

# Getting things done on the command line

Eric Franzosa (franzosa@hsph.harvard.edu)

<http://franzosa.net/bst273>

# Overview

- Command-line philosophy
- File systems
- Utils for...
  - Navigation
  - Organization
  - Inspecting files
  - Data manipulation
- Chaining programs
- Practice

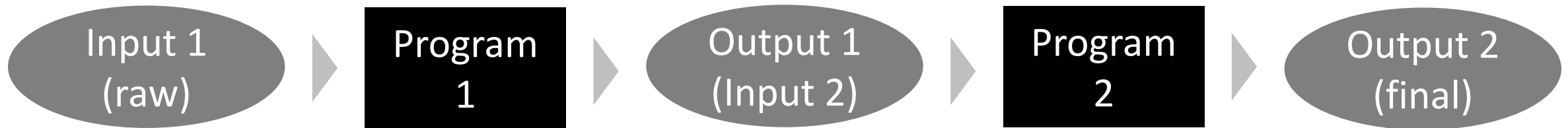
# Some terminology

- A “**command line**” is a text-based interface to a computer.
- The MacOS **Terminal** and Windows **Command Prompt** are programs that provide command-line interfaces to Mac and Windows computers.
- Command-line environments are often referred to generically as “**terminals**” or “**consoles**” (a throwback to their antiquated, physical equivalents).
- A “**shell**” is a program that runs inside a terminal to interpret and execute the commands that you type (~the Python interpreter).
- **Bash** is a popular shell used on Unix-based computers.



# Why work from the command line?

- Faster than clicking around a graphical user interface (GUI).
- Can answer a lot of basic data questions with command-line tools.
- Convenient for working on a remote computer.
- Command-line “chains” provide practice for designing algorithms.



Each command-line program/util is highly optimized for a particular task, e.g. counting lines in a file

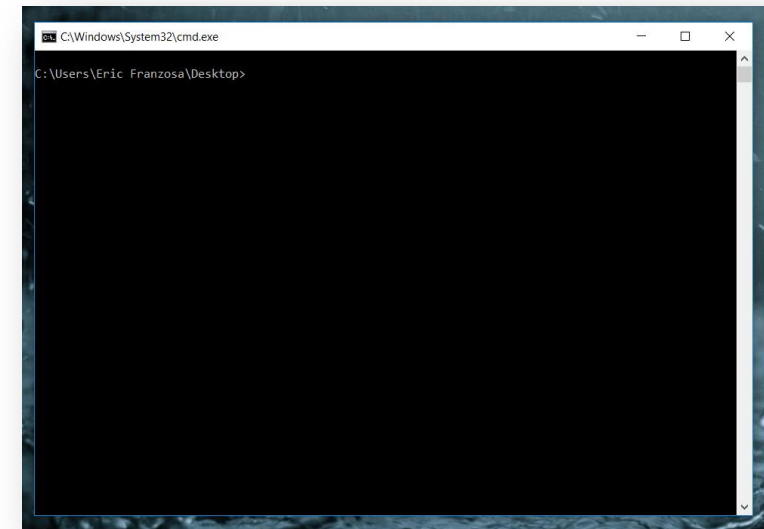
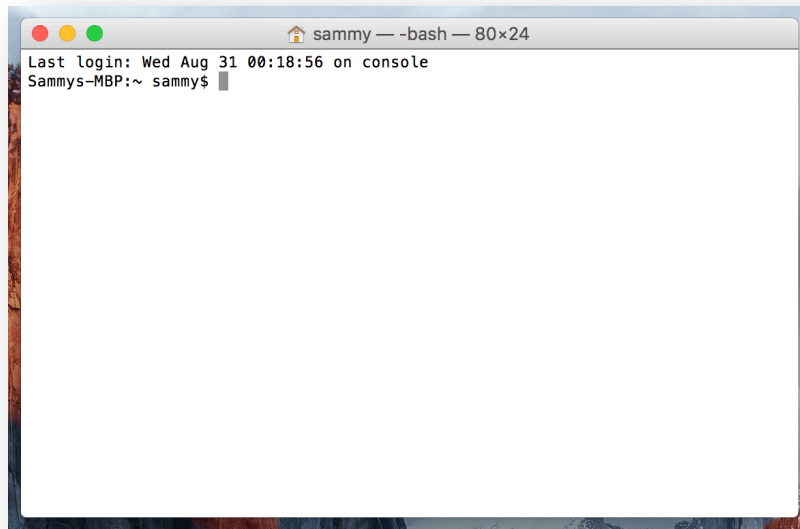
# Notes for Windows users



- The standard Windows Command Prompt is an oddball
  - Uses different names for many of the basic commands (I will point these out)
  - Lacks most of the useful, less-basic commands
- Some alternative options to consider (from least to most complex)
  - **GOW** supplements basic programs and adds in some of the missing ones
    - This is what I use
  - **Windows Powershell** is a built-in alternative Command Prompt
    - “Fixes” some inconsistencies, but not all
  - The **Linux Subsystem for Windows 10**
    - A full-featured Linux shell within Windows 10
  - **Cygwin** is an installable alternative Command Prompt
    - Behaves exactly like other terminals, but a bit harder to set up and use

# Opening a Terminal/Command Prompt

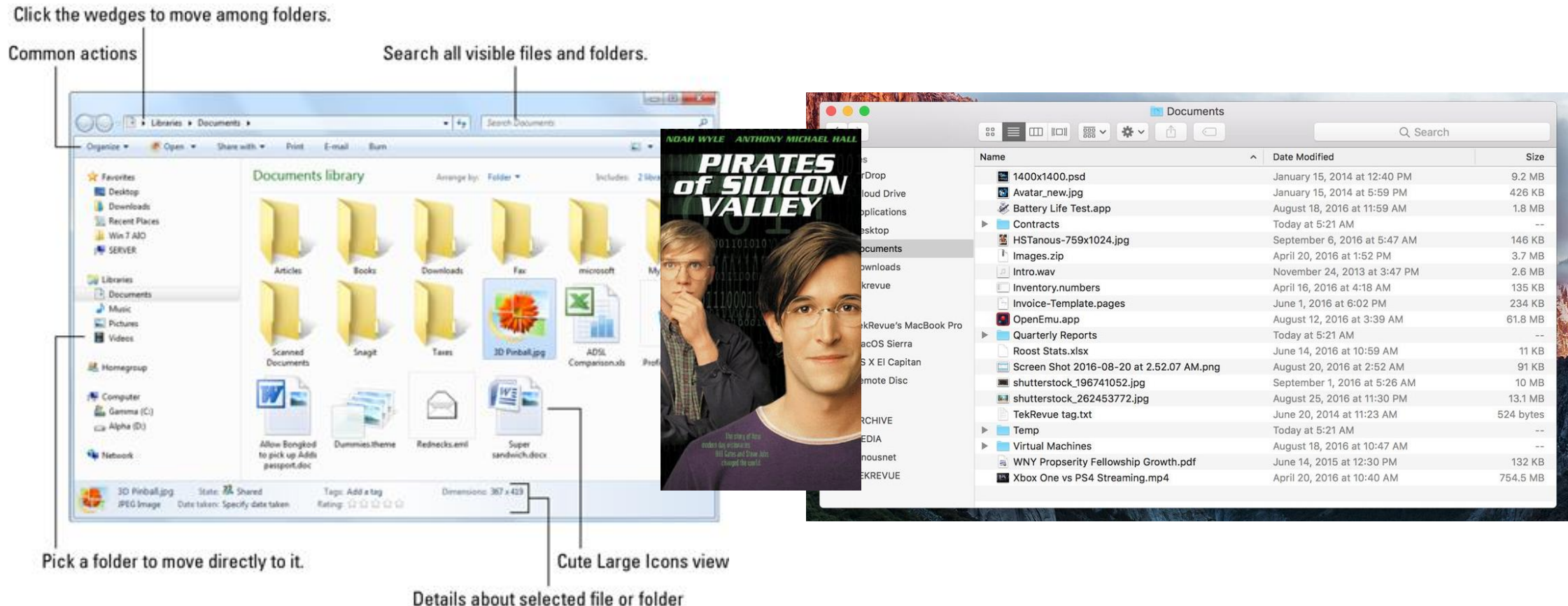
- In MacOS, open a Terminal
  - Spotlight → Type “Terminal” → Select **Terminal**
  - *Option-click Terminal in Dock: “Keep in Dock”*
- In Windows 10, open the Anaconda Prompt
  - Start/Search → Type “Anaconda Prompt” → Select **Anaconda Prompt**
  - *Right-click Anaconda Prompt in taskbar: “Pin to Taskbar”*



# File systems

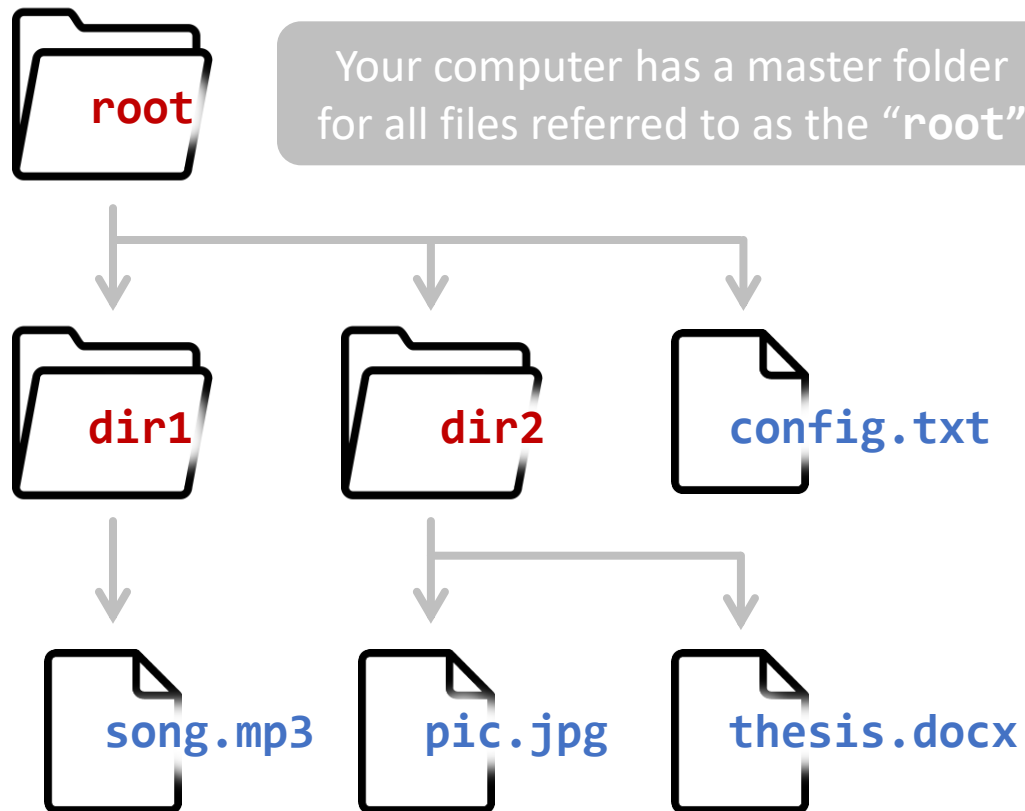
# File systems

- The **MacOS Finder** and **Windows Explorer** provide graphical interfaces to Mac/Windows file systems, respectively.



# File systems

- A file system is a hierarchical (tree) organization of data on your computer.
- Folders contain files (and other folders) in **parent-child relationships**.

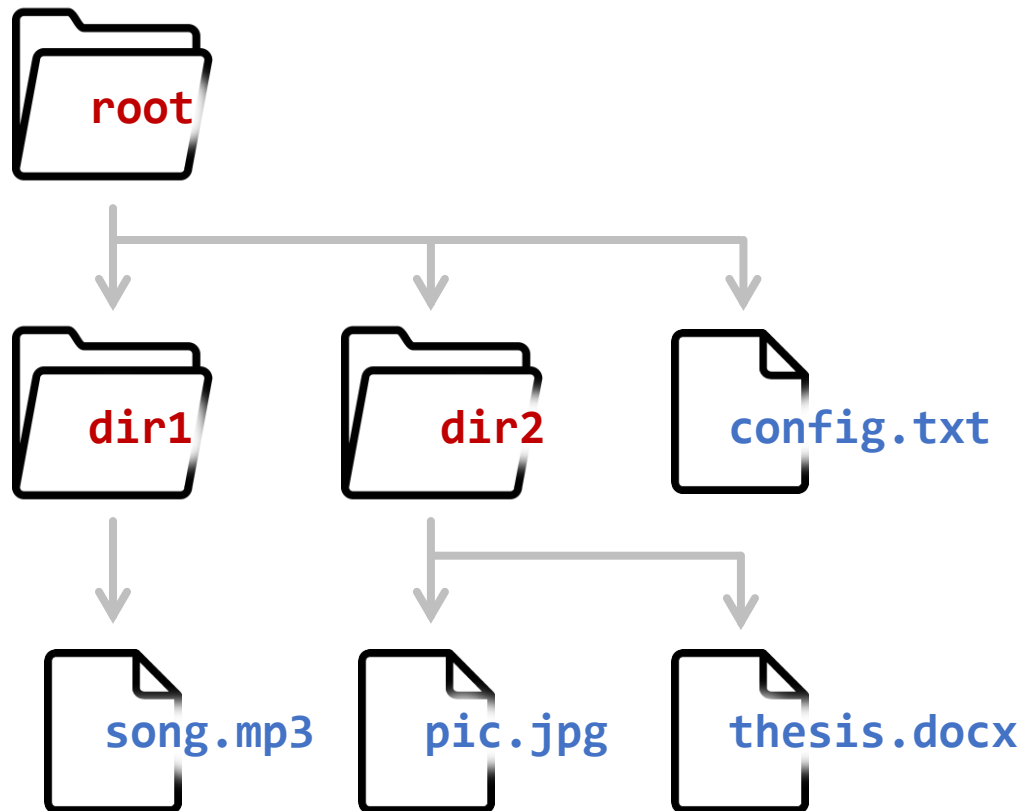


The words "folder" and "directory" are used interchangeably

Files are often distinguished from folders by a file extension (.xyz): a convenience feature for hinting at file contents and associating them with appropriate programs

# File systems

- Another representation of the same data:

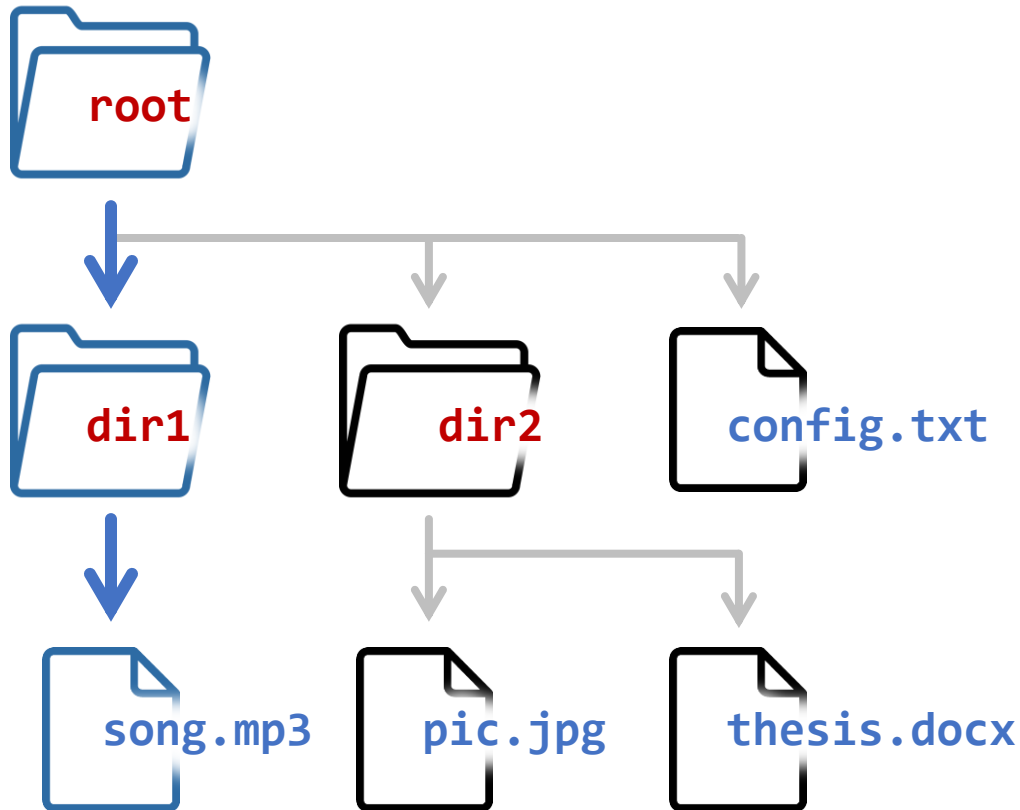


(we'll see that the "tree" command can actually sketch this out for us)

```
root
├── config.txt
├── dir1
│   └── song.mp3
└── dir2
    ├── pic.jpg
    └── thesis.docx
```

# Paths

- There is a path in the tree to every folder or file on the computer.
- This path can be represented as a string of text.



`/dir1/song.mp3`

In most operating systems, the root of the hierarchy is represented by “/” (forward-slash) and the same symbol represents parent-child relationships.

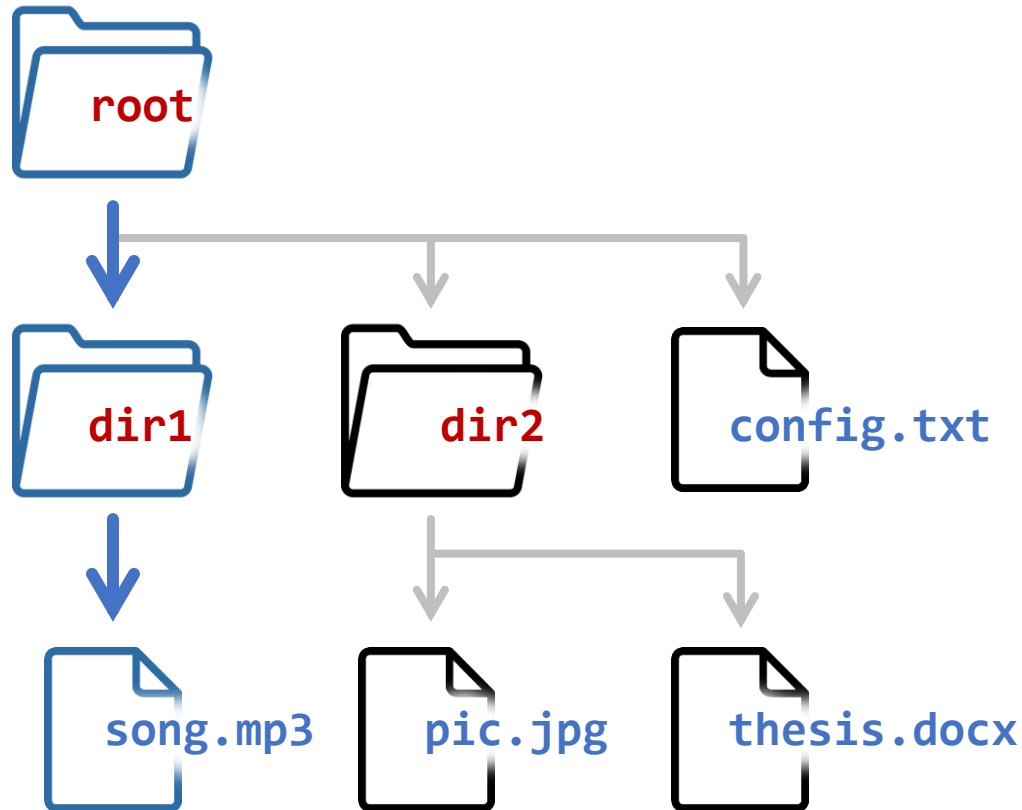
`C:\dir1\song.mp3`

Windows has a separate root for each hard drive disk (HDD), with the default being “C:”. Windows uses “\” (back-slash) for parent-child relationships.



# Paths

- Paths that begin at the root are called “absolute paths”.
- Absolute paths are safer; they don’t depend on “frame of reference”.

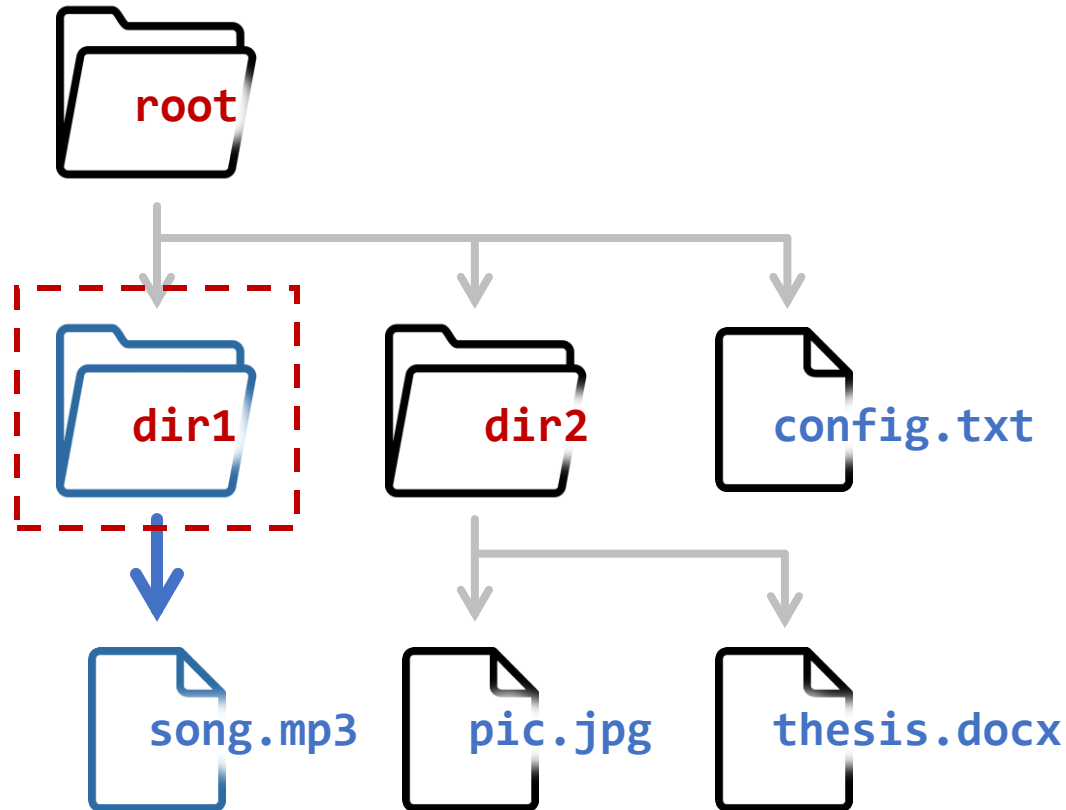


`/dir1/song.mp3`

Absolute path to song.mp3

# Paths

- All other paths are “relative paths”.
- “Relative” → “Relative to your working directory” (more on that shortly).

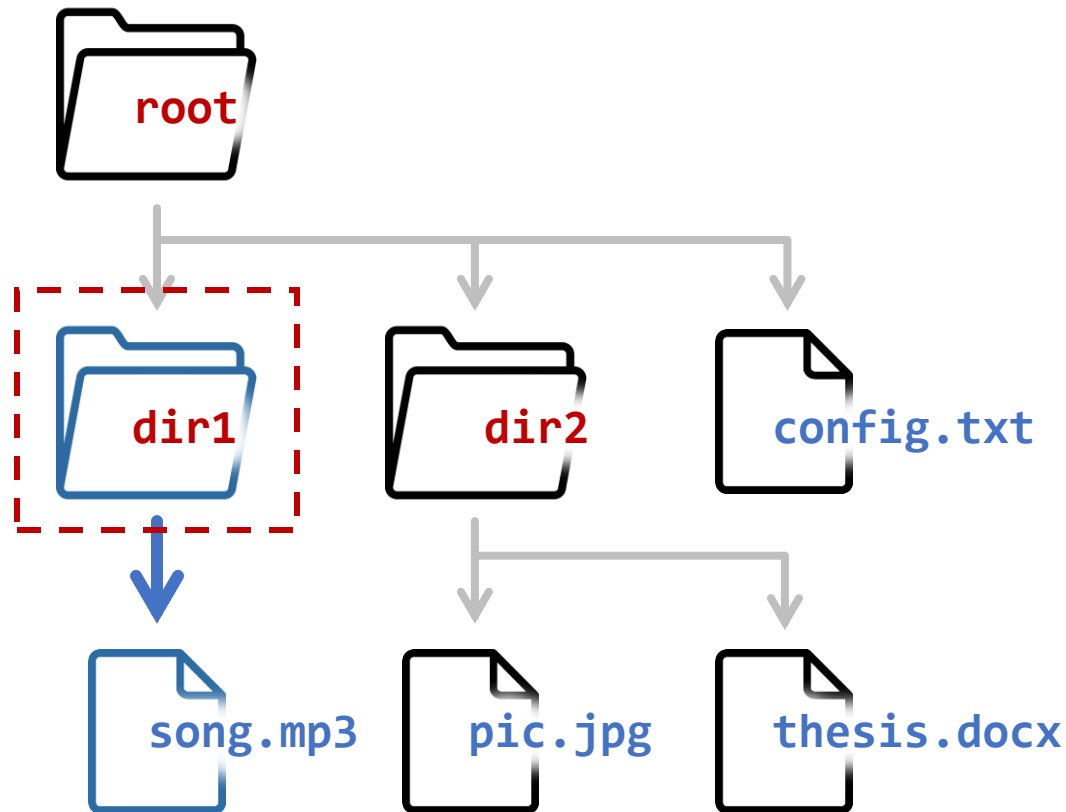


song.mp3

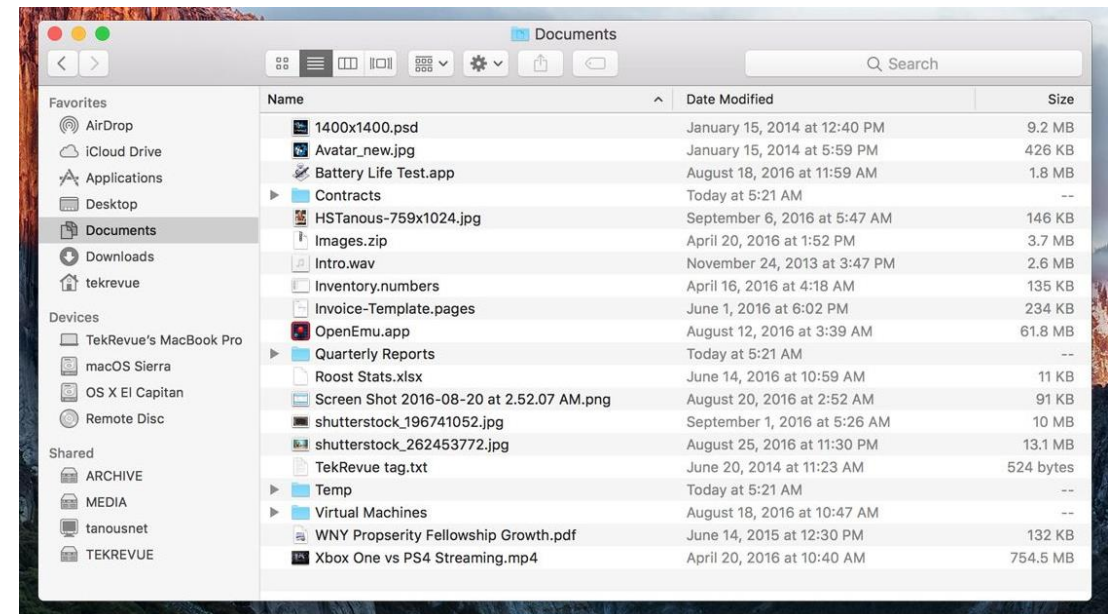
Relative path to song.mp3 if I'm in dir1/

# Paths

- Relative paths exist because the terminal is “in” a given directory at any specific time (the default is usually your home folder, not the root).



This directory is called the “Working Directory”



Analogous to having finder “open” to Documents/

# Misc. notes about file systems / paths

- Mac/Linux distinguish between upper vs. lowercase letters (Windows doesn't)
- Your **home folder** is usually a step below the root
  - `/home/franzosa` on Mac/Linux
  - `C:\Users\Franzosa` on Windows
- On Mac/Linux, `~` is a shortcut for the absolute path to your home directory
  - `~/Downloads` is equivalent to `/home/franzosa/Downloads`
- On all systems, `..` is a shortcut for “**my parent folder**”
  - Working in `~/Downloads/`, `..` is equivalent to `/home/franzosa` or `~`
- On all systems, `.` is a shortcut for “**here**” (i.e. the **working directory**)
  - Working in `~/Downloads/`, `.` is equivalent to `~/Downloads`

Getting help & tricks

# Command line utilities: man

## What does it do?

Displays a “manual” page for another tool, detailing expected input data and optional flags

## Usage

```
$ man less
```

| Options | What does it do? |
|---------|------------------|
|---------|------------------|

|   |                                             |
|---|---------------------------------------------|
| Q | [within program] Exits the current man page |
|---|---------------------------------------------|



\*The Windows equivalent is  
`$ help PROGRAM (or) $ PROGRAM --help`

# Command line utilities: man

LESS(1)

General Commands Manual

LESS(1)

## NAME

less - opposite of more

## SYNOPSIS

```
less -?
less --help
less -V
less --version
less [-[+]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWx~]
    [-b space] [-h lines] [-j line] [-k keyfile]
    [-{oO} logfile] [-p pattern] [-P prompt] [-t tag]
    [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines]
    [-# shift] [+][+]cmd] [--] [filename]...
(See the OPTIONS section for alternate option syntax with long option names.)
```

## DESCRIPTION

Less is a program similar to more (1), but it has many more features. Less does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi (1). Less uses termcap (or terminfo on some systems), so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top of the screen are prefixed with a caret.)

Commands are based on both more and vi. Commands may be preceded by a decimal number, called N in the descriptions below. The number is used by some commands, as indicated.

## COMMANDS

In the following descriptions, ^X means control-X. ESC stands for the ESCAPE key; for example ESC-v means the two character sequence "ESCAPE", then "v".

h or H Help: display a summary of these commands. If you forget all the other commands, remember this one.

SPACE or ^V or f or ^F

Scroll forward N lines, default one window (see option -z below). If N is more than the screen size, only the final screenful is displayed. Warning: some systems use ^V as a special literalization character.

z Like SPACE, but if N is specified, it becomes the new window size.

# Command-line shortcuts

- Press the “UP” arrow to cycle through previously executed commands
  - Don’t retype commands unless they are very short
- Press <TAB> to auto-complete program names/paths from a prefix
  - For example, type “/home/doc” and press <TAB> to fill “/home/documents”
  - Mac/Linux will list auto-complete options (if >1)
  - Windows will cycle through them as you click <TAB> repeatedly

# Navigation

(pwd, ls, cd)

# Command line utilities: pwd

## What does it do?

Prints the current working directory

## Usage

```
$ pwd
```

Note: “\$” is used to specify the start of a line on a generic command prompt.



\*The windows equivalent is cd

# Command line utilities: ls

## What does it do?

List files in the current directory

## Usage

```
$ ls          # list all files in current directory
```

```
$ ls *.txt    # list all text files
```

| Options | What does it do?                                   |
|---------|----------------------------------------------------|
| -l      | List file details                                  |
| -h      | Human-readable file sizes (e.g. 3.3MB vs. 3354123) |



\*The windows equivalent is dir

# Command line utilities: cd

## What does it do?

Changes directories

## Usage

```
$ cd /home/docs # follow the absolute path to /home/docs
```

```
$ cd docs      # go to subfolder docs in working directory
```

```
$ cd ..       # go to parent folder of working directory
```

# Organization

(rm, cp, mv, mkdir)

# Command line utilities: rm

## What does it do?

Deletes (removes) a file **PERMANENTLY**

## Usage

```
$ rm file # removes file from the working directory
```

| Options | What does it do?                                |
|---------|-------------------------------------------------|
| -r      | Force-remove a directory (will fail by default) |
| -i      | Confirm each deletion event                     |



\*The windows equivalent is `del` for files and `rmdir /S` for folders

# Command line utilities: cp

## What does it do?

Copy a file

## Usage

```
$ cp file1 file2      # copy file1 to file2 in working directory
```

```
$ cp /home/a /home/b # copy a to b (wherever you are)
```

```
$ cp /home/a ../a     # copy a to my parent folder
```

| Options | What does it do?                                  |
|---------|---------------------------------------------------|
| -i      | Warn before overwriting a file                    |
| -r      | Copy recursively (needed for copying directories) |



\*The windows equivalent is copy

# Command line utilities: mv

## What does it do?

Move a file to a new location (or rename a file)

## Usage

```
$ mv file1 file2      # rename file1 as file2 in working directory
```

```
$ mv /home/a /home/b # move a to b (wherever you are)
```

```
$ mv /home/a ../a     # move a to my parent folder
```

| Options | What does it do?               |
|---------|--------------------------------|
| -i      | Warn before overwriting a file |



\*The windows equivalent is move

# Command line utilities: mkdir

## What does it do?

Make a directory

## Usage

```
$ mkdir /home/docs # make the directory docs/ in /home/
```

```
$ mkdir docs      # make docs/ in working directory
```

```
$ mkdir ../docs   # make docs/ in parent directory
```

| Options | What does it do? |
|---------|------------------|
|---------|------------------|

|    |                                                       |
|----|-------------------------------------------------------|
| -p | Create non-existent parents in the path to new folder |
|----|-------------------------------------------------------|

# Words of warning

- Manipulating files from the command line is a lot less forgiving than working in the graphical MacOS and Windows file browsers.
- **!!! By default, no warnings/confirmation for deleting/overwriting files !!!**
  - The system will prevent you from deleting “important” file (i.e. files required to keep your computer running, but not “my\_thesis.docx”)
- As you are practicing command-line skills, I recommend working within a folder of (e.g.) Dropbox or Google Drive so you can restore deleted files.
- Programmers generally use backups/snapshots and revision control systems (e.g. git/github) to restore work after making mistakes.

# Inspecting files

(less, head/tail, cat)

 No direct Windows equivalents from here on ☹️

# Command line utilities: less

## What does it do?

View a file or data stream with navigation options

## Usage

```
$ less my_file
```

```
$ cat *.txt | less # 'less' is often used at the end of a chain
```

| Options | What does it do?                                                           |
|---------|----------------------------------------------------------------------------|
| -S      | Don't wrap long lines                                                      |
| Q       | [while running] Exit the program                                           |
| ←↑→↓    | [while running] Arrow keys to navigate in file (PageUp, PageDown work too) |
| /       | [while running] Search for text, use n and N to see next/previous matches  |

# Command line utilities: head/tail

## What does it do?

Stream the first/last (tail/head) lines of a data stream (default 10)

## Usage

```
$ head my_file # print lines 1-10
```

```
$ tail my_file # print last 10 lines of file
```

```
$ head -100 my_file | tail # print lines 91-100
```

| Options | What does it do?                    |
|---------|-------------------------------------|
| -N      | Print N lines instead of default 10 |

# Command line utilities: cat

## What does it do?

Concatenates (prints) one or more files (often for feeding into another program)

## Usage

```
$ cat my_file # lines of file scroll over screen
```

```
$ cat my_file | program # send contents of file to program
```

```
$ cat *.txt | program # send contents of all text files
```

# Data manipulation

(grep, cut, sort, uniq, wc, sed)

# Command line utilities: grep

## What does it do?

Isolate lines of a data stream (file or STDIN) that match a pattern

## Usage

```
$ grep pattern my_file
```

```
$ cat *.txt | grep pattern
```

| Options | What does it do?                                                               |
|---------|--------------------------------------------------------------------------------|
| -P      | Richer pattern options (regular expressions); more on these in a later lecture |
| -v      | Isolate lines that DO NOT match the pattern (invert the match)                 |
| -i      | Case-insensitive match                                                         |
| -f      | Specify a file of patterns to match (slow if there are lots of options)        |

# Command line utilities: cut

## What does it do?

Isolate tab-delimited columns of a data stream. First column is #1 (not #0 as in Python).

## Usage

```
$ cut -f2 my_file # isolate the 2nd column of the file
```

```
$ cut -f2,3 my_file # isolate columns 2 and 3
```

```
$ cut -f2-5 my_file # isolate columns 2 THROUGH 5
```

```
$ cut -f3- my_file # isolate columns 3 to END (python [2:] slice)
```

| Options | What does it do? |
|---------|------------------|
|---------|------------------|

|    |                         |
|----|-------------------------|
| -f | Select columns (fields) |
|----|-------------------------|

|                    |                                                                                    |
|--------------------|------------------------------------------------------------------------------------|
| -t ' <i>char</i> ' | Break columns on the specified character instead of tab, e.g. -t ',' for .csv file |
|--------------------|------------------------------------------------------------------------------------|

# Command line utilities: sort

## What does it do?

Sort the lines of a data stream (alphabetically)

## Usage

```
$ sort my_file
```

| Options            | What does it do?                                                  |
|--------------------|-------------------------------------------------------------------|
| -r                 | Reverse the sort                                                  |
| -k <i>N</i>        | Sort on the value of <b>WHITESPACE</b> -delimited column <i>N</i> |
| -t ' <i>char</i> ' | Specify the delimiter character (e.g. -t '\t' for tab)            |
| -n                 | Perform a numeric sort (otherwise 10 comes before 2)              |

# Command line utilities: `uniq`

## What does it do?

Isolate the unique **ADJACENT** lines of a data stream

## Usage

```
$ sort my_file | uniq # w/o sorting, non-adjacent repeats missed
```

| Options | What does it do? |
|---------|------------------|
|---------|------------------|

|                 |                                          |
|-----------------|------------------------------------------|
| <code>-c</code> | Count the occurrence of each unique line |
|-----------------|------------------------------------------|

# Command line utilities: wc

## What does it do?

Counts the lines, words, and characters of a data stream

## Usage

```
$ wc my_file
```

| Options | What does it do?                                        |
|---------|---------------------------------------------------------|
| -l      | Only report line count (faster, and often all you want) |
| -w      | Only report word count                                  |
| -c      | Only report character count                             |

# Command line utilities: sed

## What does it do?

Edit a data stream, most often used for find/replace operations

## Usage

```
$ sed "s/find/replace/g" my_file
```

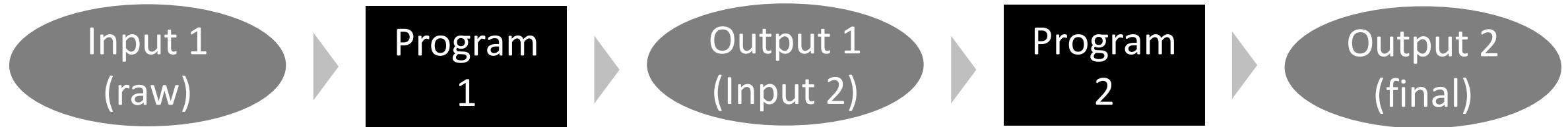
```
$ sed "s/apple/banana/g" my_file # replace all instances of "apple" with "banana"
```

| Options | What does it do?                                                                                                                                                                |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -i      | Edit file "in place" (use with caution)                                                                                                                                         |
|         | " <i>find</i> " can be a regular expression, and " <i>replace</i> " can use captured elements of the pattern (this will make more sense after our regular expressions lecture). |

Chains / Pipes

# Chaining commands

- We can “chain” the output of one command as the input of the next:



- This is a good model for how we’ll build programs later in the course: with the output of one piece of code feeding into another piece of code.

# Chaining commands with pipe, |



words.txt

```
contralto  
dominion  
halcyon  
intercalate  
paradigm  
perceptron  
raconteur  
syntax
```

```
$ head -4 words.txt
```

```
contralto  
dominion  
halcyon  
intercalate
```

```
$ grep "y" words.txt
```

```
halcyon  
syntax
```

```
$ head -4 words.txt | grep "y"
```

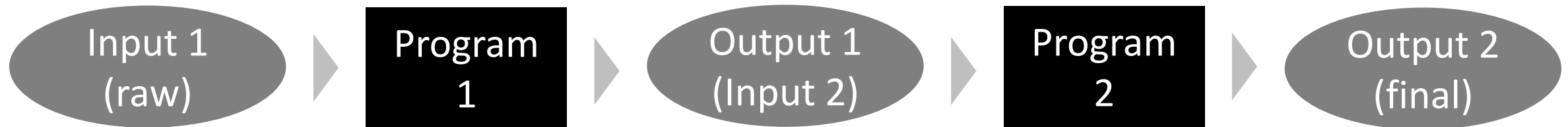
```
halcyon
```

# Starting a chain

- Start with first command naming the input file (`my_file.txt`):
  - `grep "Hello" my_file.txt`
- Combine (concatenate) multiple files of the same type:
  - `cat *.txt`
- Another way to provide a single file as STDIN (less common):
  - `grep "Hello" < my_file.txt`

# Extending the chain

- Use the pipe ‘|’ to direct STDOUT of program  $N$  as STDIN of program  $N+1$ 
  - `grep “Hello” my_file.txt | sort`
- Can do this repeatedly
  - `grep “Hello” my_file.txt | sort | uniq | wc -l`



# Ending the chain

- Don't do anything special, results print to screen:
  - `grep "Hello" my_file.txt | sort | uniq | wc -l`
- View the final results in `less`:
  - `grep "Hello" my_file.txt | sort | uniq | wc -l | less`
- Write the results to a file:
  - `grep "Hello" my_file.txt | sort | uniq | wc -l > hello_count.txt`
  - **!!! WARNING: if "hello\_count.txt" exists, it will be overwritten !!!**

Practice:  
Puzzles with demo.zip

# demo.zip

- Download the archive demo.zip from Canvas
- Open a terminal and use `cd` to navigate to demo.zip
- Unzip the archive by executing `unzip demo.zip`
- Use the file tree in the corner as a reference

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

# Part 1: More words

1. Use `cd` to navigate to the location of `wikipedia_top100.txt`
2. Inspect the contents of the file with `less`
3. Use `wc` to verify that the file contains 100 words
4. Use `mv` to rename the file `more_words.txt`
5. Use `cd` to return to the `demo/` folder

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

# Part 2: We the people

1. Use `mkdir` to make a new folder called `gov2/`
2. Use `cd` to enter your new folder
3. Use `cp` to make a copy of `us_constitution.txt` in `gov2/`
  1. Hint: you will use the `“.”` and `“..”` tricks
4. Use `grep` to find lines containing the phrase “United States”
5. Then add a pipe and `wc` to count them
6. Use `cd` to return to the `demo/` folder

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

# Part 3: Bug bytes

- Kevin and I do research on the human microbiome
- The file `demo/hmp2012/metadata.tsv` lists metagenomic samples from the first phase of the Human Microbiome Project (HMP)
- Each sample comes from a body site of a particular subject at a given visit



`metadata.tsv`

| #SAMPLE_ID | SUBJECT_ID | VISIT | BODY_SITE               |
|------------|------------|-------|-------------------------|
| 700101852  | 159591683  | 2     | Supragingival_plaque    |
| 700033601  | 159611913  | 1     | Supragingival_plaque    |
| 700032328  | 159915365  | 1     | Supragingival_plaque    |
| 700098450  | 809635352  | 1     | Posterior_fornix        |
| 700016482  | 159207311  | 1     | Tongue_dorsum           |
| 700102585  | 159227541  | 2     | Stool                   |
| 700106465  | 338793263  | 1     | Stool                   |
| 700101840  | 159591683  | 2     | Stool                   |
| 700015965  | 159591683  | 1     | L_Retroauricular_crease |

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

# Part 3.1: Bug bytes

- How many samples were there?
  - Use `wc` and subtract 1 to discount the header row
- How many **Stool** samples were there?
  - Solution combines `grep` and `wc`



metadata.tsv

| #SAMPLE_ID | SUBJECT_ID | VISIT | BODY_SITE               |
|------------|------------|-------|-------------------------|
| 700101852  | 159591683  | 2     | Supragingival_plaque    |
| 700033601  | 159611913  | 1     | Supragingival_plaque    |
| 700032328  | 159915365  | 1     | Supragingival_plaque    |
| 700098450  | 809635352  | 1     | Posterior_fornix        |
| 700016482  | 159207311  | 1     | Tongue_dorsum           |
| 700102585  | 159227541  | 2     | Stool                   |
| 700106465  | 338793263  | 1     | Stool                   |
| 700101840  | 159591683  | 2     | Stool                   |
| 700015965  | 159591683  | 1     | L_Retroauricular_crease |

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

## Part 3.2: Bug bytes

- How many unique body sites were surveyed?
  - Solution uses `cut`, `grep`, `sort`, `uniq`, and `wc`
- How many unique subjects participated?
  - Solution uses `cut`, `grep`, `sort`, `uniq`, and `wc`



`metadata.tsv`

| #SAMPLE_ID | SUBJECT_ID | VISIT | BODY_SITE               |
|------------|------------|-------|-------------------------|
| 700101852  | 159591683  | 2     | Supragingival_plaque    |
| 700033601  | 159611913  | 1     | Supragingival_plaque    |
| 700032328  | 159915365  | 1     | Supragingival_plaque    |
| 700098450  | 809635352  | 1     | Posterior_fornix        |
| 700016482  | 159207311  | 1     | Tongue_dorsum           |
| 700102585  | 159227541  | 2     | Stool                   |
| 700106465  | 338793263  | 1     | Stool                   |
| 700101840  | 159591683  | 2     | Stool                   |
| 700015965  | 159591683  | 1     | L_Retroauricular_crease |

NOTE: These ones are harder!  
Feel free to go right to the solutions (end of the slide deck) and build the solution one command at a time.

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

## Part 3.3: Bug bytes

- Which six body sites were sampled the most?
  - Solution uses `cut`, `sort` (twice), `uniq -c`, and `tail`
- How many subjects contributed more than 10 samples?
  - Solution uses `cut`, `sort` (twice), `uniq -c`, and `tail`



metadata.tsv

| #SAMPLE_ID | SUBJECT_ID | VISIT | BODY_SITE               |
|------------|------------|-------|-------------------------|
| 700101852  | 159591683  | 2     | Supragingival_plaque    |
| 700033601  | 159611913  | 1     | Supragingival_plaque    |
| 700032328  | 159915365  | 1     | Supragingival_plaque    |
| 700098450  | 809635352  | 1     | Posterior_fornix        |
| 700016482  | 159207311  | 1     | Tongue_dorsum           |
| 700102585  | 159227541  | 2     | Stool                   |
| 700106465  | 338793263  | 1     | Stool                   |
| 700101840  | 159591683  | 2     | Stool                   |
| 700015965  | 159591683  | 1     | L_Retroauricular_crease |

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

## Part 3.4: Bug bytes

- How many (subject, body site) pairs were sampled 3 times?
  - Solution uses `cut`, `sort` (twice), `uniq -c`, and `tail`
- How many biological samples had sequencing replicates?
  - I.e. (subject, visit, body site) triples that appear more than once



`metadata.tsv`

| #SAMPLE_ID | SUBJECT_ID | VISIT | BODY_SITE               |
|------------|------------|-------|-------------------------|
| 700101852  | 159591683  | 2     | Supragingival_plaque    |
| 700033601  | 159611913  | 1     | Supragingival_plaque    |
| 700032328  | 159915365  | 1     | Supragingival_plaque    |
| 700098450  | 809635352  | 1     | Posterior_fornix        |
| 700016482  | 159207311  | 1     | Tongue_dorsum           |
| 700102585  | 159227541  | 2     | Stool                   |
| 700106465  | 338793263  | 1     | Stool                   |
| 700101840  | 159591683  | 2     | Stool                   |
| 700015965  | 159591683  | 1     | L_Retroauricular_crease |

```
demo/
├── README.txt
├── hmp2012
│   └── metadata.tsv
├── gov
│   └── us_constitution.txt
└── words
    ├── words.txt
    └── more_words
        └── wikipedia_top100.txt
```

# SPOILER ALERT!

(answers to HMP questions follow)

# Part 3.1: Bug bytes (solutions)

- Samples:
  - `wc -l metadata.tsv`
  - **ANS = 727 (discounting headers)**
- Stool samples:
  - `grep "Stool" metadata.tsv | wc -l`
  - **ANS = 141**

# Part 3.2: Bug bytes (solutions)

- Unique body sites:
  - `grep -v “#” metadata.tsv | cut -f4 | sort | uniq | wc -l`
  - **ANS = 16**
  - *Note: initial grep used to discard the header row*
- Unique subjects:
  - `grep -v “#” metadata.tsv | cut -f2 | sort | uniq | wc -l`
  - **ANS = 103**
- Try building the “body site” solution one command at a time:
  - `grep -v “#” metadata.tsv | less`
  - `grep -v “#” metadata.tsv | cut -f4 | less`
  - `grep -v “#” metadata.tsv | cut -f4 | sort | less`
  - `grep -v “#” metadata.tsv | cut -f4 | sort | uniq | less`
  - `grep -v “#” metadata.tsv | cut -f4 | sort | uniq | wc -l`

# Part 3.3: Bug bytes (solutions)

- Six most-sampled body sites:
  - `cut -f4 metadata.tsv | sort | uniq -c | sort | tail -6`
  - **ANS = Posterior\_fornix, Anterior\_nares, Buccal\_mucosa, Supragingival\_plaque, Tongue\_dorsum, Stool**
- Subjects with 10+ samples:
  - `cut -f2 metadata.tsv | sort | uniq -c | sort | tail -22`
  - **ANS = 21**
  - *Note: In this and following examples, we re-run the command with different sized “tails” until we find the find the transition we want. In this case, we are looking for the first time a 10 appears.*

## Part 3.4: Bug bytes (solutions)

- (Subject, body site) pairs sampled 3+ times:
  - `cut -f2,4 metadata.tsv | sort | uniq -c | sort | tail -16`
  - **ANS = 15 (including pairs sampled >3 times)**
- Sequencing replicates:
  - `cut -f2,3,4 metadata.tsv | sort | uniq -c | sort | tail -15`
  - **ANS = 14**

Extras

# Command line utilities: column

## What does it do?

“Normalize” the widths of column entries (*Excelify* your data)

## Usage

```
$ column -t my_file
```

| Options | What does it do? |
|---------|------------------|
|---------|------------------|

|                               |                                                                                               |
|-------------------------------|-----------------------------------------------------------------------------------------------|
| <code>-s '<i>char</i>'</code> | Specify the delimiter character (e.g. <code>-s '\t'</code> for tab); default = any whitespace |
|-------------------------------|-----------------------------------------------------------------------------------------------|

# Command line utilities: `diff`

## What does it do?

Compare two **TEXT** files and display differing lines

## Usage

```
$ diff my_file1 my_file2
```