# Library Carpentry Week Two: Controlling Data (with the Shell)

1730-2030, Monday 16 November, City University London

## Lesson Plan

This lesson follows on from '[Week One: Basics](https://github.com/LibraryCarpentry/week-one-library-carpentry)', though there are no pre-requisites in terms of skills or knowledge required to complete this lesson.

### Installation (to be completed before the session)

Windows users, see the section entitled 'Installing Git Bash' in the Programming Historian lesson [*Introduction to the Bash Command Line*](http://programminghistorian.org/lessons/intro-to-bash). OS X and Linux users, simply make sure you know how to find your 'Terminal'.

### Data collection

Conducted as attendees enter the room

Thinking about this session on counting and mining using the command line, please rate your skill level. Would you say that you know:

1. Nothing
2. A Little
3. Lots
4. Lots and Lots

## Hour One

### Introduction

**SLIDE** Lib Carp schedule

**SLIDE** Reminder of where to get help

**SLIDE** Schedule for the session

In this session we will introduce programming by looking at how data can be manipulated, counted, and mined using the Unix shell, a command line interface to your computer and the files it has access to.

A Unix shell is a command-line interpreter that provides a user interface for the Linux operating system and for Unix-like systems (such as iOS). For Windows users, popular shells such as Cygwin or Git Bash provide a Unix-like interface (a command line interface preferable - to me at least - to Windows own flavour of command line). This session will cover a small number of basic commands using Git Bash for Windows users, Terminal for iOS. These commands constitute building blocks upon which more complex commands can be constructed to fit your data or project.

**SLIDE** The motivations for wanting to learn shell commands are many and various. What you can quickly learn is how to query lots of data for the information you want super fast. Say, for example, you have a query from a reader about the number of articles published in 2009 in academic history journals whose title contains the word 'International'. Now you could search a database. Alternatively you could work directly with the relevant data. The British Library has open data on journal articles, and I've prepared from that...

wc -l 2014-01\_JA.tsv

... a 500,000 line data file containing that information. Excel will struggle to manipulate that, but the shell won't. Let's look at this shell script:

grep 2009 2014-01\_JA.tsv | grep INTERNATIONAL | awk -F'\t' '{print $5}' | sort | uniq -c

This is simple, powerful, and does what we want. It may seem intimating but, as you'll discover this evening, it is deeply logical and eminently within your reach. Let us go through each part in turn:

* grep 2009 2014-01\_JA.tsv : this tells the machine to look in the spreadsheet 2014-01\_JA.tsv for all the lines that contain the string 2009 and to store those in memory. The pipe then tells the machine to hold those in memory for the minute as we have something else we want to do...
* grep INTERNATIONAL : ...that is look for the capitalised string international on those lines that have 2009 in them. The shell is case sensitive by default and I know that in my data most (if not all) occurrences of international in caps will be in a column that lists journal titles. Note: this isn't super exact, but I know my data and know it'll do the job of now (think back to the automate vs manual discussion last week). Again it holds this subset in memory and...
* awk -F'\t' '{print $5}' : ...moves on to the next bit. This is the most fiddly of the bits and not something we will cover properly today. But all it says is that 2014-01\_JA.tsv is a tab separated spreadsheet and to print out to the shell the 5th column (which is the one I know contains journal titles) of all the lines we've queried down to (those with 2009 in them, and then those with INTERNATIONAL in them) and to hold that in memory so that we can then...
* sort : ...sort that column. sort does what is says on the tin, and sorts the data we have left (which should just a single column containing journal titles) holding that sorted list in memory...
* uniq -c : ... so that we can then, finally, tell the machine to remove duplicates but as it is doing so count those duplicates and hold that data in memory.

As this is the last bit, the shell then - by default - prints the results to a shell: the number of articles published in 2009 in academic journals whose title contains the word 'International', with counts separated by journal. This last bit I have added in as, as we can see, we have a few false positives. We'd have to go back to our data to find out why, but this is a very good start: from 500,000 lines of journal article metadata to a few numbers and names in a small line of code.

You won't be doing exactly this by the end, but you won't be far off. We'll be covering quite a lot but the key commands are in the handout for reference. We'll proceed in a follow my leader fashion. The format will be that I'll demo a command, you copy, and then we'll discuss the results. Stickies for when you get stuck or something doesn't appear to work.

### Basics - navigating the shell

**SLIDE** We will begin with the basics of navigating the Unix shell.

Start by opening your shell. When you run it, you will likely see a black window with a cursor flashing next to a dollar sign. This is our command line.

If, when opening a command window or at any other time today, you are unsure of where you are in a computer's file system, you can find out what directory you are in using pwd command, which stands for "print working directory", and hitting enter - which executes commands in the shell. Try typing pwd and hitting enter.

To orient ourselves, let's get a listing of what files are in this directory. Type ls and you will see a list of every file and directory within your current location.

You may want more information than just a list of files. You can do this by specifying various *flags* to go with our basic commands. These are additions to a command that provide the computer with a bit more guidance of what sort of output or manipulation you want.

If you type ls -l and hit enter the computer returns a long list of files that contains information similar to what you'd find in your finder or explorer: the size of the files in bytes, the date it was created or last modified, and the file name.

In everyday usage you will be more used to units of measurement like kilobytes, megabytes, and gigabytes. Luckily, there's another flag -h that when used with the -l option, use unit suffixes: Byte, Kilobyte, Megabyte, Gigabyte, Terabyte and Petabyte in order to reduce the number of digits to three or less using base 2 for sizes.

Now ls -h won't work on its own. When you want to use two flags, you can just run them together. So, by typing ls -lh and hitting enter you receive an output in a human-readable format (note: that the order here doesn't matter).

You've now spent a great deal of time in your home directory. Let's go somewhere else. You can do that through the cd or Change Directory command.

If you type cd desktop you are now on your desktop (note: for Windows users, the case of the file/directory doesn't matter. For Linux users like me, it does, and I believe that is the same for Mac people). To double check, type pwd and you should see something that represents your desktop.

You'll note that this only takes you 'down' through your directory structure (as in into more nested directories). If you want to go back, you can type cd ... This moves us 'up' one directory, putting us back where we started. If you ever get completely lost, the command cd -- will bring you right back to the home directory, right where you started.

Try exploring: move around the computer, get used to moving in and out of directories, see how different file types appear in the Unix shell. **TWO MINUTES**

Being able to navigate your file system using the bash shell is very important for using the Unix shell effectively. And as you become more comfortable, you'll soon find yourself skipping directly to the directory that you want.

#### Summary

Within the Unix shell you can now:

* use the command pwd to find out where you are in on your computer
* use the command ls to list directory contents
* use flags -l and -lh to guide the output of the ls command
* use the command cd to move around your computer

### Basics

**SLIDE** As well as navigating directories, you can interact with files on the command line: you can read them, open them, run them, and even edit them, often without ever having to leave the interface. Sometimes it is easier to do this using a Graphical User Interface, such as Word or your normal explorer, but the more you work here the more it is useful and the more you write scripts the more you'll need this basic knowledge

Here's a few basic ways to interact with files.

First, you can create a new directory. For convenience's sake, we will create it in directory you extracted the data provided in advance. Here type mkdir firstdir and hit enter. This used the mkdir command (meaning 'Making Directory') to create a directory named 'firstdir'. Now, move into that directory using the cd command.

But wait! There's a trick to make things a bit quicker. Go up one directory (cd ..). To navigate to the firstdir directory you could type cd firstdir. Alternatively, you could type cd f and then hit tab. You will notice that the interface completes the line to cd firstdir. **Hitting tab at any time within the shell will prompt it to attempt to auto-complete the line based on the files or sub-directories in the current directory. Where two or more files have the same characters, the auto-complete will only fill up to the first point of difference, after which you can add more characters, and try using tab again. We would encourage using this method throughout today to see how it behaves (as it saves loads of time and effort!).**

The next step is to manipulate files.

Navigate to the text directory in the pre-circulated data directory. In here there is a copy of Jonathan Swift's *Gulliver's Travels* downloaded from Project Gutenberg. type ls -lh and hit enter to see details of this file.

You can read the text right here. To try this, type cat 829-0.txt. The terminal window erupts and *Gulliver's Travels* cascades by: this is what is known as printing to the shell. And it is great, in theory, but you can't really make any sense of that amount of text. Instead, you may want to just look at the first or the last bit of the file. **TIP: to cancel this print of 829-0.txt, or indeed any ongoing in the Unix shell, hit ctrl+c**

Type head 829-0.txt and hit enter. This provides a view of the first ten lines, whereas tail 829-0.txt provides a perspective on the last ten lines. This is a good way to quickly determine the contents of the file.

You may also want to change the file name to something more descriptive. You can 'move' it to a new name by using the mv or move command. To do this type mv 829-0.txt gulliver.txt and hit enter.

Afterwards, when you perform a ls command, you will see that it is now gulliver.txt. Had you wanted to duplicate it, you could have used the cp or copy command by typing cp 829-0.txt gulliver.txt

Now that you have seen and used several new commands, it's time for another trick. Hit the up arrow twice on your keyboard. Notice that mv 829-0.txt gulliver.txt appears before your cursor. You can continue pressing the up arrow to cycle through your previous commands. The down arrow cycles back toward your most recent command. This is another important labour saving function and something we'll use a lot this evening.

After having read and renamed several files, you may wish to bring their text together into one file. Before we do that let's use cp to duplicate the Gulliver file and give it the filename gulliver-backup.txt: any ideas how you do that? (**ANSWER**: cp gulliver.txt gulliver-backup.txt). Good, now that you have two copies of *Gulliver's Travels*, let's put them together to make an **even longer** book.

To combine, or concatenate, two or more files use the cat command again. Type cat gulliver.txt gulliver-backup.txt and press enter. This prints, or displays, the combined files within the shell. However, it is too long to read on this window! Luckily, by using the > redirector, you can send the output to a new file, rather than the terminal window. Hit up to get to your last command and amend the line to cat gulliver.txt gulliver-backup.txt > gulliver-twice.txt and hit enter. Now, when you type ls you'll see gulliver-twice.txt appear in your directory.

When combining more than two files, using a wildcard can help avoid having to write out each filename individually. Again, labour saving! A useful wildcard is \* which is a place holder for zero or more characters or numbers (note: this is slightly different from regex...). So, if you type cat \*.txt > everything-together.txt and hit enter, a combination of all the .txt files in the current directory are combined in alphabetical order as everything-together.txt. This can be very useful if you need to combine a large number of smaller file within a directory so that you can work with them in a text analysis program. Another wildcard worth remembering is ? which is a place holder for a single character or number. We shall return to shell wildcards later - for now, note again that they are similar to but not the same as the Regex we saw last week.

Finally, onto deleting. We won't use the now, but if you do want to delete a file, for whatever reason, the command is rm, or remove. **Be careful with the rm command**, as you don't want to delete files that you do not mean to. Unlike deleting from within your Graphical User Interface, there is **no** recycling bin or undo options. For that reason, if you are in doubt, you may want to exercise caution or maintain a regular backup of your data.

The syntax for rm is the same as cp and mv: for example rm gulliver.txt, adding wildcards as appropriate to specify the files to delete.

#### Summary

Within the Unix shell you can now:

* use the command mv to rename and move files.
* use the command cp to create a file from an existing file.
* use the command cat to combine more than one file of the same file type.
* use the wildcards \* and ? as place holders that delimit which files are to be manipulated by a given an action.
* use the rm command to delete unwanted files.

## Hour Two

### Stuff you can use in your practice - Manipulating, counting and mining research data

**SLIDE** Now you can work with the unix shell you can move onto learning how to count and mine data. These are rather simple and are unlikely to totally revolutionise your work. They are, however, alongside the consistent file structure and naming I touched on last week, the foundation of a more powerful set of commands that can count and mine your data.

#### Counting

You will begin by counting the contents of files using the Unix shell. The Unix shell can be used to quickly generate counts from across files, something that is tricky to achieve using the graphical user interfaces of standard office suites.

In the Unix shell, use the cd command to navigate to the directory that contains our data: the tabular subdirectory of the ...libcarp-wk2-data directory. Remember, if at any time you are not sure where you are in your directory structure, type pwd and hit enter.

Type ls -lh and then hit enter. This prints, or displays, a list that includes a file and a subdirectory.

The file in this directory is a zipped up version of the dataset 2014-01\_JA.tsv that contains journal article metadata.

The subdirectory is named derived\_data. It contains a single text file (to which we shall return) and four .tsv files derived from 2014-01\_JA.tsv. Each of these four includes all data where a keyword such as africa or america appears in the 'Title' field of 2014-01\_JA.tsv. The derived\_data directory also includes a subdirectory called results.

*Note: TSV files are those in which within each row the units of data (or cells) are separated by tabs. They are similar to CSV (comma seperated value) files were the values are separated by commas. The latter are more common but can cause problems with the kind of data we have, where commas can be found within the cells (though with the right encoding this can be overcome). Either way both can be read in simple text editors or in spreadsheet programs such as Libre Office Calc or Microsoft Excel.*

Before you begin working with these files, you should move into the directory in which they are stored. Navigate to ...libcarp-wk2-data/tabular/derived\_data directory.

Now that you are here you can count the contents of the files.

The Unix command for counting is wc. Type wc -w 2014-01-31\_JA-africa.tsv and hit enter. The flag -w combined with wc instructs the computer to print a word count, and the name of the file that has been counted, into the shell.

As was seen earlier today flags such as -w are an essential part of getting the most out of the Unix shell as they give you better control over commands.

If your reader request or piece of work is more concerned number of entries (or lines) than the number of words, you can use the line count flag. Type wc -l 2014-01-31\_JA-africa.tsv and hit enter. Combined with wc the flag -l prints a line count and the name of the file that has been counted.

Finally, type wc -c 2014-01-31\_JA-africa.tsv and hit enter. This uses the flag -c in combination with the command wc to print a character count for 2014-01-31\_JA-africa.tsv *Note: OS X users should replace the -c flag with -m.*

With these three flags, the most obvious thing we can use wc for is to quickly compare the shape of sources in digital format - for example word counts per page of a book, the distribution of characters per page across a collection of newspapers, the average line lengths used by poets. You can also use wc with a combination of wildcards and flags to build more complex queries.

Can you guess what the line wc -l 2014-01-31\_JA-a\*.tsv will do? Correct! This prints the line counts for 2014-01-31\_JA-africa.tsv and 2014-01-31\_JA-america.tsv, offering a simple means of comparing these two sets of research data. Of course, it may be faster if you only have a handful of files to compare the line count for the two documents in Libre Office Calc, Microsoft Excel, or a similar spreadsheet program. But when wishing to compare the line count for tens, hundreds, or thousands of documents, the Unix shell has a clear speed advantage.

Moreover, as our datasets increase in size you can use the Unix shell to do more than copy these line counts by hand, by the use of print screen, or by copy and paste methods. Using the > redirect operator we saw earlier you can export our query results to a new file. Type wc -l 2014-01-31\_JA-a\*.tsv > results/DATE\_JA-a-wc.txt (or something like that, the last bit after results/ could be anything!) and hit enter. This runs the same query as before, but rather than print the results within the Unix shell it saves the results as DATE\_JA\_a-wc.txt. By prefacing this with results/ the shelll is instructed to save the .txt file to the results sub-directory. To check this, navigate to the results subdirectory, hit enter, type ls, and hit enter again to see this file. Type head DATE\_JA-a-wc.txt to see the file contents in the shell (as it is 10 lines or fewer in length, all the file contents will be shown here).

#### Mining

The Unix shell can do much more than count the words, characters, and lines within a file. The grep command (meaning 'global regular expression print') is used to search across multiple files for specific strings of characters. It is able to do so much faster than the graphical search interface offered by most operating systems or office suites. And combined with the > operator, the grep command becomes a powerful research tool can be used to mine your data for characteristics or word clusters that appear across multiple files and then export that data to a new file. The only limitations here are your imagination, the shape of your data, and - when working with thousands or millions of files - the processing power at your disposal.

To begin using grep, first navigate to the derived\_data directory (cd ..). Here type grep 1999 \*.tsv and hit enter. This query looks across all files in the directory that fit the given criteria (the .tsv files) for instances of the string, or character cluster, '1999'. It then prints them within the shell.

Press the up arrow once in order to cycle back to your most recent action. Amend grep 1999 \*.tsv to grep -c 1999 \*.tsv and hit enter. The shell now prints the number of times the string 1999 appeared in each \*.tsv file. If you look at the output from the previous command, this tends to be refer to the date field for each journal article.

Strings need not be numbers. grep -c revolution \*.tsv, for example, counts the instances of the string revolution within the defined files and prints those counts to the shell. Run this, observe the output, and then amend it to grep -ci revolution \*.tsv. This repeats the query, but prints a case insensitive count (including instances of both revolution and Revolution). Note how the count has increased nearly 30 fold for those journal article titles that contain the keyword 'america'. As before, cycling back and adding > results/, followed by a filename (ideally in .txt format), will save the results to a data file.

So far we have counted strings in file and printed to the shell or to file those counts. But the real power of grep comes in that you can also use it to create subsets of tabulated data (or indeed any data) from one or multiple files. Type grep -i revolution \*.tsv and hit enter. This script looks in the defined files and prints any lines containing revolution (without regard to case) to the shell. grep -i revolution \*.tsv > results/DATE\_JAi-revolution.tsv saves it to file.

However if we look at this file, it contains every instance of the string 'revolution' including as a single word and as part of other words such as 'revolutionary'. This perhaps isn't as useful as we thought... Thankfully, the -w flag instructs grep to look for whole words only, giving us greater precision in our search. Type grep -iw revolution \*.tsv > results/DATE\_JAiw-revolution.tsv and hit enter. This script looks in both of the defined files and exports any lines containing the whole word revolution (without regard to case) to the specified .tsv file. wc -l \*revolution shows us the difference between them.

Finally, you can use the regular expression syntax covered [last week!](https://github.com/LibraryCarpentry/week-one-library-carpentry/blob/master/lesson-materials/2015-08-13_LibCarp-lesson-one.md) to search for similar words. In gallic.txt we have the string fr[ae]nc[eh]. The square brackets here ask the machine to match any character in the range specified. So when used with grep as grep -iw --file=gallic.txt \*.tsv the shell will print out each line containing the string:

* france
* french
* frence
* franch

#### Exercise

With the person next to you, select a word to search for and use what you have learnt do to the following:

Search for all case sensitive instances of that word in all four derived tsv files in this directory. Print you results to the shell.

* grep hero \*.tsv

Search for all case sensitive instances of that word in the 'America' and 'Africa' tsv files in this directory. Print you results to the shell.

* grep hero 2014-01-31\*

Count all case sensitive instances of that word in the 'America' and 'Africa' tsv files in this directory. Print you results to the shell.

* grep -c hero 2014-01-31\*

Count all case insensitive instances of that word in the 'America' and 'Africa' tsv files in this directory. Print you results to the shell.

* grep -ci hero 2014-01-31\*

Search for all case insensitive instances of that word in the 'America' and 'Africa' tsv files in this directory. Print you results to a new .tsv file.

* grep -i hero 2014-01-31\* > new.tsv

Search for all case insensitive instances of that whole word in the 'America' and 'Africa' tsv files in this directory. Print you results to a new .tsv file.

* grep -iw hero 2014-01-31\* > new2.tsv

Compare the line counts of the last two files.

* wc -l FILENAMES

Open both files in a text editor (Notepad++, Atom, Kate, whatever you prefer) or Excel-like program to see the difference between searching strings and searching whole words using grep

#### Recap

Within the Unix shell you can now:

* use the wc command with the flags -w and -l to count the words and lines in a file or a series of files.
* use the redirector and structure > subdirectory/filename to save results into a subdirectory.
* use the grep command to search for instances of a string.
* use with grep the -c flag to count instances of a string, the -i flag to return a case insensitive search for a string, the -v flag to exclude a string from the results, and -w to return a whole word only search
* use - --file=list.txt to use the file list.txt as the source of strings used in a query
* combine these commands and flags to build complex queries in a way that suggests the potential for using the Unix shell to count and mine your research data and research projects.

**DATE CAPTURE**

At this point, I'd like you to describe on your sticky note, briefly, a use case for this that might change your practice.

## Hour Three

### Stuff that reflects what library users are doing with library data - Working with free text

**SLIDE** So far we have looked at how to use the Unix shell to manipulate, count, and mine tabulated data. Most libary data, especially digitised documents used by some of research communities libraries support, is much messier journal article metadata. Nonetheless many of the same techniques can be applied to non-tabulated data, such as free text, we just need to think carefully about what it is we are counting and how we can get the best out of the Unix shell.

Thankfully there are plenty of folks out there doing this sort of work and we can borrow what they do as an introduction to working with these more complex files. So for this final exercise we're going to leap forward a little in terms of difficulty to an scenario where we won't learn about everything that is happening in detail or discuss at length each command. We're going to prepare and pull apart a text to show the potential of using the Unix shell in research. And where commands we've learnt about are used, I've left some of the figuring out to do to you - so please refer to your notes if you get stuck!

**NB: leave 10 minutes at the end to go through final bits**

#### Grabbing a text, cleaning it up

*Work on this exercise with the person next to you*

Head to .../libcarp-wk2-data/text/. We're going to work again with the gulliver.txt file we saw earlier.

**SHOW THE FILE WITH less -N gulliver.txt**

We're going to start by using the sed command. The command allows you to edit files directly.

Type sed '9352,9714d' gulliver.txt > gulliver-nofoot.txt and hit enter.

The command sed in combination with the d value will look at gulliver.txt and delete all values between the rows specified. The > action then prompts the script to this edited text to the new file specified.

Now type sed '1,37d' gulliver-nofoot.txt > gulliver-noheadfoot.txt and hit enter. This does the same as before, but for the header.

You now have a cleaner text. The next step is to prepare it even further for rigorous analysis.

We now use the tr command, used for translating or deleting characters. Type tr -d [:punct:] < gulliver-noheadfoot.txt > gulliver-noheadfootpunct.txt and hit enter.

This uses the translate command and a special syntax to remove all punctuation. It also requires the use of both the output redirect > we have seen and the input redirect < we haven't seen.

Finally regularise the text by removing all the uppercase lettering. Type tr [:upper:] [:lower:] < gulliver-noheadfootpunct.txt > gulliver-clean.txt and hit enter.

Open the gulliver-clean.txt in a text editor. Note how the text has been transformed ready for analysis.

#### Pulling a text apart, counting word frequencies

We are now ready to pull the text apart.

Type tr ' ' '\n' < gulliver-clean.txt > gulliver-linebyline.txt and hit enter.

This uses the translate command again, this time to translate every blank space into \n which renders as a new line. Every word in the file will now have its own line.

This isn't much use, so to get a better sense of the data we need to use another new command called sort. Type sort gulliver-linebyline.txt > gulliver-ordered.txt and hit enter.

This script uses the sort command to rearrange the text from its original order into an alphabetical configuration. Open the file in a text editor and after scrolling past some blank space you will begin to see some numbers and finally words, or at least lots of copies of 'a'!

This is looking more useful, but we can go one step further. Type uniq -c gulliver-ordered.txt > gulliver-final.txt and hit enter.

This script uses uniq, another new command, in combination with the -c flag to both remove duplicate lines and produce a word count of those duplicates.

**Note: there is a windows/linux issue here worth flagging about special characters**

Note that these steps can be simplified by building 'pipes'. So...

tr ' ' '\n' < gulliver-clean.txt | sort | uniq -c > gulliver-final.txt

...would have done the line-by-line, sorting, and removal of duplicates in one go.

Either way we have now taken the text apart and produced a count for each word in it. This is data we can prod and poke and visualise, that can form the basis of our investigations, and can compare with other texts processed in the same way. And if we need to run a different set of transformation for a different analysis, we can return to gulliver-clean.txt to start that work

**note there are a few bits of punctuation in here - I've left these in deliberately as you should always bug fix! The internet is a always a good place to start searching for why this might have happened (something about the punct command we used...)**

And all this using a few commands on an otherwise unassuming but very powerful command line.

Before we move on, we'll go back to the opening command:

grep 2009 2014-01\_JA.tsv | grep INTERNATIONAL | awk -F'\t' '{print $5}' | sort | uniq -c

Can you describe - without looking at your notes... - exactly what is going on here? (I'll forgive you not know the awk bit given that we'll not covered that...)

#### Where to go next

**SLIDE** Deborah S. Ray and Eric J. Ray, *Unix and Linux: visual quickstart guide*, 4th edition (2009). Invaluable (and not expensive) as a reference guide - especially if you only use the command line sporadically!

[The Command Line Crash Course](http://cli.learncodethehardway.org/book/) 'learn code the hard way' -- good for reminders of the basics.

[Automate the Boring Stuff](https://automatetheboringstuff.com/)

**SLIDE** [Coursera Computer Science 101](https://www.coursera.org/course/cs101) If you feel you need some context to what we've done today, this is ideal covering how computers work, jargon, and key concepts in programming (such as loops and logic). Free, doesn't have to be taken as a class but in your own time.

Another Coursera course, [Programming for Everybody (Python)](https://www.coursera.org/course/pythonlearn) is available and lasts 10 weeks. So if you have 2-4 hours to spare. Python is popular in research programming as it is readable, relatively simple, and very powerful.

Bill Turkel and the Digital History community more broadly. The second lesson you did today was based on a lesson Bill has on [his website](http://williamjturkel.net/2013/06/15/basic-text-analysis-with-command-line-tools-in-linux/) and Bill is also a general editor of the [Programming Historian](http://programminghistorian.org/project-team). The Programming Historian is an open, collaborative book aimed at providing programming lessons to historians. Although the lessons are hooked around problems historians have, their lessons - which cover various programming languages - have a wide applicability - indeed today's course is based on two lessons I wrote with Ian Milligan, an historian at Waterloo, Canada - for ProgHist. Bill also has a wonderful tutorial on ['Named Entity Recognition with Command Line Tools in Linux'](http://williamjturkel.net/2013/06/30/named-entity-recognition-with-command-line-tools-in-linux/) which I thoroughly recommend if you want to automatically find, markup, and count names, places, and organisations in text files...

#### NER Demo

**SLIDE** Although Named Entity Recognition relies on a number of processes we need to critique, it can be run across texts quickly and simply from the command line. We start by setting the named entity recognition running on a txt (here on a text with punctuation removed)

stanford-ner/ner.sh gulliver-noheadfootpunct.txt > gulliver\_ner.txt

Looking at the text now, we can see that the NER has tagged some words with what it thinks are people, places, et al. We then clean up loose tags

sed 's/\/O / /g' < gulliver\_ner.txt > gulliver\_ner-clean.txt

From which we can count persons...

egrep -o -f personpattr gulliver\_ner-clean.txt | sed 's/\/PERSON//g' | sort | uniq -c | sort -nr > gulliver\_ner-pers-freq.txt *note: egrep is merely a variant of grep that looks for patterns*

And count places....

egrep -o -f locpattr gulliver\_ner-clean.txt | sed 's/\/LOCATION//g' | sort | uniq -c | sort -nr > gulliver\_ner-loc-freq.txt

Now the results of this are up for debate. Many persons seem to me to be missing, suggesting the applicability of the software for this purpose may be questionable. But I hope you can see that the process is simple and can be reapplied to other textual data when you want to quickly get a sense of the people or places it contains. And that it is one of many tools (another useful one being wget, a command that enables you to archive webpages) that work well on the command line.

#### Conclusion

**SLIDE** In this session you have learned to navigate the Unix shell, to undertake some basic file counting, concatenation and deletion, to query across data for common strings, to save results and derived data, and to prepare textual data for rigorous computational analysis.

This only scratches the surface of what the Unix environment is capable of. It is hoped, however, that this session has provided a taster sufficient to prompt further investigation and productive play.

Keep in mind that the full potential of the tools can offer may only emerge upon embedding these skills into real projects. Nonetheless, being able to manipulate, count and mine thousands on files is extremely useful. Even a large collection of files which do not contain any alpha-numeric data elements, such as image files, can be easily sorted, selected and queried depending on the amount of description, of metadata contained in each filename. If not a prerequisite of working with the Unix, then taking the time to structure your data in a consistent and predictable manner is certainly a significant step towards getting the most out of Unix commands. And if you can find a way of using the Unix shell regularly - perhaps only to copy or amend files - you'll keep the basics fresh, meaning that next time you have cause to use the Unix shell for more complex commands, you shouldn't need to learn it all over again.

### References

James Baker and Ian Milligan, 'Counting and mining research data with Unix', *The Programming Historian* ([2014](http://programminghistorian.org/lessons/research-data-with-unix)

Ian Milligan and James Baker, 'Introduction to the Bash Command Line', *The Programming Historian* ([2014](http://programminghistorian.org/lessons/intro-to-bash))

William J. Turkel, 'Named Entity Recognition with Command Line Tools in Linux' ([30 June 2013](http://williamjturkel.net/2013/06/30/named-entity-recognition-with-command-line-tools-in-linux/)). The section 'NER Demo' is adapted from this and shared under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported](http://creativecommons.org/licenses/by-nc-sa/3.0/).

William J. Turkel, 'Basic Text Analysis with Command Line Tools in Linux' ([15 June 2013](http://williamjturkel.net/2013/06/15/basic-text-analysis-with-command-line-tools-in-linux/). The sections 'Grabbing a text, cleaning it up' and 'Pulling a text apart, counting word frequencies' are adapted from this and shared under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported](http://creativecommons.org/licenses/by-nc-sa/3.0/).

### Next Week

Git