ten by default)
- `tail file.txt` display the last few lines (ten by default)
- `sort file.txt` sort the content (with options for numeric)
- `uniq file.txt` removes adjacent duplicates
- `wc file.txt` counts characters, words and lines
- `man command` displays the *manual page* entry for command
- `tree dir` can show a recursive directory tree (also: `ls -lR`)

Access rights

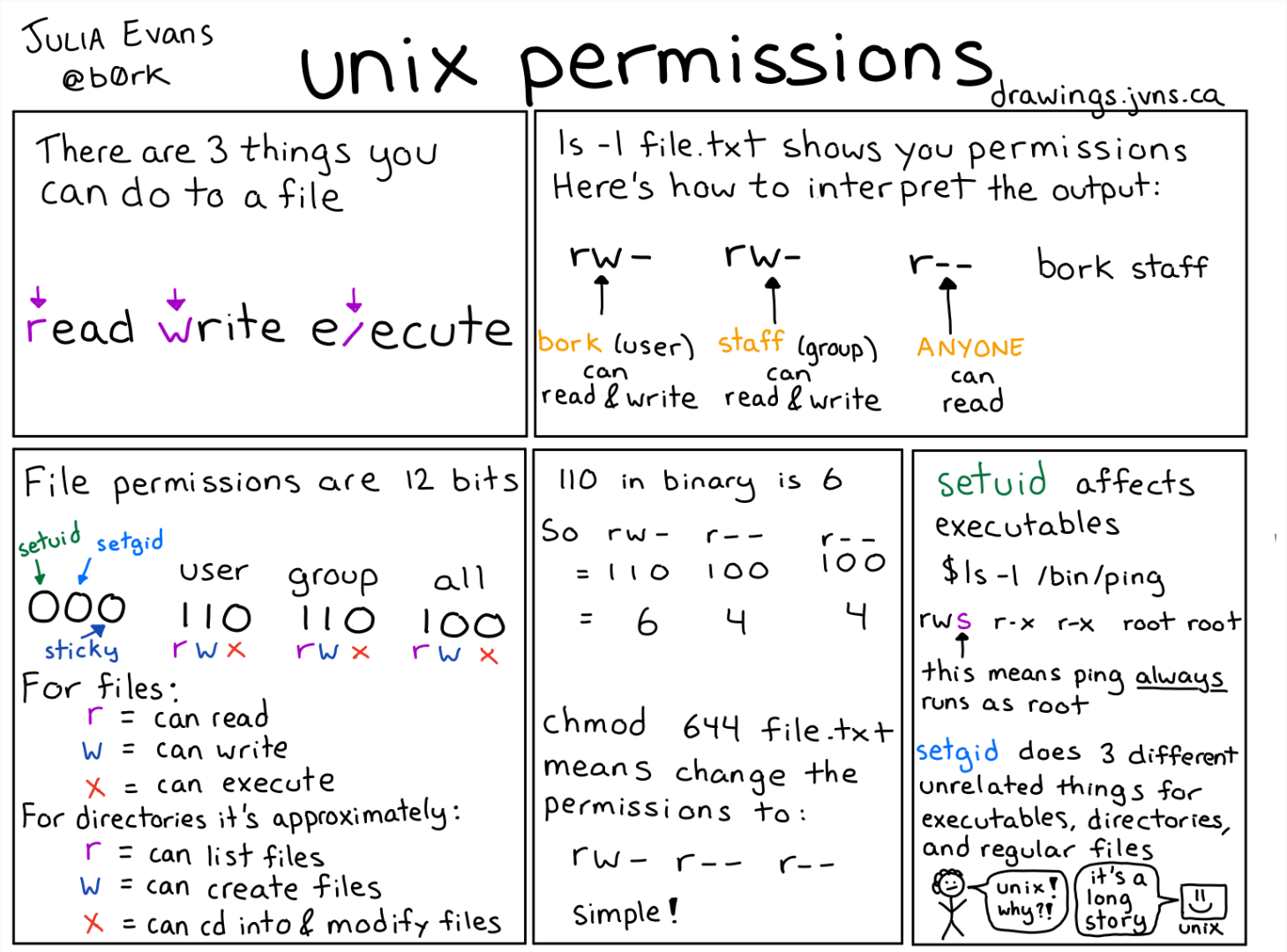
- the different user sets: *owner*, his/her *group* and *everybody else*
- three different access mode: *read*, *write* and *execute*
- for programs, execute means running; for directories right to change into
- the form `_rw_rw_r__` means
  - not a directory
  - read and write for owner and group
  - read-only for everybody else
- numerically this is 100*(4+2)* + *10*(4+2) + 4 = 664
- which is common for a file

Access rights

- the form `drwxr_xr_x` means
  - a directory
  - read, write, execute for owner
  - read and execute for group and rest
- numerically this 100*(4+2+1)* + *10*(4+1) + (4+1) = 755
- which is common for a directory

Changing access rights

- Either do `chmod 755 somedir` for compact numeric mode
- Or use `chmod ug+w somefile` for
  - *who* (one or more of `ugo`) followed by
  - *operator* `+` to add or `-` to remove and
  - *what* which is one or more of `rwx`



Source: https://drawings.jvns.ca/drawings/unixpermissions.png

Unix and Shell can be tricky

Ways to get help include

- adding `--help` to a command (try it, *e.g.* `ls --help`)
- invoking the manual page command: `man ls`
- of course Google and StackOverflow ...
- as manual pages tend to be too long and detailed:
  - `http://tldr.sh` – "too long, didn't read"
  - and similar options
- search, make notes, write them down
- save examples

The gapminder data set is data set made famous by a presentation of the late Swedish statistician / health scientist Hans Rosling. See the short video (link also at website) for more.

There is an R package on CRAN with the data set, and I used a few lines of code to disaggregate the data again into per-country files in per-continent directories.

See video2 (and the next few slides) for accessing the expanded files.

Via the Website

- A compressed `tar` archive with files is available.
- Create new directory, change into it, download and unpack:

```
mkdir stat430   # or another name
cd stat430
wget http://stat430.com/data/stat430_data.tar.gz
tar xfz stat430_data.tar.gz
```

You should now have a directory `examples` with three subdirectories: `gapminder/`, `sql/` and `titanic/`.

Also see the walk-through in video 2.

Via GitHub and `git clone`

- Go to the repo webpage
- Select the green 'Clone or Download' button
- Make sure 'Clone with https' is selected
- Copy the URL via the copy button next to the URL
- Now make a new directory, change into it and paste the URL

```
mkdir stat430   # or another name
cd stat430
git clone https://github.com/eddelbuettel/data-examples.git
```

The new directories should be created.

Via GitHub and 'New Project'

- Go to your RStudio Cloud 'Your Workspace' screen
- Select 'New Project' as a dropdown
- Select 'New Project from Git Repo'
- Go to the repo webpage
- Select the green 'Clone or Download' button
- Copy the https URL and paste it into the RStudio input box
- A new project
  - will be created
  - *and* populated with content from the git repo

Also see the walk-through in video 2.

```
head files/Africa/Algeria.csv
```

```
## country,continent,year,lifeExp,pop,gdpPercap
## Algeria,Africa,1952,43.077,9279525,2449.008185
## Algeria,Africa,1957,45.685,10270856,3013.976023
## Algeria,Africa,1962,48.303,11000948,2550.81688
## Algeria,Africa,1967,51.407,12760499,3246.991771
## Algeria,Africa,1972,54.518,14760787,4182.663766
## Algeria,Africa,1977,58.014,17152804,4910.416756
## Algeria,Africa,1982,61.368,20033753,5745.160213
## Algeria,Africa,1987,65.799,23254956,5681.358539
## Algeria,Africa,1992,67.744,26298373,5023.216647
```

Summary

Summary of Lecture 1

ILLINOIS

The shell

- A common interface to all computing machinery
- We saw folders, paths (absolute and relative), wildcards, ...
- We looked at a number of common and important commands
- We discussed permissions for files and directories
- We look at program output as input for other programs: piping
- We learned how to install hold of the example data sets