**Roslyn Walker, Student number 45149631**

Learning Journal

FOAR705 Digital Humanities

Week 6. 6 September 2019

=======================================================

Contents

[**(1) Technology deployment** 1](#_Toc18335634)

[**(2) Data carpentry exercise notes** 6](#_Toc18335635)

======================================================

## **(1) Technology deployment**

6 September 2019. Objective. Ethical (Testing duty 1) of Elaboration Plan by accessing Twitter user agreement.

Result.

Next step.

**6 September 2019. Objective. Source data testing. Duty 2 of Elaboration plan. Identify relevant source files from Twitter**

Steps. Look at format of source files from Twitter in [atom.io](https://slack-redir.net/link?url=http%3A%2F%2Fatom.io)  Unzip pack of files.

Result. There are 47 items.

Start with elaboration testing with two files: images and tweets.

Filenames: tweet-media-part 1 contains 34 images. tweet.js contains text file.

Results: first 39 lines of tweet.js

window.YTD.tweet.part0 = [ {

"retweeted" : false,

"source" : "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>",

"entities" : {

"hashtags" : [ ],

"symbols" : [ ],

"user\_mentions" : [ ],

"urls" : [ {

"url" : "https://t.co/YGf8nDBvbJ",

"expanded\_url" : "https://www.theguardian.com/environment/2019/jun/14/pope-francis-declares-climate-emergency-and-urges-action?CMP=Share\_iOSApp\_Other",

"display\_url" : "theguardian.com/environment/20…",

"indices" : [ "59", "82" ]

} ]

},

"display\_text\_range" : [ "0", "82" ],

"favorite\_count" : "0",

"id\_str" : "1139720038326259712",

"truncated" : false,

"retweet\_count" : "0",

"id" : "1139720038326259712",

"possibly\_sensitive" : false,

"created\_at" : "Sat Jun 15 02:23:26 +0000 2019",

"favorited" : false,

"full\_text" : "Pope Francis declares 'climate emergency' and urges action https://t.co/YGf8nDBvbJ",

"lang" : "en"

}, {

"retweeted" : false,

"source" : "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>",

"entities" : {

"hashtags" : [ ],

"symbols" : [ ],

"user\_mentions" : [ {

"name" : "Linda Mottram",

"screen\_name" : "LindaMottram",

"indices" : [ "3", "16" ],

"id\_str" : "82790149",

"id" : "82790149"

} ],

"urls" : [ ]

Next steps: Generate shell script to scrap data in readable form.

Filed source data in Git.

**6 September 2019. Objective. Open Twitter account for developing content and accessing source data.**

Steps: Registered for new account: name: Carrier Pigeon @Carrier71958443

Notification from Twitter to apply for a Twitter developer account

"You may continue managing your existing apps but if you would like to create new apps or use Twitter premium APIs, please apply for a developer account. As a developer platform, our first responsibility is to our users: to provide a place that supports the health of conversation on Twitter. To continue to prevent misuse of our platform, we have introduced a few new requirements for developers."

Result: Profile to register for developer account active.

Next steps: apply for a developer account on Twitter to access API. Register at <https://developer.twitter.com/en/apps>

**6 September 2019: Objective. Explore request requirements use of Public Twitter content.**

Steps: Ref <https://developer.twitter.com/en/docs/tutorials/one-time-historical-powertrack-jobs.html>Discovery that Public Twitter content dating back to the first Tweet in 2006 is available through request to Twitter, but requires purchase.

Result: As there is no budget for this project, in addition project is not endorsed for research by ethics committee. These are barriers to the exploration of public Twitter content.

Next steps. Continue explore workflow application through investigation of my own Twitter content and data files. [Cross ref: Learning Journal Date: 15 August 2019. Objective: Download data from my Twitter account.]

**5 September. Objective. Understand Git and writing a git commit message. Testing. 6. Data governance (Testing duty 5 of Elaboration plan )**

Action: Read Ref: <https://chris.beams.io/posts/git-commit/>Note: Write in the Git repository log with concise and consistent entries. Communicate the context around the change. Provides revision history. Seven rules from Chris Beams:

* Separate subject from body with a blank line
* Limit the subject line to 50 characters
* Capitalize the subject line
* Do not end the subject line with a period
* Use the imperative mood in the subject line
* Wrap the body at 72 characters
* Use the body to explain what and why vs. how

Result. Explain the problem the commit solves.

3 September 2019: Objective resolve unresponsive terminal Max Os

Error: non response in terminal

Action: **Control-L**

Result: cleared screen and displayed prompt. Screen redrew. Notes for corrective steps ref: <https://www.oreilly.com/library/view/learning-unix-for/0596004702/ch01s04.html>

**2 September 2019: Objective access a text editor.**

Action: Download <https://atom.io/> for use as nonspecific program data format.

Result: Downloaded text [editorAtom.io](http://editorAtom.io), installed in dock. Atom. Note. Provides version control with Git.

**23 August 2019: Objective: transfer ownership of my Github files from private ownership Roslyn007 to MQ-FOAR705.**

Action: In Github reassign owner to MQ-FOAR705, through settings.

In Overleaf, save to MQ-FOAR705 not Roslyn007.

Result: File ownership transferred to MQ-FOAR705 so the files are visible to teaching.

**23 August 2019: Objective: Investigate application Github.**

Action: For a demonstration of hyperref for the Learning journal HD cross-referencing [https://github.com/MQ-FOAR705/Demonstration-of-Hyperref](https://slack-redir.net/link?url=https%3A%2F%2Fgithub.com%2FMQ-FOAR705%2FDemonstration-of-Hyperref) <https://github.com/MQ-FOAR705/Demonstration-of-crossreftools> and this demo requires only one declaration

**23 August 2019: Objective: Understand application of Computational thinking.**

Steps: noted with reference in Slack:

Decomposition: Breaking down data, processes, or problems into smaller, manageable parts

Pattern Recognition: Observing patterns, trends, and regularities in data.

Algorithm Design: Developing the step by step instructions for solving this and similar problems

* First (and most important), decompose the activities that involve ‘pains’, opportunities for ‘gains’, and any solutions you proposed into small parts and/or discrete steps.
* Next (if possible), identify patterns in the problems you are trying to solve or the solutions you are proposing.
* Finally, revise the solutions you developed during BA to produce a step-by-step guide describing what you want to accomplish.
* To rephrase the above, scoping II now takes the amorphous pains/gains from Scoping I and will "output" a list of steps, a recipe, an algorithm, for individual tasks which must be done (even if you don't know how to do them) to go from start to finish.

Result: specifically interested in automated transcription for better ways to do reference from Brian to Alveo's transcriber tool: "Data storage with appropriate metadata > Loading it into a program which can extract audio from individual files > Sending that audio to The Cloud > Rendering the returns as appropriate text > Saving the text and metadata into a machine readable format that humans can also use".

Next steps: consider steps in application of automated transcription of social media under hashtag publics and counterpublics. Retain message and image frame.

Further reading on application: computational thinking - [https://robbotresources.com/new-blog/2018/11/3/what-is-computational-thinking](https://slack-redir.net/link?url=https%3A%2F%2Frobbotresources.com%2Fnew-blog%2F2018%2F11%2F3%2Fwhat-is-computational-thinking)

**23 August 2019: Objective. Task for Scoping II. Computational thinking.**

Action: seeking advice on path and application relating to repository and data organisation for image objects as archives.

Steps:

* Noted in my scoping paper tasks for developing a dataset for cataloguing and adding contextual information about works in an archive collection (i.e. thematic tagging, making connections between objects, people, places).
* Take an existing list in a word document and place this information into a spreadsheet format (Excel).
* The list for word has been generated from a list of terms used by other ephemeral art institutions in their archive repositories. I've noted these steps in my learning journal about starting work on the data set.
* My question in developing this data: What conventions should I use and work on first -- some ideas from today: coding sheet of terms, Readme files, control vocabulary --that will make my task of tagging digital records easier when posting them to the repository.
* How do I start to generate a useful list of terms for tagging an object in the repository?

Result: Advice from Shawn: To answer your question: you're in the realm of a number of existing platforms / systems. I think we discussed this last week: you are somewhere between [https://tropy.org/](https://slack-redir.net/link?url=https%3A%2F%2Ftropy.org%2F) and a content or document management system (CMS or DMS). You could even use bibliography software like Zotero to do this (it's worth investigating during elaboration at least).

Next steps. Trial applications across: bibliographic management: content and document management, and image repositories. Consider specific application to archive of social media content and images in preparation for analysis and curation. Test data thematic tagging and process data collection from source to filter content.

19 August 2019. **Objective.** Submit assignment in LaTex on Cloudstor.

**Action.** I worked through LaTex and converted my scoping study by pasting text into Overleaf and formatted it through RTF. I exported to PDF okay and submitted, and exported in format as .zip file to submit on Cloudstor.

**Error.** How do I preserve .tex when uploading to Cloudstor? Files in the submission folder appear to be .tex and pdf and zip. I repeated steps and got the same result. Why are some folders in the submission folder bright orange folders and others file extensions as noted. Some tips and steps for the best path would be great.

Next steps: Solution some folders have tex files in them submission as described above is okay.

**19 August. Objective: clarify scoping task for submission.**

Action: I feel I've gone off topic in my scoping exercise by using my PACE research internship role and task to produce a virtual exhibition of documentation through a digitised archive. I've added a note that I am using the proof of concept to build a repository for images online from social media that will form data for my thesis.

This means two tasks: (1) develop proof of concept building the virtual database and (2) application to social media.

I note that instructions asked for "Think about a major research project you have completed, and about your thesis. Imagine yourself going through the process of producing a 100+ page thesis and: "I wonder if this might put the exercise out of sync somewhere and if I should focus on page production instead?

Seeking advice from Brian on approach.

Result: Advice response from Brian: “I think having a specific and productive need for your proof of concept is far better than generic thesis support. One of the things we are grading for in the scoping is "hole in the literature" (for HD, I mean) -- so if you can demonstrate there don't exist well documented solutions for virtual exhibitions, especially as they apply to X, I feel that satisfies criteria well. The way to think about it is more "How would I get a journal article out of this?"

Error. Focus on Scoping paper II and defining the boundaries of what is in and out of the scope of the project: requires definition of boundaries eg what’s in and what’s out. Include the type of data to be used in the proof of concept. On technical aspects, commit in version control and document in Learning Journal.

Next steps. Address errors and consider/explore aspects of “hole in the literature” that would be improved by a technical solution.

Date: 15 August 2019. 20:52

**Example one: Download Twitter Content.**

**Objective: Download data from my Twitter account.**

As a test example of data retrieval for my discipline: Journalism. I am downloading data from my own Twitter archive. Following instructions posted at: download your Twitter archive: https://help.twitter.com/en/managing-your-account/how-to-download-your-twitter-archive

**Action from Twitter help menu:**

1. Go to your Account settings by clicking on the more icon in the navigation bar, and selecting Settings and privacy from the menu.
2. Under the Account section, click Your Twitter data.
3. Enter your password under Download your Twitter data, then click Confirm.
4. Click the Request data button. If your Twitter account is connected to Periscope, you’ll have the option to Request data from Periscope as well.
5. When your download is ready, we'll send an email to your connected email account. From your settings, you can click the Download data button under the Download data section.

**Error:** Needed to reset password to account, as password is saved and hidden. Completed.

1. Step one, there is no more icon, click on the profile picture, Settings and privacy is in the drop down menu.

**Result:** Twitter advises in a note on help that it may take a few days for them to prepare the download of my Twitter archive. I hadn’t received the data by the end of this training session. I will check again tomorrow. This exercise flags data preparation and may take several days to retrieve from Twitter.

I am interested in seeing how they return the data and formatting/presentation whether the data will need to be formatting for consistency and useability in a spreadsheet format.

Action: Sought advise from Brian, note to try to look at the file in a real text editor like [atom.io](https://slack-redir.net/link?url=http%3A%2F%2Fatom.io) or sublime text. There are ways of reading json, but step 1 is to take a look at it, and to look for the metadata.

Next steps: I will return to trial this approach. I would also like to download my Instagram data

**Example two: Preparing metadata for digitisation of archives project.**

**Objective:** Set up a data table to collate categories and identifying tags for an online database of digital archives.

I am building a data set of metadata tags for my Research Internship placement. My host organisation started building a list in a word document as tables. There were multiple tables and inconsistent data entry in the tables.

Action: Open new spreadsheet in Excel.

1. Import table from Word to Excel by copying table in Word and pasting it to Excel
2. I’ve opened a new spreadsheet and placed the data in the first tab.
3. In a new tab titles revised tag, started to clean data.
4. Removed formatting and changed heading using checklist for tidy data rules

Result: Placing the data in a spreadsheet provides easier access and viewing of complete set of tags. I can view the gaps in the data. I can clean up descriptors and edit for consistency.

I can build a tag list without fussing with formatting and table building in Word.

16 August 2019. **Objective:** Open data set received from Twitter.

Action: I received an email yesterday as a result of my inquiry to Twitter to access my data on Twitter.

1. The email was sent to an email account I don’t frequently used, but often link to applications to screen junk mail. The email was received within an hour of the request. The email provides a link to download, that is noted as expiring in a month.
2. The link goes back to my Twitter account and asks for my password. I entered my password and arrived at new page with a download link.
3. Download is a twitter zip file with 47 items, with two folders profile\_media and tweet\_media, files have a .js extension.

Error: Unable to utilise data without further knowledge on which files to access or what data I want to retrieve. Note the term your Twitter data includes range of a files including periscope.

**16 August 2019. Objective: explore version management on Github**

Action: Asked question to channel: When uploading a new version of a document to github, and the file name has changed will version control apply to the document along with earlier versions of the document under different title names?

Result: Advice from Brian. Github tracks changes line by line in your documents, and tries to identify your documents by content. When it's a zip file (i.e docx) then it's much harder for git to see inside. (For a fun time, rename the .docx to .zip and see what's inside).

However, for text based files, yes it will absolutely track their names as they change. There are also manual ways to tell git that it's the same file.

Result: uploaded files to Github and downloaded, next tested new versions on existing documents and annotating in commit.

## [**(2) Data carpentry exercise notes**](#_top)

**Objective: Complete lesson on Finding Things. Teaching:** 25 min **Exercises:** 20 min <https://swcarpentry.github.io/shell-novice/07-find/index.html>

Objective: Use grep to select lines from text files that match simple patterns.

Use find to find files and directories whose names match simple patterns.

Use the output of one command as the command-line argument(s) to another command.

Explain what is meant by ‘text’ and ‘binary’ files, and why many common tools don’t handle the latter well.

Objective: find specific text in file:

Action: $ cat haiku.txt

Result:

Roslyns-iMac:~ Tquestudio$ cd Desktop/data-shell/writing

Roslyns-iMac:writing Tquestudio$ cat haiku.txt

The Tao that is seen

Is not the true Tao, until

You bring fresh toner.

With searching comes loss

and the presence of absence:

"My Thesis" not found.

Yesterday it worked

Today it is not working

Software is like that.

Objective: locate line containing the word "not".

Action: $ grep not haiku.txt

Roslyns-iMac:writing Tquestudio$ grep not haiku.txt

Is not the true Tao, until

"My Thesis" not found.

Today it is not working

Objective: variations on options using grep

Action: $ grep **--help**

Result: lists commands.

Objective: Shortcut for reference on finding things.

Result:

find finds files with specific properties that match patterns.

grep selects lines in files that match patterns.

--help is an option supported by many bash commands, and programs that can be run from within Bash, to display more information on how to use these commands or programs.

man command displays the manual page for a given command.

$(command) inserts a command’s output in place.

**Objective: Complete lesson on Shell scripts Teaching: 30 min Exercises: 15 min**

Action:

head -n 15 octane.pdb | tail -n 5

Opened nano and created file middle.sh, added pipe to select lines 11-15 of the file octane.pdb.

Result

Roslyns-iMac:molecules Tquestudio$ nano middle.sh

Roslyns-iMac:molecules Tquestudio$ ls

cubane.pdb methane.pdb propane.pdb testfile02.txt

ethane.pdb middle.sh sorted-lenghts.txt

final.txt octane.pdb sorted-lengths.txt

lengths.txt pentane.pdb testfile01.txt

Roslyns-iMac:molecules Tquestudio$

Action: execute the command in file middle.sh

bash middle.sh

Result.

Roslyns-iMac:molecules Tquestudio$ bash middle.sh

ATOM 9 H 1 -4.502 0.681 0.785 1.00 0.00

ATOM 10 H 1 -5.254 -0.243 -0.537 1.00 0.00

ATOM 11 H 1 -4.357 1.252 -0.895 1.00 0.00

ATOM 12 H 1 -3.009 -0.741 -1.467 1.00 0.00

ATOM 13 H 1 -3.172 -1.337 0.206 1.00 0.00

Objective: double-quotes around arguments.

Action: head -n "$2" "$1" | tail -n "$3"

Note: modifies pipe in middle.sh so the filenames don't require rewriting. using double quotes Execute.

$ bash middle.sh pentane.pdb 15 5

Results:

Roslyns-iMac:molecules Tquestudio$ nano middle.sh

Roslyns-iMac:molecules Tquestudio$ bash middle.sh pentane.pdb 15 5

ATOM 9 H 1 1.324 0.350 -1.332 1.00 0.00

ATOM 10 H 1 1.271 1.378 0.122 1.00 0.00

ATOM 11 H 1 -0.074 -0.384 1.288 1.00 0.00

ATOM 12 H 1 -0.048 -1.362 -0.205 1.00 0.00

ATOM 13 H 1 -1.183 0.500 -1.412 1.00 0.00

Action: to process many files in a single pipeline and sort .pdb files by length.

$ wc **-l** **\***.pdb | sort **-n**

Result.

Roslyns-iMac:molecules Tquestudio$ nano middle.sh

Roslyns-iMac:molecules Tquestudio$ wc -l \*.pdb | sort -n

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

107 total

Command shortcuts:

* Save commands in files (usually called shell scripts) for re-use.
* bash filename runs the commands saved in a file.
* $@ refers to all of a shell script’s command-line arguments.
* $1, $2, etc., refer to the first command-line argument, the second command-line argument, etc.
* Place variables in quotes if the values might have spaces in them.
* Letting users decide what files to process is more flexible and more consistent with built-in Unix commands.

Week 5: 3 September 2019, Lesson. Loops. <https://swcarpentry.github.io/shell-novice/05-loop/index.html>

**Week 5, 3 September Objective: Lesson. Loops**

Action: Teaching: 40 min Exercises: 10 min

Action: Navigate to creatures directory.

Result: Roslyns-iMac:~ Tquestudio$ cd ~/Desktop/data-shell/

Roslyns-iMac:data-shell Tquestudio$ ls

backup north-pacific-gyre thesis

creatures notes.txt thesis\_backup

data pizza.cfg writing

molecules solar.pdf

Roslyns-iMac:data-shell Tquestudio$ cd creatures

Roslyns-iMac:creatures Tquestudio$ ls

basilisk.dat minotaur.dat unicorn.dat

Action: to look at the files first five lines.

head **-n** 5 basilisk.dat minotaur.dat unicorn.dat

Result:

Roslyns-iMac:creatures Tquestudio$ head -n 5 basilisk.dat minotaur.dat unicorn.dat

==> basilisk.dat <==

COMMON NAME: basilisk

CLASSIFICATION: basiliscus vulgaris

UPDATED: 1745-05-02

CCCCAACGAG

GAAACAGATC

==> minotaur.dat <==

COMMON NAME: minotaur

CLASSIFICATION: bos hominus

UPDATED: 1765-02-17

CCCGAAGGAC

CGACATCTCT

==> unicorn.dat <==

COMMON NAME: unicorn

CLASSIFICATION: equus monoceros

UPDATED: 1738-11-24

AGCCGGGTCG

CTTTACCTTA

Action: command head -n 2 and pipe this to tail -n 1

Note: The shell prompt changes from $ to > and back again as we were typing in our loop. The second prompt, >, is different to remind us that we haven’t finished typing a complete command yet. A semicolon, ;, can be used to separate two commands written on a single line.

When the shell sees the keyword for, it knows to repeat a command (or group of commands) once for each item in a list. Each time the loop runs (called an iteration), an item in the list is assigned in sequence to the **variable**, and the commands inside the loop are executed, before moving on to the next item in the list. Inside the loop, we call for the variable’s value by putting $ in front of it. The $ tells the shell interpreter to treat the variable as a variable name and substitute its value in its place, rather than treat it as text or an external command.

Action: Variables in loops.

$ **for** datafile **in** **\***.pdb

> **do**

> ls **\***.pdb

> **done**

$ **for** datafile **in** cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

> **do**

> ls cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

> **done**

Result

Roslyns-iMac:creatures Tquestudio$ cd ~/Desktop/data-shell/

Roslyns-iMac:data-shell Tquestudio$ ls

backup molecules pizza.cfg thesis\_backup

creatures north-pacific-gyre solar.pdf writing

data notes.txt thesis

Roslyns-iMac:data-shell Tquestudio$ cd molecules

Roslyns-iMac:molecules Tquestudio$ ls

cubane.pdb lengths.txt pentane.pdb sorted-lengths.txt

ethane.pdb methane.pdb propane.pdb testfile01.txt

final.txt octane.pdb sorted-lenghts.txt testfile02.txt

Roslyns-iMac:molecules Tquestudio$ for datafile in\*.pdb >do >ls \*.pdb >done

-bash: syntax error near unexpected token `in\*.pdb'

Roslyns-iMac:molecules Tquestudio$ for datafile in \*.pdb > do > ls \*.pdb > done

-bash: syntax error near unexpected token `>'

Roslyns-iMac:molecules Tquestudio$ for datafile in \*.pdb

> do

> ls \*.pdb

> done

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

cubane.pdb ethane.pdb methane.pdb octane.pdb pentane.pdb propane.pdb

Action:

$ **for** datafile **in** **\***.pdb

> **do**

> ls $datafile

> **done**

Result:

Roslyns-iMac:molecules Tquestudio$ for datafile in \*.pdb

> do

> ls $datafile

> done

cubane.pdb

ethane.pdb

methane.pdb

octane.pdb

pentane.pdb

propane.pdb

Note variation of two loops outputs: the datafile variable is evaluated using $datafile, and then listed using ls.

Action:

$ **for** filename **in** c**\***

> **do**

> ls $filename

> **done**

Note: \* matches zero or more characters, so any file name starting with the letter c, followed by zero or more other characters will be matched.

Result:

Roslyns-iMac:molecules Tquestudio$ for filename in c\*

> do

> ls $filename

> done

cubane.pdb

**Objective: Lesson saving a file in a loop**

Note:The text from each file in turn gets written to the alkanes.pdb file. However, the file gets overwritten on each loop interation, so the final content of alkanes.pdb is the text from the propane.pdb file.

Note: from example: >> appends to a file, rather than overwriting it with the redirected output from a command. Given the output from the cat command has been redirected, nothing is printed to the screen.

History shortcut commands

Ctrl-R enters a history search mode ‘reverse-i-search’ and finds the most recent command in history that matches the text you enter next. Press Ctrl-R one or more additional times to search for earlier matches.

!! retrieves the immediately preceding command (or use the up-arrow)

!$ retrieves the last word of the last command. type less !$ to look at the file.

Notes: A **for** loop repeats commands once for every thing in a list.

* Every **for** loop needs a variable to refer to the thing it is currently operating on.
* Use **$name**to expand a variable (i.e., get its value). **${name}** can also be used.
* Do not use spaces, quotes, or wildcard characters such as ‘\*’ or ‘?’ in filenames, as it complicates variable expansion.
* Give files consistent names that are easy to match with wildcard patterns to make it easy to select them for looping.
* Use the up-arrow key to scroll up through previous commands to edit and repeat them.
* Use **Ctrl-R**to search through the previously entered commands.
* Use **history** to display recent commands, and **!number**to repeat a command by number.

**Week 5/ 2 September 2019. Software Carpentry Shell**

Intention: Read the full shell-novice lesson and finish attempting all exercises in the shell-novice lesson. Note: Record individual commands as specific sets of intention/action/result in your learning journal.

(1) Steps: https://swcarpentry.github.io/shell-novice/

(2) Sections: Pipes and Filters, Loops, Shell Scripts and Finding Things.

**Week 5/ 2 September 2019. Software Carpentry Unix Shell**

Objectives: Redirect a command output to a file. Process a file instead of keyboard input using redirection. Construct command pipelines with two or more stages. Explain what usually happens if a program or pipeline isn’t given any input to process. Explain Unix’s ‘small pieces, loosely joined’ philosophy.

**(1) Action: Lesson on pipes and filters to combine existing commands to do new things.**

Teaching 25 mins, exercises 10 mins.

Action: cd ~/Desktop/data-shell/

**cd molecules**

**wc \*.pdb**

**wc -l \*.pdb**

Result: 20 cubane.pdb

12 ethane.pdb

9 methane.pdb

30 octane.pdb

21 pentane.pdb

15 propane.pdb

107 total

**$ wc -l \*.pdb > lengths.txt**

Note: The greater than symbol, >, tells the shell to redirect the command’s output to a file instead of printing it to the screen. The shell creates the file if it doesn’t exist. If the file exists, it will be overwritten, may result in data loss.

(2) Intention: send the content of lengths.txt to the screen using **cat lengths.txt**. Note: The cat command means ‘concatenate’ join together, and prints the contents of files one after another.

Action: **cat lenghts.txt**

Result:

20 cubane.pdb

12 ethane.pdb

9 methane.pdb

30 octane.pdb

21 pentane.pdb

15 propane.pdb

107 total

Note: also use less lenghts.txt displays a screenful of the file and stops. go back **b** quit **q**

Intention: **sort** command to sort its content.

Action: sort -n to specify a numerical rather than alpha sort.

Action: sort **-n** lengths.txt

Result: contents are sorted numerically from smallest to largest.

Error. typo in file name. Correcting by. mv lenghts.txt lengths.txt

Result. lengths.txt corrected. located in molecules.

Action: send content of lenghts.txt to screen using **cat lengths.txt**

Result: Roslyns-iMac:molecules Tquestudio$ cat lengths.txt

20 cubane.pdb

12 ethane.pdb

9 methane.pdb

30 octane.pdb

21 pentane.pdb

15 propane.pdb

107 total

Action: sort on a file numerically.

Result: Roslyns-iMac:molecules Tquestudio$ sort -n lengths.txt

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

107 total

Action: place in a temporary file.

Roslyns-iMac:molecules Tquestudio$ sort -n lengths.text > sorted-lenghts.txt

sort: No such file or directory

Roslyns-iMac:molecules Tquestudio$ sort -n lengths.txt > sorted-lengths.txt

Roslyns-iMac:molecules Tquestudio$ head -n sorted-lengths.txt

head: illegal line count -- sorted-lengths.txt

Roslyns-iMac:molecules Tquestudio$ head -n 1 sorted-lengths.txt

Result:

9 methane.pdb

Note: Using **-n 1**with **head** tells it to use the first line of the file;**-n 20**would get the first 20,. **sorted-lengths.txt**contains the lengths of our files ordered from least to greatest, the output of **head**must be the file with the fewest lines.

Note: redirect output to a new file name to avoid error.

Objective: understand >> meaning

echo The echo command prints text

Action:

Roslyns-iMac:molecules Tquestudio$ echo hello > testfile01.txt

Roslyns-iMac:molecules Tquestudio$ echo hello >> testfile02.txt

Roslyns-iMac:molecules Tquestudio$ ls

cubane.pdb octane.pdb sorted-lengths.txt

ethane.pdb pentane.pdb testfile01.txt

lengths.txt propane.pdb testfile02.txt

methane.pdb sorted-lenghts.txt

Roslyns-iMac:molecules Tquestudio$ echo hello > testfile01.txt

Roslyns-iMac:molecules Tquestudio$ echo hello >> testfile02.txt

Roslyns-iMac:molecules Tquestudio$ ls

cubane.pdb octane.pdb sorted-lengths.txt

ethane.pdb pentane.pdb testfile01.txt

lengths.txt propane.pdb testfile02.txt

methane.pdb sorted-lenghts.txt

Roslyns-iMac:molecules Tquestudio$

Result. > overwrites file. >> add text to file output.

Objective: Understand appending data.

Action: Use **head** command to print lines from the start of a file. Use **tail** command to print lines from the end of a file.

$ sort **-n** lengths.txt | head **-n** 1

Note: vertical bar is a pipe. (uppercase backslash.) It tells the shell to use the output command on the left as the input to the command on the right.

Action: send the output of **wc** to **sort**, and output to **head**.

$ wc **-l** **\***.pdb | sort **-n**

Result: Roslyns-iMac:molecules Tquestudio$ sort -n lengths.txt | head -n 1

9 methane.pdb

Roslyns-iMac:molecules Tquestudio$ wc -l \*.pdb |sort -n

9 methane.pdb

12 ethane.pdb

15 propane.pdb

20 cubane.pdb

21 pentane.pdb

30 octane.pdb

107 total

Action: send the output to another pipe, to **head** the pipeline:

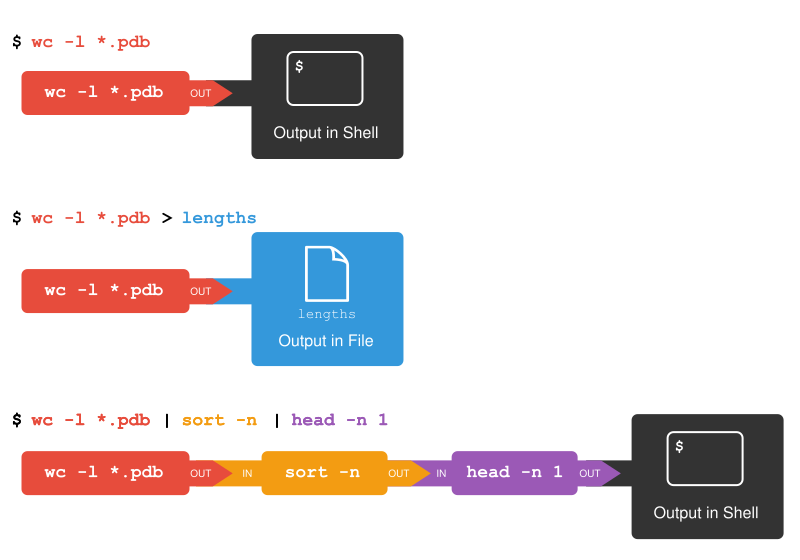
$ wc **-l** **\***.pdb | sort **-n** | head **-n** 1

Result:

Roslyns-iMac:molecules Tquestudio$ wc -l \*.pdb |sort -n | head -n 1

9 methane.pdb

Note: visualisation of output destinations



Action: find the 3 files which have the least number of lines

Result: Roslyns-iMac:molecules Tquestudio$ wc -l \* | sort -n | head -n 3

0 sorted-lenghts.txt

1 testfile01.txt

2 testfile02.txt

Note: a filter is a program: **wc** or **sort** and transforms a stream of input to a stream of output. The pipe character **|** is used to connect the output from one command to the input of another.**>** is used to redirect standard output to a file.

Note: the program reads lines of text from standard input and writes lines of text to standard output; write your programs this way so that you and other people can put those programs into pipes to multiply their power.

**Objective: Answer question, What text passes through each of the pipes and the final redirect in the pipeline below?**

Action: $ cat animals.txt | head -n 5 | tail -n 3 | sort -r > final.txt

Result:

* the cat command means join together and prints the contents of files one after another
* The pipe character | is used to connect the output from one command to the input of another
* head command takes the first 5 lines of animals.txt
* -n Using -n 1 with head tells it to use the five lines of the file
* the last three lines using the tail command are taken from the previous five to print lines from the end of a file
* the sort command sorts 3 lines in reverse order contents of all
* -r recursive option
* > is used to redirect standard output to a file called final.txt

Result: check with command cat final.txt.

Roslyns-iMac:data-shell Tquestudio$ cd data

Roslyns-iMac:data Tquestudio$ ls

amino-acids.txt animals.txt morse.txt planets.txt sunspot.txt

animal-counts elements pdb salmon.txt

Roslyns-iMac:data Tquestudio$ cd animals

-bash: cd: animals: No such file or directory

Roslyns-iMac:data Tquestudio$ ls

amino-acids.txt animals.txt morse.txt planets.txt sunspot.txt

Roslyns-iMac:data Tquestudio$ cat animals.txt | head -n 5 | tail -n 3| sort -r > final.txt

Roslyns-iMac:data Tquestudio$ ls

amino-acids.txt animals.txt final.txt pdb salmon.txt

animal-counts elements morse.txt planets.txt sunspot.txt

Result: Roslyns-iMac:data Tquestudio$ cat final.txt

2012-11-06,rabbit

2012-11-06,deer

2012-11-05,raccoon

**Objective: Lesson on pipe construction**

Action: $ cut **-d** , **-f** 2 animals.txt

Result:

Roslyns-iMac:data Tquestudio$ cut -d , -f 2 animals.txt

deer

rabbit

raccoon

rabbit

deer

fox

rabbit

bear

Note: The**uniq** command filters out adjacent matching lines in a file. To extend this pipeline remove the duplicates. Use **cut**command to remove sections of each line in the file, **cut**expects the lines to be separated into columns by a **Tab**character. A character used in this way is a called a **delimiter**. the **-d**option specifies the comma as our delimiter character. **-f**option to specify what to extract in the second field (column).

Action:

$ cut **-d** , **-f** 2 animals.txt | sort | uniq

Result: Roslyns-iMac:data Tquestudio$ cut -d , -f 2 animals.txt | sort | uniq

bear

deer

fox

rabbit

raccoon

Note: The uniq command has a -c option which gives a count of the number of times a line occurs in its input. The command to produce a table that shows the total count of each type of animal in the file **cut -d, -f 2 animals.txt | sort | uniq -c**

Objective: Lesson on checking files

Action: $ cd north-pacific-gyre/2012-07-03

$ wc **-l** **\***.txt

Result:

Roslyns-iMac:data Tquestudio$ ls

amino-acids.txt animals.txt final.txt pdb salmon.txt

animal-counts elements morse.txt planets.txt sunspot.txt

Roslyns-iMac:data Tquestudio$ cd ~/Desktop/data-shell

Roslyns-iMac:data-shell Tquestudio$ ls

backup molecules pizza.cfg thesis\_backup

creatures north-pacific-gyre solar.pdf writing

data notes.txt thesis

Roslyns-iMac:data-shell Tquestudio$ cd north-pacific-gyre/2012-07-03

Roslyns-iMac:2012-07-03 Tquestudio$ ls

NENE01729A.txt NENE01751A.txt NENE01843A.txt NENE01978A.txt NENE02040A.txt NENE02043A.txt goostats

NENE01729B.txt NENE01751B.txt NENE01843B.txt NENE01978B.txt NENE02040B.txt NENE02043B.txt

NENE01736A.txt NENE01812A.txt NENE01971Z.txt NENE02018B.txt NENE02040Z.txt goodiff

Roslyns-iMac:2012-07-03 Tquestudio$

Roslyns-iMac:2012-07-03 Tquestudio$ wc -l \*.txt

300 NENE01729A.txt

300 NENE01729B.txt

300 NENE01736A.txt

300 NENE01751A.txt

300 NENE01751B.txt

300 NENE01812A.txt

300 NENE01843A.txt

300 NENE01843B.txt

300 NENE01971Z.txt

300 NENE01978A.txt

300 NENE01978B.txt

240 NENE02018B.txt

300 NENE02040A.txt

300 NENE02040B.txt

300 NENE02040Z.txt

300 NENE02043A.txt

300 NENE02043B.txt

5040 total

Action: Roslyns-iMac:2012-07-03 Tquestudio$ wc -l \*.txt | sort -n | head -n 5

Result:

240 NENE02018B.txt

300 NENE01729A.txt

300 NENE01729B.txt

300 NENE01736A.txt

300 NENE01751A.txt

$ wc **-l** **\***.txt | sort **-n** | tail **-n** 5

Check file with 240 lines, rerun data.

Result: Roslyns-iMac:2012-07-03 Tquestudio$ wc -l \*.txt | sort -n | tail -n 5

300 NENE02040B.txt

300 NENE02040Z.txt

300 NENE02043A.txt

300 NENE02043B.txt

5040 total

$ ls **\***Z.txt

Result: Roslyns-iMac:2012-07-03 Tquestudio$ ls \*Z.txt

NENE01971Z.txt NENE02040Z.txt

Note: Wildcard commands

Delete them using **rm**,

select files using the wildcard expression **\*[AB].txt**.

**\***matches any number of characters; the expression **[AB]** matches either an ‘A’ or a ‘B’

To remove files in current directory: **rm \*.txt**

Note: Summary of commands:

* cat displays the contents of its inputs.
* head displays the first 10 lines of its input.
* tail displays the last 10 lines of its input.
* sort sorts its inputs.
* wc counts lines, words, and characters in its inputs.
* command > file redirects a command’s output to a file (overwriting any existing content).
* command >> file appends a command’s output to a file.
* first | second is a pipeline: the output of the first command is used as the input to the second.
* The best way to use the shell is to use pipes to combine simple single-purpose programs (filters).

**Week 4 Objective complete homework** noted on cloudstor:

Action: Work on: <http://swcarpentry.github.io/shell-novice/> and Do episodes: x Introduction, Navigating Files and Directories, Working with Files.

Results: Annotation of lessons and objectives noted below. Updated Learning Journal placed submitted to Github.

**Week 4/ 2 September 2019.**

Objective: Lesson moving files and directories

Action: cd ~/Desktop/data-shell/

change file name moving, mv is argument (options mv -i or mv --interactive

mv thesis/draft.txt thesis/quotes.txt

move quotes.txt

$ mv thesis/quotes.txt .

Move files from parent to current directory

$ mv ../analyzed/sucrose.dat ../analyzed/maltose.dat .

Note: .. refers to the parent directory (i.e. one above the current directory) and that . refers to the current directory

**Objective: lesson copying files and directories**

Action: **CP** command works like **mv** but copies file

**cp quotes.txt thesis/quotations.txt**

copy a directory and contents using the recursive option **-r**

cp -r thesis thesis\_backup

Result. thesis:

quotations.txt

thesis\_backup:

quotations.txt

**Objective: rename files eg correct typo**

Action: mv statstics.txt statistics.txt

Results: statistics.txt name corrected.

**Objective: remove files and directories**

Action: rm quotes.txt

Note: deleting is forever, Use **-i** before removal to confirm delection. eg/ rm -i thesis\_backup/quotations.txt

rm -r -i

Note: rm  only works on files, not directories. rm can remove a directory and all its contents if the recursive option -r is used, and it will do so without any confirmation prompts:

**Objective lesson: Multiple files and directories.**

Action: using wildcards to copy or move several files. \* is a wildcard matches zero or more characters. ? is also a wildcard matching on character

Steps: to duplicate a folder structure.

Notes:

* **cp old new**copies a file.
* **mkdir path** creates a new directory.
* **mv old new** moves (renames) a file or directory.
* **rm path** removes (deletes) a file.
* **\*** matches zero or more characters in a filename, so **\*.txt**matches all files ending in .txt.
* **?**matches any single character in a filename, so ?.txt matches a.txt but not any.txt.
* Use of the Control key may be described in many ways, including Ctrl-X, Control-X, and ^X.
* The shell does not have a trash bin: once something is deleted, it’s really gone.
* Most files’ names are something.extension. The extension isn’t required, and doesn’t guarantee anything, but is normally used to indicate the type of data in the file.
* Depending on the type of work you do, you may need a more powerful text editor than Nano.

Result: successful execution.

**Week 4/ 2 September 2019.**

**Objective:** Lesson on working with files and directories create, copy, and delete files and directories and edit files

Action. Lesson 30 mins and exercise 20min. Commence. 22:39.

Steps: change directory to cd Desktop/data-shell

**ls -F** list what's in the directory

Created a directory. **mkdir thesis**

Note: When creating file names:

* Don’t use spaces.

Spaces can make a name more meaningful, but since spaces are used to separate arguments on the command line it is better to avoid them in names of files and directories. You can use - or \_ instead

* Don’t begin the name with - (dash).
* Commands treat names starting with - as options.
* Stick with letters, numbers, . (period or ‘full stop’), - (dash) and \_ (underscore).
* Many other characters have special meanings on the command line. We will learn about some of these during this lesson. There are special characters that can cause your command to not work as expected and can even result in data loss.
* If you need to refer to names of files or directories that have spaces or other special characters, you should surround the name in quotes ("").

change director to thesis: ls -F

created a text file called Nano nano draft.txt (opens text editor) saved file and placed in thesis. Follow prompt at footer of nano. draft.txt

**Ctrl + 0** to write data to disk. **Return** to save.

**Ctrl + x** to quit editor and return to shell

Notes:

* The **touch** command generated a new file called **my\_file.txt**in the current directory.
* inspect the file with **ls -l**, the size of my\_file.txt is 0 bytes. text is blank.
* The touch command to generate a blank text file for an output file.

Note: use two-part names. filename extension indicates data type

**Objective: lesson on pipeline organising files**

Action: create directory: north-pacific-gyre, directory 2012-07-03. Directory named: year-month-day. Naming convention of lab unique ten character ID NENE01729A.txt

Note:

Results: The file system is responsible for managing information on the disk. Information is stored in files, which are stored in directories (folders). Directories can also store other directories, which forms a directory tree.

* cd path changes the current working directory.
* ls path prints a listing of a specific file or directory; ls on its own lists the current working directory.
* pwd prints the user’s current working directory.
* / on its own is the root directory of the whole file system.
* A relative path specifies a location starting from the current location.
* An absolute path specifies a location from the root of the file system.
* Directory names in a path are separated with / on Unix, but \ on Windows.
* .. means ‘the directory above the current one’; . on its own means ‘the current directory’.

**Week 4/ 2 September 2019. Objective**: Lesson understand navigating files and directories

Action. Lesson 30 mins and exercise 10min. Commence. 17:46.

Action. Note file system organises data into files and directories (folders).

Steps: Command **pwd** (print working directory). Note: directories: bin, data, users, tmp. Current working directory. / slash at front of file is a root directory, inside a name a separator. **ls** prints names of files in current directory **-F** (switch or flag) tells **ls** to classify output **ls -F**  Note: command, option, argument **ls --help** ... **ma ls** options.

**man ls** (flag)for a description of ls command and its options. **Q** to quit.

The **-l**option makes **ls**use a long listing format, shows the directory name, file size and last modification. Using **-h**option and  **-l**option shows smaller readable file.

List recursively and by time. **ls -R** (all directories) **ls -l** timestamps

see Desktop directory with **ls -F** Desktop command **ls**  the **-F** option and the argument Desktop. The argument Desktop tells **ls** that we want a listing of something other than our current working directory:

change location command: **ls -F Desktop/data-shell**

Output:

Roslyns-iMac:~ Tquestudio$ ls -F Desktop/data-shell

creatures/ molecules/ notes.txt solar.pdf

data/ north-pacific-gyre/ pizza.cfg writing/

Summary of paths:

1. cd . current directory
2. cd / root directory
3. cd /home/amanda change directory and location
4. cd ../.. goes up two levels
5. cd ~ user’s home directory
6. cd home navigate into a directory
7. cd ~/data/.. navigate into a directory
8. cd go back to the user’s home directory
9. cd .. go up one level

**Week 4/ 2 September 2019. Objective: Lesson objective introduction to the Unix Shell**

Result. Understand use of command-line interfaces CLI via Terminal application. Note read-evaluate-print loop. The Shell -- Bash. Command **ls** Note: If the shell can’t find a program it will print an error message as **ks**. command not found. Note: the grammar of a shell allows tools to be combined into pipelines to manage large volumes of data automatically. Improves workflow and repeat.

**Week 4/ 2 September 2019. Objective: Setup for lesson objective introduction to the Unix Shell**

Action: Opened http://swcarpentry.github.io/shell-novice.

Steps: Downloaded [data-shell.zip](http://swcarpentry.github.io/shell-novice/data/data-shell.zip) on Desktop. Files Unzipped A new folder called data-shell on Desktop. Opened terminal and typed **cd**, home folder is the working directory. Note: default Unix Shell for Mac OS is Bash access via Terminal in Applications folder.

**30 August 2019 Objective: Understand pipes and filters from class notes.**

Action: Connect pipe to a program and use output of the program as input to another program. Sort lines in a file. Example, sort 23000 user names from Titter.

Commands and steps for "knowledge in the head":

ls to look into directory

cd change directory

wc \*.pdb. (shows word count)

-l \*.pdb (shows line count)

Output useful for random stuff. Not useful for saving.

Use angle bracket to redirect output to a file

Use arrow up to go back in history

Sort default is alpha numeric

Wc -l \*.pdb >

View file contents cat lengths.txt

Redirection of a filtered output of a file into a file and viewed the file.

cat ~/Desktop/data-shell.zip

control l clear.

Input text. sort -n

Wc get word count -l run list of files \*.pdb pipe links to a file sort top output

**Week 4. 23 August: Spreadsheets for Social Sciences** https://datacarpentry.org/spreadsheets-socialsci/

Action: Export the csv. View it in a text editor like Atom.io, Sublime Text, or notepad++ Think about the benefits of an always-readable and not tied to a subscription or specific program data format.

**1. Lesson objective: Dates as Data. Do tasks for data organization and practices for effective data wrangling**

Action. 1 10 mins. Dates as data. Dates are inconsistent in presentation which is problematic for data automation. Fact: In Excel date is stored a number 41822, and 41822 + 90 = 41912 which Excel interprets as the 30 September 2014. Regional variations of date cause data errors, best to note date data (month, day, and year).

Action. Tasks, split dates into component value for ease in handling. Action for exercise: add three new columns, input month, day, year as numeric (general) entry. Saved file.

Action. 2. Default year. Noted spreadsheet defaults to current year, where no value is applied. Can throw out data through the date default. Note to use caution with historical data. Excel translate post 1900 dates into internal format, resulting in mixed data.

**2. Lesson objective: Quality assurance—lessons to limit error in data entry.**

Note. Validate data on input. Age should be numeric greater than 0 and less than 120. Can note these as example of acceptable values for cells.

Action: Restricting data to a numeric range and Restricting data to entries from a list.

* Step one. Copy data to new tab. Task using data sheet titled. SAFI\_clean. Selected no\_members column amended **data validation field** for range minimum 1 and maximum 30. Tested result. Working okay.
* Step two. Task add message for error return. **Input Message**.
* Step three. Customised resulting message for years living for a data range between 0 and 120 using **Input Message**.

Action: Restrict data to entries from a list. Followed action using **Data Validation**, Settings, added an input message: grass, muddaub, burntbricks, sunbricks, cement. Drop box for controlled vocabulary now functional.

3. Lesson objective. Exporting data. To practice exporting data from spreadsheets and store in a universal file format.

Step one. Open **file** and **save as** csv. Note: CSV files can be opened in Excel. File saved as SAFI\_clean\_2.csv

Note: When using data that contains commas, enclose the fields with double quotes

**Task three: Data carpentry exercise: formatting problems** <https://datacarpentry.org/spreadsheets-socialsci/02-common-mistakes/>

Action: 20 Min duration. Start: 22.51 End 22:59

Note: Observe errors in tables.

Adding here for ready reference, tidy data rules noted as key points to this section:

* one table for one spreadsheet.
* keep data in one tab
* use a new tab and copy when cleaning the data up
* note zeros as zeros [0]
* use null value for missing data.
* avoid formatting for presentation
* keep comments in a separate column
* place unit of data in the column header
* place one piece of information in each cell, avoid multiple on one cell
* don’t use special characters in the data
* don’t use spaces, numbers or special characters in the column header

**Task two: Data carpentry exercise: formatting data tables in spreadsheets.**

Action: 15 min duration, 15 min exercises. Start 22:21. Complete 22:47

Noted importance of using well-formatted tables from the outset. Note best to automate conversion for optimal layout/format where different software or interface requires it.

1. Open messy data sheet.

Note: Messy has two tabs. The data table formats are inconsistent.

* the spreadsheet has three tables. Mixed use of numeric and text which should be consistent. Difficult to read both tables because of inconsistent formatting. Information/data is confusing. Zeros should be added to record zero. Missing data should have null entry. Comments should be in a separate column not mixed in the data columns. The units of the measures should appear in the header for each column. Special characters should not be used in data headings or in the data.

2. Open clean version of data.

Question: What is not immediately obvious to me about this data? What questions would I need to know the answers to in order to analyze and interpret this data? What types of metadata that should be recorded about this dataset.

Result: It is not clear what the questions were for the data results, this could be noted. Metadata is free text for comments, units used in this sheet they disrupt the format of the data file and better placed in a codebook. Use a standard, such as Data Documentation Initiative (DDI) for reference: <http://www.ddialliance.org/>

Task one: Data carpentry

Read and do all exercises. Exercises should be recorded in your Learning Journal and uploaded to cloudstor.

1. <https://datacarpentry.org/spreadsheets-socialsci/00-intro/index.html>
2. <https://datacarpentry.org/spreadsheets-socialsci/01-format-data/index.html>
3. https://datacarpentry.org/spreadsheets-socialsci/02-common-mistakes/index.html

*Task note: For learning journal add 2 examples of problems in data produced by your discipline. You will get a “Well done indeed” from me if you can find these problems in published datasets in your discipline, and cleaning one of your disciplines’ datasets could very well be your Proof of Concept.*

**Task one: Data carpentry exercise.**

Objective: Complete learning task <https://datacarpentry.org/spreadsheets-socialsci/00-intro/index.html>

Action: 15 min duration with exercises 3 min. Open Microsoft Excel.

Noted text of problems with spreadsheets. The graphical interface can complicate replicating steps. Data quality can be compromised by errors introduced through applying formula to neighbouring cells.

*Starting exercise: How many people have used spreadsheets in their research?*

*How many people have accidentally done something that made them frustrated or sad?*

I answered these questions in the class sheet on Github. No major problems encountered with spreadsheets so far.

Result: Start. 22:14 Completed. 22:19

### **Learning Journal week 1 action notes:**

Note: Scoping Exercise is not due next week, due in week 3 instead.

Objective: Think about frustrating and repetitive practices in your research process from collection to publication. Key concepts and terms for drafting, with reference to key authors.

Result: drafting notes for scoping exercises.

Objective: Try to restore a file from a 6 month or older backup. Document how it goes.

Result: I use the cloud, so I can access my files from multiple devices. It’s worked well so far. Only one glitch two years back when I muddled up passwords for devices and the cloud locked me out. After a day on the phone to Apple genius, recovered cloud and device access okay. I’ve sometimes had trouble with version control between the laptop and desktop computers.

Objective: Sign into <https://foar705.slack.com/> and say hi. Take a look at two year old convos.

Action: account active in Slack. Viewing #random #general channels and direct messaging.

Action: Sign into cloudstor: [https://cloudstor.aarnet.edu.au/plus/index.php](https://slack-redir.net/link?url=https%3A%2F%2Fcloudstor.aarnet.edu.au%2Fplus%2Findex.php) and let us know in slack when you have, so we can add you to a group.

Result: signed into cloudstor okay.

Objective: Look into "project management tools" like Trello, Jira, and Asana, and form opinions.

Result: I’ve used Slack for professional coordination of an online journal, it works well however the organization did not purchase the full use, so files were wiped after seven days. That meant we had to use additional tools to document resource files and links. As this was standard operating procedure it worked okay. However, team members often cited, oops message lost, gobbled by Slack. Also used Slack for coordination of team in a Tedx project.

Objective read: [https://impossiblehq.com/an-unexpected-ass-kicking/](https://slack-redir.net/link?url=https%3A%2F%2Fimpossiblehq.com%2Fan-unexpected-ass-kicking%2F) Completed.

Extra optional: Read program or be programmed by Rushkoff. Completed.

Result: material linked in references with summary entry.

Objective: Sign into: [https://github.com](https://slack-redir.net/link?url=https%3A%2F%2Fgithub.com) and tell me your username on slack. Completed.