File utilities and command environments

Curtis Huttenhower (chuttenh@hsph.harvard.edu)

Eric Franzosa (franzosa@hsph.harvard.edu)

http://huttenhower.sph.harvard.edu/bst281

Midterm: journal club

- Sign up for a journal club presentation by **end of day Wednesday!**
- Email instructional team with:
  - Group members (2-5, names + CCed).
  - Paper to present.
  - ~2-4 sentence justification.
    - Recent and/or high-impact quantitative biology.
  - Date of presentation (March 1, 6, 8).
    - 3-4 presentations / day.
    - First come, first served.
- 20 minute presentation + 5 minutes questions.
  - All group members must present.
  - Background, overview/results, methods, discussion/interpretation, questions.

The unix and linux (*nix) philosophy

- Work at the command line (terminal)
- Manipulate data as text
- Build/use programs that perform specific tasks very well
- Chain together small programs to perform complex tasks
- “Chaining” involves passing data as STDIN/STDOUT streams

The command line and you

- Being comfortable on the command line is helpful for data analysis/coding:
  - Command line tools save you from reinventing the wheel
- The data analysis programming trifecta:
  - Command line experience
  - A utility language for data manipulation (e.g. Python)
  - A statistical analysis and graphics language (e.g. R, though Python works too)
- The command line is a 90/10 learning process:
  - 90% of the data questions you'll encounter can be answered with 10% of the command line functionality (which you can master in a few weeks)
  - The rest you pick up bit-by-bit over your career

Using command-line tools

- If you're on a mac (or Linux), you have all of these built-in
- Not available by default in Windows command prompt, but a few options:
    - Windows **PowerShell**
    - **Cygwin** (mini-Linux that runs as a terminal program on Windows)
    - Windows 10 Bash shell (*experimental*)
    - **GOW** (Gnu on Windows) lightweight Cygwin alternative

*nix 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

*nix 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 com-
       mands, 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.  Warn-
              ing: some systems use ^V as a special literalization character.

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

*nix utilities: `ls`

What does it do?
List files in the current directory

Usage
`$ ls`
`$ 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)
	Lots of build in sorting options (size, type, last update, etc.)

*nix utilities: `rm`

What does it do?
Deletes (removes) a file **PERMANENTLY**

Usage
`$ rm my_file`

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

*nix utilities: cp

What does it do?
Copy a file

Usage
`$ cp my_file my_copy`

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

*nix utilities: mv

What does it do?

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

Usage

```
$ mv my_file MYFILE    # rename my_file as MY_FILE
$ my /my_dir/my_file . # move file "here" (.)
```

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

Path syntax

$ is used to indicate the start of a command (your prompt may look different)

```
$ command
```

/ is the root of the file system (C:\ on Windows, typically; note the backslash)

```
$ ls /
```

A subdirectory of the root directory

```
$ ls /my_dir/
```

. refers to the current directory

```
$ wc -l ./my_file.txt
$ cp /my_dir/my_file.txt . # copy file “here”
```

.. refers to the parent directory

```
$ mv ../my_file.txt .
```

*nix utilities: `cat`

What does it do?

Dumps a file on disk to STDOUT (for feeding into another program)

Usage

```
$ cat my_file           # lines of file scroll over screen
$ cat my_file | program # lines of file enter program as STDIN
```

*nix utilities: `less`

What does it do?
View a file or data stream with navigation options

Usage

```
$ less my_file
```

```
$ cat my_file | less # less often used at the end of a chain
```

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

*nix 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 my_file | 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)

*nix 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‘C’`	Break columns on the specified character instead of tab, e.g. `–t‘,’` for .csv file

*nix utilities: `sort`

What does it do?

Sort the lines of a data stream (alphabetically, by default)

Usage

```
$ sort my_file
```

Options	What does it do?
`-r`	Reverse the sort
`-k` N	Sort on the value of **WHITESPACE**-delimited column N
`-t`‘C’	Specify the delimiter character (e.g. `–t‘\t’` for tab)
`-n`	*Perform a numeric sort* (otherwise `10` comes before 2)

*nix 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?
`-c`	Count the unique lines instead of printing them

*nix 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

*nix utilities: `head/tail`

What does it do?
Stream the first/last (tail/head) lines of a data stream (default 10)

Usage

```
$ head my_file
```

```
$ head -n 100 my_file | tail # Stream lines 91-100
```

Options	What does it do?
`-n` N	Stream N lines instead of default 10

*nix utilities: `column`

What does it do?
"Normalize" the widths of column entries (Excel*ify* your data)

Usage
`$ column -t my_file`

Options	What does it do?
`-s'`*C*`'`	Specify the delimiter character (e.g. `-s'\t'` for tab); default = any whitespace

*nix 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” (dangerous)
	“*find*” can be a regular expression, and “*replace*” can use captured elements of the pattern (this will make more sense after our regular expressions lecture).

*nix utilities: `diff`

What does it do?

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

Usage

```
$ diff my_file1 my_file2
```

Chaining programs

- Most programs read from STDIN and write to STDOUT
- Start a chain by supplying a file name as an argument:
  - `grep “Hello” my_file.txt`
- You will sometimes see this syntax (“<” reads a file on disk to STDOUT):
  - `grep “Hello” < my_file.txt`
- 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`
- Dump the results to a file (**OVERWRITES!**)
  - `grep “Hello” my_file.txt | sort | uniq | wc –l` **`> hello_count.txt`**

Output/redirect to a file

Chaining programs

patient_data.tsv		
UNIQUE_ID	SUBJECT_NAME	SUBJECT_ZIP
134245	Smith, John	02135
145623	Doe, John	02134
...	...	...

What does this do?

```
$ cut –f3 patient_data.tsv | grep “02135” | wc -l
```

What does this do?

```
$ cut –f3 patient_data.tsv | sort | uniq | wc -l
```

What does this do?

```
$ sed “s/, Jon/, John/g” | grep “, John” | wc –l
```

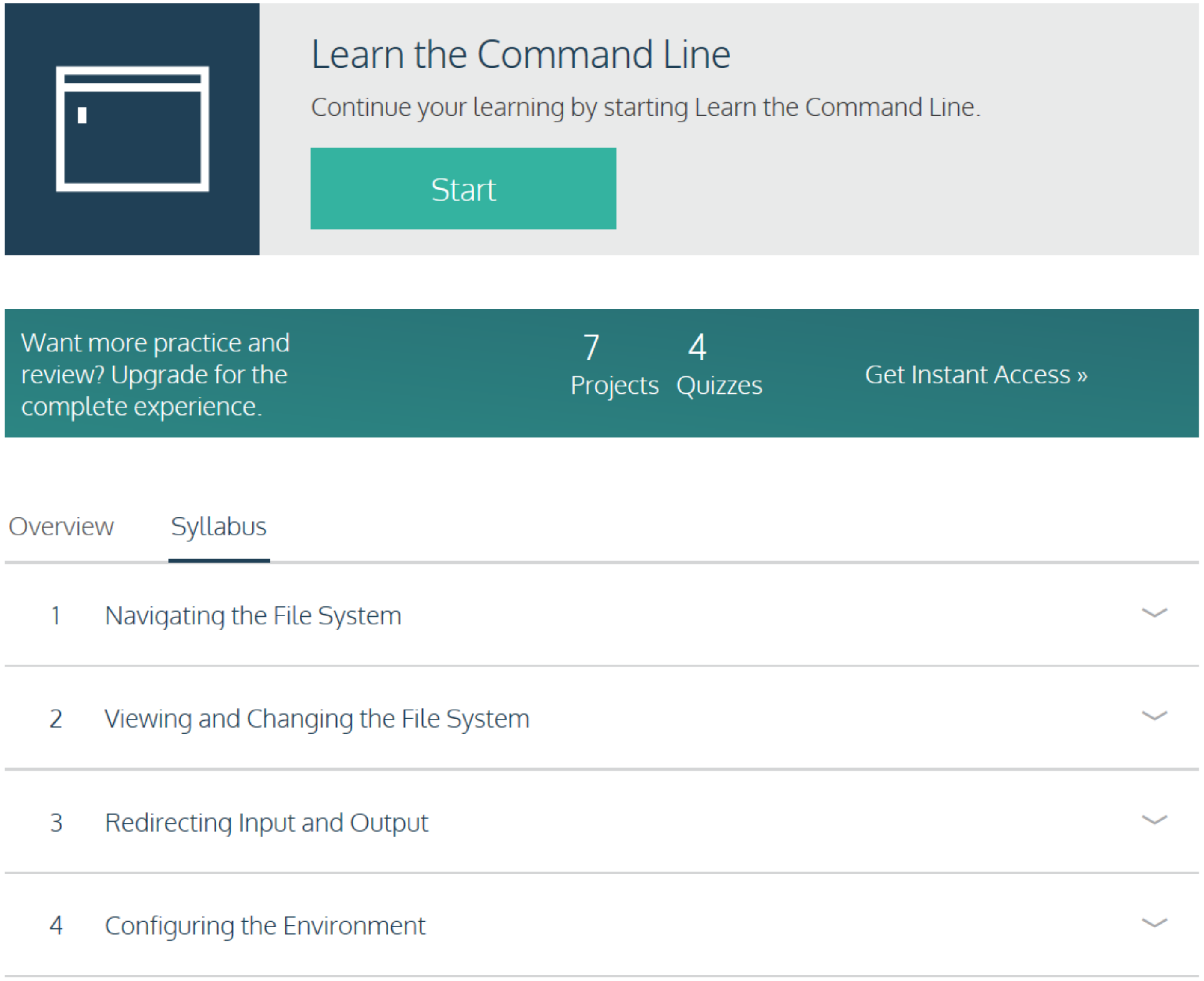
Chaining programs

- Consider `hmp2012_metadata.tsv`
- (Simplified) metadata for Human Microbiome Project samples
- Each sample comes from a body site of a particular subject at a given visit
- Questions:
  - How many unique subjects? Body sites?
  - cat hmp2012_metadata.tsv|cut -f4|sort|uniq|wc -l
  - What are the six most commonly sampled body sites?
  - cat hmp2012_metadata.tsv|cut -f4|sort|uniq -c|sort| tail -n 6
  - How many "retroauricular_crease" (ear) samples, ignoring left/right distinction?
  - cat hmp2012_metadata.tsv|cut -f4|grep "_crease"|wc -l
  - cat hmp2012_metadata.tsv|cut -f4|sed "s/[LR]_//g"|sort|uniq -c
  - How many subjects contributed more than 10 samples?
  - How many (subject, body site) pairs were sampled 3 times?
  - Are there any technical replicates?
    - Unique samples with same (subject, body site, visit) triple

Calling command-line tools in Python: `subprocess`

- `import subprocess`
- `subprocess.call( “ls –l | wc -l”, shell=True )`
  - Runs the given command
  - Output goes to the screen when command is FINISHED
- `subprocess.check_output( “ls –l | wc -l”, shell=True )`
  - Runs the given command
  - Output returned as a STRING when command is FINISHED
  - Can then be manipulated in Python
  - Multiline output delimited by “\n”
- `subprocess.Popen( )` for more advanced options
  - Provides a “handle” to the process that you can interact with (like a file handle)
  - E.g. live-stream output rather than waiting for process to finish

https://www.codecademy.com/learn/learn-the-command-line



Learn the Command Line

Continue your learning by starting Learn the Command Line.

Want more practice and review? Upgrade for the complete experience.

7 Projects 4 Quizzes

1 Navigating the File System

2 Viewing and Changing the File System

3 Redirecting Input and Output

4 Configuring the Environment