Library Carpentry

Michigan State University

Nov 10-13, 2025

Workshop information

Useful links

- Zoom link: https://msu.zoom.us/j/98144353527
- Workshop website: https://abigailsparling.github.io/2025-11-10-msu-online/
- Library Carpentry: https://librarycarpentry.org/
- Code of Conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

Instructors

- Rachel Fernandez
- Allie Tatarian
- Kyla Jemison
- Abigail Sparling
- Broncio Aguilar-Sanjuan

Helpers

- Jonathan Barber
- Eleanor Carr
- Titi Kou-Herrema
- Rachel Fernandez
- Sruthin Gaddam
- Abigail Sparling
- Claire Kopenhafer
- Kyla Jemison

- Pre-workshop Survey - https://carpentries.typeform.com/to/wi32rS?slug=2025-11-10-msu-online
- Post-workshop Survey - https://carpentries.typeform.com/to/UgVdRQ?slug=2025-11-10-msu-online

---

Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/policies/coc

---

Introductions

Please write: name, pronouns, role/department, one thing that you'd like to learn today

- Suzi Teghtmeyer, she/her, librarian in collections, reference and a liaison to mutiple departments. Today I'd like to learn better means of database management, aside from Microsoft products.
- Terence O'Neill, he/him, Entrepreurship Librarian, I'm excited to learn about tools for cleaning up tabular data, possibly for use with computational analysis.
- Sruthin Gaddam, he/him, UX and Accessibility Librarian, MSU Libraries. Today I'd like to learn how to efficiently clean up large CSV files.
- Julie Taylor, she/her, Publishing Services Coordinator, MSU Libraries. I currently use Excel + Acess to manage data & statistics. I'd like to learn alternatives, for better/worse.
- Mike Laney, he/him, Head of the Vincent Voice Library and Kline Digital and Multimedi Center. I'd like to learn how to more efficiently clean/work with data and become comfortable w/ using Github
- Eli Wachter, they/them, Data Visualization Librarian, MSU LIbraries. I'd like to learn better strategies for verison control
- Sarah Mainville, she/her, Media Preservation Librarian MSU Libraries. Become more comfortable using github and other tools to help with data collection/curation.
- Alex Hauser, she/her, Business Librarian at MSU Libraries. Interested in data/spreadsheet cleanup and introduction to other tech/tools
- Colleen Lyon, she/her, Engineering Librarian at MSU Libraries. Try to get better with using and manipulating tabular data.
- Leila Sherbini, she/her, Media Preservation Resident Librarian, getting more comfortable with coding and command language
- Christine Peffer, she/her, Academic Specialist with H-Net/DH@MSU, becoming more comfortable with cleaning up CSV and training others to do the same
- Kin Sing Lee, he/him/his, faculty, interested in learning some basic skill of Data clean up, organization for my group. We are interested to share our experimental data compliance to FAIR Data principle in future publications.
- Becky Holm, she/her, Law Library Assitant, to learn as much as I can
- Kristen Mapes, she/her, Interim Director of Digital Humanities, here to learn what I can (have to run for a couple of meetings over the days)
- Nick DeVolder, he/him, Library Assistant Turfgrass Information Center, interested in data management best practices

Day 1: Tidy Data

Lesson Material: https://librarycarpentry.github.io/lc-spreadsheets/
Data Download: https://librarycarpentry.github.io/lc-spreadsheets/data/training_attendance.xlsx
Minute Card: https://forms.office.com/r/cKa0nyC9hz

Agenda

- Using spreadsheet programs for data organisation
- Formatting data tables in spreadsheets
- Dates as data
- Basic quality assurance and control
- Exporting data from spreadsheet
- Wrap-up/Minute card

Notes

- Intro spreadsheets
- Overview of the lesson: this lesson will cover good data practices, dates as data
- https://glosario.carpentries.org/en/
- Breaking Jargon Activity (5 minutes)
    - Room 1 (Christine P, Kyla J, Nick D)
        - Relational database
        - Linked data
        - Git/github
    - Room 2 (Eli, Sarah, Becky)
        - Any accronyms (such as VB - Visual Basic, CSV)
        - Difference between dataset and data frame
        - Regular expressions
        - Linked data
    - Room 3 (Terence, Mike, Sruthin)
        - CSV - what's significant about tabular data as CSV
        - Regular expressions
        - Versioning in git
    - Room 4 (Alex H, Leila S, Suzi T)
        - shell
        - pipes (different in exel vs command line?)
    - Room 5 (Julie, Colleen)
        - shell
        - scripting vs coding
        - git vs GitHub
        - terminal vs command line/command prompt (mac/linux vs pc?)
    - Reporting out
        - Help with data and coding terms: https://glosario.carpentries.org/en/

        - Dataset vs data frame: dataset is the actual data; dataframe is a structured representation of that data (when you need to manipulate it or analyze it) (hope this is somewhat helpful? haha)
        - Some common interests in Regular Expressions (Room 2, 3)
        - Shell (open the shell)! We will talk about this later in the week; ways to interact with your system
        - Pipes: use pipes in different ways
        - More will be addressed later in the week :)
- Activity: Discussion Questions-->**SPREADSHEETS**
    - I use Excel often. I wish I had a better strategy for saving formulas I use frequently
    - I use spreadsheets a lot. Useful for sorting data, keeping track of stats, for prepping data for upload to repositories. Dates are easy to screw up (particularly in Excel).
    - One mistake I have made is going from Excel to CSV and losing my additional tabbed tables
    - I use spreadsheets (Excel) all the time, mainly for invoicing and data collection. Every month, I combine all the invoices from the month into one worksheet and have a main spreadsheet to collect & tabulate all the relevant data from the month's invoices. I use VB modules to automate the monthly compilations. I also then copy/paste the "main" spreadsheet of all the important data into Access so that I can run queries. I would be interested in learning more about Pivot Table. A frustration would be the limitations data storage in spreadsheet, but I started with .xls not .xlsx formats. I've also created calendars and other sheets that use lots of fomulas and conditional formatting. Also for converting units/math.
    - I use Excel for calculations, concatonating columns, downloads from websites, converting text to tables back to text, filtering columner data, etc. I use them a lot!
    - I use Excel, Google Sheets, AirTable for different things such as inventories for objects, collecting and sharing metadata (descriptive, technical, preservation). In AirTable we use it for this collection of metadata and also assigning tasks and project management. Frustrations are consistent formatting across different platforms or when the tool autocorrects things in a way that I do not want.
    - I use Google sheets for workflow tracking with student employees that are entering data in a bibliographic database. I use Excel for tracking journals that have been processed for metdata records. Both have limited math/equations, a lot of text storing/sorting.
    - I use Excel and google sheets often for project management tasks/tracking statuses while working with students and other collaborators; collecting metadata; tracking patterns; etc
    - Primarily using Google Sheets and Numbers for inventories and collecting metadata.
    -
    - I've mostly used spreadsheets (Excel, Google Sheets) to collect/troubleshoot data in collections. Inventory maintenance like titles that need to be marked as restricted in a special collection, things that are marked as in progress that need to be checked in, reviewing call numbers, etc. I think the most frustrating thing is to create uniform naming system when working with large sets.
    - I have used spreadsheets to track tasks/projects, for map making, and for data. I use them fairly often. Recently I've used them for sorting data which has been a revelation and really shown me the capability they have for certain activities. My biggest mistake/frustration was merging cells early on and deeply regretting it/not getting them back to normal. But also dates are a nightmare for me - lots of recordings with imprecise dates and trying to get everything to be formatted semi-coherently.
    - Excel or Google Sheets almost exclusively. Excel for projects at work, Google sheets for

survey results from a Google Form.

- Formatting data tables and spreadsheets
    - Common mistake is using spreadsheet like notebook software
    - Has to be formatted for computer to read
    - Important: start with well-formated table
    - Create new file for analyzed data to be able to track changes and reproduce file
    - Readme file can be used to track the work with data
    - Cardinal Rules for Tidy Data
        - Put all your variables in its own columns
        - Put each observation in its own row
        - Don't combine multiple pieces of info in one cell
        - Leave the raw data raw (create a new tab or spreadsheet before manipulation)
        - Export clean data to text based format (like CSV)
    - (Looking at a RDM training example):
        - D column: PGR|PDRA|other: too many information in the same column
        - Make sure each entry has its own row (two dates should not be in the same row)
    - Messy Data Activity --> 15 minutes
        - Checking 2016 and 2017 tabs, and combine them into one tab
    - Report out
        - Room 1
            - Used text to column, combined open access and RDM and added column for cancelled
            - Able to combine
        - Room 2
            - Question: Multiple columns for delivered by to accomodate multiple? Depends on how you want to measure later on
            - The importance of document: because we don't know how these date will be used later on, so document everything!
        - Room 3
            - Noticed: there is one session that had two trainers
            - Discussions on column names
            - Added a column "status" (to deal with sessions that are canceled, or even for "postponed")
            - Added a column "total attendees"
            - Have a readme file will be great: to document the choices made throughout this process
        - Room 4
            - A very important notes: should we leave empty cell or NOT?
            - Read: https://librarycarpentry.github.io/lc-spreadsheets/02-common-mistakes.html
        - Room 5
            - Focus on "dates"
            - It is DIFFICULT to format the data without knowing what this data will be used for.

Messy Data Assignment

1. First download the data
2. Open up the data in a spreadsheet program.

3. You can see that there are three tabs. Various people have recorded training attendance statistics over 2016 and 2017, and they have kept track of the data in their own way. Now you're being asked to evaluate the training programme and you want to be able to start doing statistics with the data.
4. With the person next to you, work on the messy data so that a computer will be able to understand it. Clean up the 2016 and 2017 tabs, and put them all together in one spreadsheet.
5. After you go through this exercise, we'll discuss as a group what you think was wrong with this data and how you fixed it.

Dates in spreadsheets

- Formatting can cause problems in display from different spreadsheet programs
- There are different data formats (e.g. day-month, day-of-the-week-month-day-year, etc.)
- Different spreadsheet programs treat the same date format differently
- Date formats also change between countries (e.g. day-month-year vs. month-day year)
- Example with the messy data from the activity: Excel allows to force a certain date format
    - Format --> Cells --> Date or Format --> Cells --> Custom
    - Dates in Excel are calculated as days since 12-31-1899 and can therefore be added that way (e.g. 4/29/2015 + 37 = 6/5/2015)
- Date functions: examples include YEAR(), MONTH(), DAY(), DATE(Year, Month, Day), DATEVALUE(), and TEXT(Value, "Formatting code to apply") in Excel and YEAR(), MONTH(), DAY(), DATE(Year; Month; Day), DATEVALUE("Text") in LibreOffice (another open-source spreadsheet software)

- Challenge: Exchanging Date Information Between Formats
    - What happens to the dates in the dates tab of our workbook if we save an Excel sheet as a .CSV and then open it as a plain text file (e.g. in TextEdit)? If we open it in Excel?
    - Try this! What were the results?
    - Discussion:
        - File --> Save As --> choose file format --> choose location --> click "Save"
        - There is no year included when opened as plain text
        - If you convert back to Excel, the default is the present year
- Better date formats:
    - Year-month-day format in separate columns
    - Each element can be easily obtained from an Excel formatted data column using the functions YEAR(), MONTH(), and DAY()
    - It is important to format the resulting year, month, and day columns as plain text rather than as a date format
    - However, these columns are now formulas. We know that we can copy and paste as values, but what are some other ways to transfer between files as values in a more efficient manner? One answer to this would be that you should consider why you're copying and pasting values/formulas between spreadsheets, and asking if there are better ways to move data.
    - A note: if you save an Excel file as a plain CSV, the formulas take on their values rather than saving as formulas
    - ISO 8601: universal date format. There is not currently a default format to directly convert to ISO 8601, but this can easily be added to the custom formats tab
        - The advice is to manually input dates using YYYYMMDDhhmmss as a plain text string rather than relying on Excel's cell formatting menu

Basic Quality Control (QC) techniques

- QC --> check for errors *after* entering data
- Save all original files and create a new file to do your QC
- Activity:
    - Sort len_hours column in messy sample data from largest to smallest
        - What do you notice?
            - Cells with numbers and text are sorted differently than cells with only numbers
            - The units differ - some used minutes and some used hours (e.g. 15 could mean .25 hours, 15 minutes, or 15 hours)
            - 1.5 hours and 90 minutes look like different values
- Conditional formatting can be used to flag inconsistent values
    - A note: Odd values/Data QC can be done programmatically using R or Python also
    - Home --> Conditional Formatting
    - For instance: Gradient formatting can ID oddly high or low values
    - Cancelled events have no attendees

Exporting data from spreadsheet

- We have identified some problems with storing things as .xls or .xlsx (Excel) formats, including issues with formulas and dates
- Excel format is also proprietary - it's owned by Microsoft and may not always be available/be available everywhere
- CSV files (comma-separated values) is a universal, open and static format because they are plain text. Because of this, you can also open them in many different softwares.
- To export a data as a CSV: File --> Save As --> Comma-separated format
- CSV files cannot be saved with multiple sheets, so the first sheet is the only one that gets saved

Questions

- How can units of measurement be removed?
    - To convert between units: CONVERT(CELL, "mn", "hr")
    - "Find and replace" to remove text in numeric cells
- To get the ISO dates by formula: TEXT(CELL, "YYYYMMDDhhmmss")

Day 2: Regular Expressions

Lesson Material: https://librarycarpentry.github.io/lc-data-intro/
Data Download: https://github.com/LibraryCarpentry/lc-data-intro/tree/main/episodes/files just the PLS_FY17.zip file

- Google Sheets - https://docs.google.com/spreadsheets/d/11A7GZs5p9tsqlDYg4LLmRpROQdQDd1NluWq3sXeQ4yo/edit?usp=sharing

Minute Card: https://forms.office.com/r/r3a7jeyt4B

Attendance

Name and one word or phrase that stuck in your mind from yesterday

- Sruthin Gaddam. The word that stuck in my mind was ISO format.
- Rachel Fernandez (Helper), reproducible (I said this a lot)
- Colleen Lyon, "ctrl D for filling a column"
- Eli Wachter. "Leave raw data raw."
- Becky Holm, recording the steps that you take for replication purposes
- Nick DeVolder, All variables in their own column
- Alex Hauser, add a column instead of formatting to note data
- Terence O'Neill- same as Eli's; don't mess with the raw data.
- Christine Peffer - clean data is foundational for analysis!
- Suzi Teghtmeyer, clean data and ReadMe.txt!
- Julie Taylor, ISO formatting for dates
- Kristen Mapes (just loving everyone's above me!)
- Leila Sherbini, text to column
- Mike Laney, null data

Notes

- Agenda:
  - *Intro (30 minutes)*
  - *Matching and Extracting Strings (30 minutes)*
  - *15-minute break*
  - *Regex in Google Sheets (30 minutes)*
  - Optional - Software Setup for Day 3/4 or Regular Expressions in MarcEdit
- Link to the spreadsheet: https://docs.google.com/spreadsheets/d/11A7GZs5p9tsqlDYg4LLmRpROQdQDd1NluWq3sXeQ4yo/edit?usp=sharing
- It is useful to do things in batch and automate repeatable tasks. Automation can save you time for other things.
- Borrow Borrow Borrow and Borrow again: If you have something similar to what you need try to adapt it to your problem
- Learning Code - not required, but can help you figure out problems or follow code
- Regular Expressions are AWESOME!!! (+1) Used in many programming environments for pattern matching.
- A Regular Expression is often reffered to as regex.
- Library searches introduce us to a small set of regular expressions such as the wildcard * char
- Regular Expressions have Literal characters and Meta characters.
  - Meta characters can be escaped for any literal meaning
- A good example is to use regex is to search for a pattern of phone numbers
- There are subtle differences between how each programming language interprets regex
- Regular expressions comparison chart: https://gist.github.com/CMCDragonkai/6c933f4a7d713ef712145c5eb94a1816
- Regular Expressions are extremely literal.
- Common regex metacharacters
  - [ABC] matches A or B or C.

        - [A-Z] matches any upper case letter.
        - [A-Za-z] matches any upper or lower case letter.
        - [A-Za-z0-9] matches any upper or lower case letter or any digit.
        - . matches any character.
        - \d matches any single digit.
        - \w matches any part of word character (equivalent to [A-Za-z0-9]).
        - \s matches any space, tab, or newline.
        - \ used to escape the following character when that character is a special character. So, for example, a regular expression that found .com would be \.com because . is a special character that matches any character.
        - ^ is an “anchor” which asserts the position at the start of the line. So what you put after the caret will only match if they are the first characters of a line. The caret is also known as a circumflex.
        - $ is an “anchor” which asserts the position at the end of the line. So what you put before it will only match if they are the last characters of a line.
        - \b asserts that the pattern must match at a word boundary. Putting this either side of a word stops the regular expression matching longer variants of words. So:
            - the regular expression `mark` will match not only mark but also find marking, market, unremarkable, and so on.
            - the regular expression `\bword` will match word, wordless, and wordlessly.
            - the regular expression `comb\b` will match comb and honeycomb but not combine.
            - the regular expression `\brespect\b` will match respect but not respectable ordisrespectful.
- Challenge - what do you see ^[Oo]rgani.e\b
    - ^ indicates the start of the line
    - [Oo] can match "O" or "o"
    - followed by rgani
    - followed by any character which is indicated by "."
    - followed by "e" and end boundary with "\b"
- Other useful special characters
    - * matches the preceding element zero or more times. For example, ab*c matches “ac”, “abc”, “abbbc”, etc.
    - + matches the preceding element one or more times. For example, ab+c matches “abc”, “abbbc” but not “ac”.
    - ? matches when the preceding character appears zero or one time.
    - {VALUE} matches the preceding character the number of times defined by VALUE; ranges, say, 1-6, can be specified with the syntax {VALUE,VALUE}, e.g. \d{1,9} will match any number between one and nine digits in length.
    - | means or.
    - /i renders an expression case-insensitive (equivalent to [A-Za-z]).
- Challenge ^[Oo]rgani.e\w*
    - "\w" means match any character in [A-Za-z0-9], no special characters
    - "*" matches zero or any number of characters
    - Possible matches
        - Organized
        - Organised
        - Organize123
- Challenge [Oo]rgani.e\w+$
    - "+" matches the preceding element one or more times

    - "$" indicates the end of the line
- Usually work with regular expressions not to get a regular expression and see what it finds, but rather want to find certain things and create regular excpressions
- Regular Expressions Cheat Sheet: https://librarycarpentry.org/lc-data-intro/reference
- Challenge: Fr[ea]nc[eh]
    - Possible solutions: French, France, Franch, Frence
- Challenge: Fr[ea]nc[eh]$
    - Possible solutions: French, France, Franch, Frence
    - $ means match end of the line
- Challenge: What would match the strings French and France that appear at the beginning of a line?
    - ^(French|France)
    - ^Fr[ae]nc[eh]
- Challenge: How do you match the whole words colour and color (case insensitive)?
    - \b[Cc][Oo][Ll][Oo][Uu]?[Rr]\b
    - ^/colo[u]?r/i$
    - ? matches u zero or one times
- \s and . can both include spaces
- Usually more than one way to use regular expressions to find certain patterns - "we can do anything we want" that works!
- Match means if a regex matches a word in a document it would highlight them
- Match digits 4 times: \d{4} or \d\d\d\d or. [0-9]{4} or [0-9][0-9][0-9][0-9]
- Regular expressions are for pattern matching. And you don't have to remeber everything.
- Regex101.com https://regex101.com/
    - This site helps you to easily test your regex
    - There is a cheat sheet at the bottom in the Quick Reference section
    - This will also breakdown your regex and give you a nice explanation in the Explanation section
    - Your matches are shown in the left column under Match Information section
        - There is an "Export Matches" button in this section that can help you export matches
    - "\S" matches anything other than a space, tab, or a new line
- Possible regex for emails: \S+@\S+\.\w{2,3}
- Possible regex for ph# without the area code: \d{3}[-|\.]\d{4}
- Possible regex for phone numbers with area codes: \(?\d{3}\)?[-\.\s]*\d{3}[-\.\s]*\d{4}
- Possible regex for phone mumbers that matches country code: (\+?1[-|\s]?)?\(?\d{3}\)?[-\.\s]*\d{3}[-\.\s]*\d{4}
- Regex using google sheets
    - Make a copy of the sheet to work to work on: https://docs.google.com/spreadsheets/d/11A7GZs5p9tsqlDYg4LLmRpROQdQDd1NluWq3sXeQ4yo/edit?usp=sharing
    - Multiple types of info in the address column (address , coordinates)
    - Match coordinates using \(-?\d{1,3}\.\d+,\s*-\d{1,3}\.\d+\)
    - =REGEXEXTRACT is the function to use in google sheets to extract data
        - e.g =REGEXEXTRACT(G2, "-?\d{1,3}\.\d+,\s?-?\d{1,3}\.\d+")
    - =REGEXREPLACE function can replace the regex with a string or with empty
        - e.g =REGEXREPLACE(G2, "\(-?\d{1,3}\.\d+,\s?-?\d{1,3}\.\d+\) ", "")
    - Use Find and Replace, search using regex and replace with nothing to replace data
        - be careful of sneaky extra spaces if copy/pasting

        - Good to remember
            - in a straight bracket [] the hyphen signifies a range [a-z], but in a curly bracket {} the comma signifies a range {1,3}

- Testing links:
    - Regex101.com https://regex101.com/
    - https://regexr.com/
    - Visualize regex: https://regexper.com/
    - https://regexcrossword.com/
- Regex for either 3 or 5 digit numbers \d{3}|\d{5}
- Marcedit
    - https://librarycarpentry.github.io/lc-marcedit
    - Use File->Open and find your file to open in MARC edit
    - Use tools->edit field data. There should be a checkbox for Use Regular Expressions
    - Write your regex and select Process to replace
    - The process button has a dropdown to preview results
    - The find and replace has a similar option to use regex to find and replace text
    - Use tools->regular expression store to save your regex

Questions

-
-

Day 3: Unix Shell

Lesson Material: https://librarycarpentry.github.io/lc-shell/
Data Download: https://librarycarpentry.github.io/lc-shell/data/shell-lesson.zip
Minute Card: https://forms.office.com/r/508RZPG2qJ

Attendance

Name and one word or phrase that stuck in your mind from yesterday

- Sruthin Gaddam: Use commas in curly braces e.g. {1,3}
- Julie Taylor: InDesign's GREP ***IS*** RegEx!!! :)
- Eli Wachter: Circumflex is another word for caret
- Colleen Lyon: regex101.com
- Mike Laney: regextract!
- Becky Holm: regular expression can be used in Google Sheets
- Sarah Mainville: regex crossword!
- Alex Hauser, regex is a thing (first time learning about it at all)
- Nick DeVolder: REGEXEXTRACT and REXEXREPLACE in Google sheets
- Christine Peffer: same as Sarah above—did not know about regex previously!

- Suzi Teghtmeyer RegExtract!

Notes

- Lesson: https://librarycarpentry.github.io/lc-shell/
- Data to download: https://librarycarpentry.github.io/lc-shell/data/shell-lesson.zip
    - Please move the downloaded shell-lesson.zip to your Desktop and unzip it.
- Operating system setup
    - Mac users: Applications > Utilties > Terminal (or Spotlight Search for Terminal)
    - Windows users: download & install Git Bash (https://gitforwindows.org/)
- The Unix Shell is useful for manipulating large batches of files (changing filenames, editing contents, etc)
- The Unix Shell is an example of a command line interface
    - This is how people interacted with the computer before graphical user interfaces where you can click around.
    - The $ is a prompt where you can start typing.
    - The ~ is a shorthand for your main "home" folder/directory
- The "**pwd**" command stands for "print working folder". It tells you the folder your shell is currently looking at
- Use the "**ls**" command to see the contents of your current folder
    - It may be color coded depending on your settings
- Commands can be modified with "flags"
    - Flags either start with one dash or two
    - For example, "**ls -l**" will list additional columns of information when showing folder contents
    - We can use "**ls -lh**" to see file sizes in human-friendly units (e.g. MB, KB, GB) instead of just in bytes. The "-h" flag only works when combined with the "-l" flag
- We can move to another directory with the "**cd**" command for "change desktop"
    - e.g. "cd Desktop" will move our terminal to our Desktop folder.
    - Nothing will be printed when the command executes successfully. You can always use "pwd" to check your location.
    - If you try to "cd" into a directory that doesn't exist, the command will give you an error saying "no such file or directory"
- A shortcut to return to the directory you were just in is "**cd -**"
- To go up one directory level, you can use the shortcut "**cd ..**"
    - If you're in your Desktop folder, "cd .." will take you to your main user directory
- The Bash Shell let's you press the Tab key to autocomplete a partially written file or folder name.
    - it will only complete if it can uniquely identify one name based on what you've typed so far
- WINDOWS ONLY: if you want to open the graphical file explorer for the directory your shell is currently in, type "explorer ."
    - Typing "explorer" without a dot will open your main home directory instead of the folder you're currently in
- A single period "." stands for the current folder (whereas two dots ".." is the parent)
- MAC ONLY: you can use the "**man**" command to get more information about a Bash command, including the arguments and flags, e.g. "man ls"
    - Use the space bar to go between pages and q to quit

- FOR EVERYONE: the man.he.net website has the same contents as the "man" command
- Typically, flags use one dash "-" for single letters and two dashes "--" for whole words
    - single-letter flags can be combined; e.g. "ls -l -h" can also be written "ls -lh" as we've seen
    - flags are CASE SENSITIVE!
- Challenge: Advanced "ls" Commands
    - Use the manual resources to find how to sort files by their size using "ls"
    - ANSWER: use the "-S" flag with "ls": "ls -S". Note the capitalization! (-S = sort by size)
    - Add the "-l" to confirm this ordering. Remember the "-h" flag will show the file sizes in human-friendly units
    - NOTE: using lowercase "-s" is similar to the "-l" flag but it *only* shows the file sizes (-s = show size)
- Use "cd" to access the "shell-lesson" folder you extracted earlier
    - cd Desktop/shell-lesson or cd shell-lesson depending on where you start from
        - Alternatively, you can use "cd ~/Desktop/shell-lesson" to reach the correct folder no matter where you are
    - Remember tab completion!
- Inside the "shell-lesson" directory, we're going to make a new directory using the "**mkdir**" command
    - mkdir firstdir
    - "firstdir" is the name of our new directory
- Now, let's look at the contents of "shell-lesson" with "ls -lh"
    - You'll see your new "firstdir" directory
    - There are several TSV files
    - There are two TXT files with numbers as their filenames. These files are books from Project Gutenberg and we'll be using those first
- The "**cat**" command stands for "concatenate" but it will print the entire contents of a file to your screen
    - **Be careful using this with large files!!**
    - You can use the "**clear**" command to clear your terminal if you did print out a large file.
- You can view just the first few lines of a file with the "head" command (the default is 10 lines)
    - head 829-0.txt
- Similarly, the "**tail**" command shows the last 10 lines of a file
    - tail 829-0.txt
- You can use the "**less**" command to see a file one page at a time
    - use space bar to scroll
    - use "q" to quit
- You can use the up and down arrow keys to scroll through your previous commands
- You can use head, tail, and cat to view mutliple files at once:
    - head 829-0.txt 33504-0.txt
- The wildcard character "*" (one or more letters of any kind) can be used to grab multiple files without typing them all out
    - head .txt
- The ? mark is a wildcard for one or zero characters
- We can rename files using the "**mv**" command which actually stands for "move"
    - mv 829-0.txt gulliver.txt
    - Type "ls" to confirm that the file name has been changed. You will no longer see 829-0.txt but instead see gulliver.txt
- We can make copies of files using the "**cp**" command
    - cp gulliver.txt gulliver-backup.txt

        - We can use "ls" to see that we have both gulliver.txt and backup.txt in our directory.
- Challenges: use the "mv" command to rename "firstdir" to "backup" and move our backup of gulliver.txt into that backup folder
    - Remember you can use the "man" command or man.he.net to read the instructions for the "mv" command
    - ANSWERS:
        - mv firstdir backup
        - mv gulliver-backup.txt backup/
- We can use the "**history**" command to see previous commands from our session
    - We can use ! with the number of a command to rerun it, for example "!124"
- The "**echo**" command will print out whatever you give it. Your message should use quotes.
    - echo "Library carpentry is fun!"
    - This is most useful when we make **variables**
        - NAME=Allie
        - echo "$NAME is teaching Library Carpentry"
        - variable names are case sensitive
- We can use the "**date**" command to give us the current date.
    - We can imbed this inside an echo print statement:
    - echo "It's really nice out on $(date)"
- We promised that the Bash Shell is good for automation. Let's learn how to write loops!
    - To do this, we're going to make a bunch of example empty files using "**touch**"
        - The touch command was original intended to update the last modified data of a file, but it can also be used to make an empty file
    - touch a.txt b.txt c.txt d.txt
    - You can use "ls" to confirm they are present
- The basic structure of a loop:
    - for thing in list_of_things
    - do
        - operation_using $thing
    - done
    - The indentation is not required by improves legibility, especially in scripts.
        - it can cause problems on the command line
- If you start typing out a loop on the command line, your prompt will change from $ to > on all lines after the first. The $ and > are included below for an example but you should **not** type them
    - $ for filename in ?.txt
    - > do
    - > echo "$filename"
    - > cp "$filename" backup_"$filename"
    - > done
    - Confirm your backup files were created using "ls"
    - Make sure you have double quotes and the dollar sign when using a variable or things may not work correctly; small typos can have undesired consequences
        - You can replace filename with whatever variable name you'd like, but meaningful names are best
- Exercise:
    - What will a loop do if we say "for filename in *.txt" instead of "?.txt"?
        - it will loop over *all* text files

        - If we make a loop using the "file" variable, what will the following do?
        - for file in .txt
        - do
            - echo "$file"
            - head -n 1 "$file"
            - tail -n 1 "$file"
        - done
            - Print the name of the file, the first line, and the last line.
            - the "-n" flag let's you change the number of lines to be printed by head and less. Recall the default is 10
            - Empty files won't print anything for "head" or "tail"
- You can put commands like a for-loop into their own file. These end with the ".sh" file extension for "shell". These files are called shell scripts and can be run inside the shell.
    - This is useful for commands you use frequently
- Question: can I rename more than just text files?
    - Yes! You can rename any kind of file using the techniques we've seen
- If we run "ls -lhS", we see our biggeset files are the TSV files
    - These are "tab separated value" files. They're similar to CSV (comma separated value) files but data is separated by tabs instead of commas
    - This is a non-proprietary file format (unlike, say, excel sheets)
- Again, we can look at the whole file using "cat" but it's probably better to use "head"
    - head -n 3 2014-01_JA.tsv
    - This will show us the header and some sample data entries
- We can use the "**wc**" command to see the number of lines, words, and bytes in a document
    - wc .tsv
    - You can get just one piece of information using flags
        - "-l" for lines
        - "-w" for words
- We can save the output of the "wc" command to its own file using a **redirect**
    - You can retype the previous command or use the up arrow key to bring it up
    - Add "> lengths.txt"
        - wc -l .tsv > lengths.txt
    - If we run "cat lengths.txt" we'll see the output of "wc" has been saved!
- What if we want the output of "wc" to be sorted by length?
    - We can use "**sort -n**" to sort by number (instead of alphabetically)
    - sort -n lengths.txt > sorted-lengths.txt
    - Use "cat" to see the documents sorted by number of lines
        - You could use "head -n 1" and "tail -n 1" to easily see which are the shortest!
- This whole process is more convienient if we don't have to make a bunch of intermediate files
    - This can be done using a **pipe** "|" (shift+backslash key)
    - A pipe makes the output of the command on the left the input to the command on the right
    - wc -l .tsv | sort -n | head -n 1
        - This will tell you which file has the fewest lines
- Challenges
    - What happpens if you pipe "wc -l *.tsv | sort -n | head -n 1" into "cat"?
        - wc -l .tsv | sort -n | head -n 1 | cat
        - This doesn't change much since "head" already prints to the command line, but it could be useful with a command that doesn't already print

        - Adjust "wc -l *.tsv | sort -n | head -n 1" to instead count the number of words, order them, and then print the 10 files with the most words
            - wc -w .tsv | sort -n | tail -n 11
            - Need 11 lines because the final line is a total!
- We can search files using "**grep**". This stands for "global regular expression print", so we can search files using regular expressions!
    - To search for the year 1999:
        - grep 1999 .tsv
- If we want a count of how many times a string appears, we can use the "-c" flag
    - grep -c 1999 .tsv
    - grep -c revolution .tsv
    - Question: what if you want all the endings of "revolution?" revolutionary, revolutions etc
        - By default, grep will match those (i.e. not " exact match" search)
        - There is a flag to change this behavior and match "revolution" exactly: the "-w" flag
            - This can still be used with "-i" to make our exact match case insensitive
- The "-i" flag can also be used to make search case insensitive
    - grep -ci revoultion .tsv
    - Notice how the count goes up!
- We can redirect the output of our searches into a date-stamped file. We specifically want to use year-month-day format
    - mkdir results
    - grep -i revolution .tsv > results/$(date "+%Y-%m-%d")_JAi-revolution.tsv
        - the "+%Y-%m-%d" argument we give the "date" command specifies the format we want to use.
        - We use "i-revolution" in the filename to give information about how our search was performed
    - use "ls results/" to see the new file
- Question: what's the difference between Bash & the Unix Shell?
    - they're very similar and are often used interchangeably, but technicall the shell is the thing we're typing in and bash is the language
- We can use grep with regular expressions (yesterday's lesson!)
    - The flags that you use are different on Mac and Windows
        - WINDOWS: grep -P
        - MAC: grep -E
    - **These notes will use the Windows version**
    - grep -iwP 'fr[ae]nc[eh]' .tsv
        - note the single quotes around our regular expression!
        - This will also make franch and frence which aren't real words so it should be ok
    - Remember we can add the "-c" flag to count the instances of both French and France
- Let's use grep to process Little Women. Specifically, how many times does each of the main character's names appear?
    - First, let's rename the file
        - mv pg514.txt littlewomen.txt
    - Then, let's write a loop over an array of the names. Pay attention to the quotes!
        - for name in "Jo" "Meg" "Beth" "Amy"
        - do
            - echo "$name"
            - grep -wo "$name" littlewomen.txt | wc -l

            - done
            - The "-o" flag will output each match on it's own line
            - Note that the "-w" flag means we won't pick up instances of "Jo's" (possessive) but also won't pick up words like "Joseph"
- If you press the up arrow key, you'll see the for loop we just typed but with semicolons instead of new lines. They perform the same function.
- Do we need all those double quotes?
    - They protect us in case any of the strings we're searching have spaces in them.
    - For instance, "grep -wo Louisa May Alcott" will cause a bunch of errors because it thinks May and Alcott are files we want to search for the word "Louisa"
- Breakout Excerises: https://librarycarpentry.github.io/lc-shell/06-free-text.html
    - Option 1 is the only one with an optional challenge at the end (you don't have to do it)
    - Option 2 is shorter; recommend 1 or 3 unless you work with JSON files frequently
- These exercises (re)introduce the "**less**", "**sed**", "**tr**", and "**uniq**" commands for scrolling through files, deleting lines, translating (changing) or deleting characters, and removing duplicates
    - The ultimate goal was to find how many instances of each unique word
    - To do this, we had to clean the file first (remove punctuation, make it all lowercase)
    - Each cleanup step could be joined together with pipes to avoid making multiple intermediate files
        - The more pipes you use, the more aware you need to be of what each individual step is outputting

Questions

Day 4: Git

Lesson Material: https://librarycarpentry.github.io/lc-git/
Data Download:
Post-workshop survey: https://carpentries.typeform.com/to/UgVdRQ?slug=2025-11-10-msu-online

Questions

Review Exercise (small groups)

https://librarycarpentry.github.io/lc-git/instructor/04-review.html

- How to use the Whiteboard in Zoom: https://answers.uillinois.edu/ischool/129751

Attendance

Name and one word or phrase that stuck in your mind from yesterday

- Eli Wachter: man.he.net
- Mike Laney, tr [:upper:] [:lower:]

- Colleen Lyon: using the command line is fun
- Sruthin Gaddam: the "tr" cmd is AWESOME!!!
- Leila Sherbini, pipes!
- Christine Peffer: grep!
- Nick DeVolder: cleaning data to analyze text
- Becky Holm, pipes and Microsoft does not play nice with Bash/Unix
- Suzi Teghtmeyer looping

Notes

- VC=Source Control, Version Control
- Version Control helps manage changes between versions
- get git version using "git --version"
- Can be used for variety of projects, both personal and professional, when working with peers, friends.
    - Schedule sharing for an international trip.
    - working with IT colleagues to create logic for a script to act on MARC and item records in our LSP
    - building collections of files/items in a drupal group
- GitHub has a remote copy that lives somewhere in the clould but you have reference to it on your computer
- You can promote or discover other similar projects on GitHub
- Working on Getting Started with Git: https://librarycarpentry.github.io/lc-git/02-getting-started.html
- Use "pwd" to see the current directory
- Use "cd" to change directory. e.g. cd /path/to/dir
- Use git --version to see if git is installed and see the current version
- Use git config --list to see the configs for your setup
- Use git config --global user.name "Your Name" to set your user name
- Use git config --global user.email "Your GitHub email" to set your email. This should be the same email that you used to signup with github
    - Use https://github.com/settings/emails to check which email you used to signup
- Use git config --global init.defaultBranch main to set the default init branch to "main"
- use "mkdir" to make new directories. e.g. mkdir hello-world
- Use "git init" to initialize an empty git repository in a directory
- Use "ls -la" to see all directories. This shows all hidden directories.
- "git status " will show you changed files in the repository
- Helpful reference: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
- Use "git add" cmd to include files that need to be committed
    - Use "git rm --cached <file>" to remove any files that have been added
- Use "git commit -m "commit message" " to commit your changes.
    - The changes are just commited not pushed to your GitHub repo yet!
- Login into GitHub. Use the "+" icon on the top right, when clicked an option to create a remote repository is shown
    - Enter the repo name and use the "Create Repository" button on the bottom to create a new repository
- Use "git remote add origin {github:ssh-url}" to link remote repository to your local repository
    - List of flags for the git remote command: https://git-scm.com/docs/git-remote

- Use ssh-keygen -t ed25519 -C "yourname@domain.name" to generate a public & private key
    - Public key: this can be shared with external parties like GitHub. This is basically a lock
    - Private key: it's a key to your lock. Do not share
- Add your shh public key at https://github.com/settings/keys
    - Select "New SSH Key", enter a descriptive title like name@worklaptop and copy your public key in the Key textbox
- "git push -u origin main" command uploads your current repository to GitHub
    - "-u" stands for upstream
    - "main" is the name of the branch
- "git pull" pulls the changes from the remote repository to your local repo
- "git log" command gives you the log of all the changes and the commit messages
- Git analogies: https://medium.com/@tech.media.unicorn/git-github-for-total-beginners-an-analogies-first-guide-44b014632571
- Github Pages - A space on GitHub to publish your own websites
    - GitHub pages will be available at username.github.io/repository-name/

Resources:

References/Tutorials:

- https://librarycarpentry.github.io/lc-git/reference.html
- Git Cheat Sheet: https://training.github.com/downloads/github-git-cheat-sheet.pdf
- Markdown Formatting: https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
- Git Tutorials: https://skills.github.com/
    - https://github.com/skills/introduction-to-github
    - https://github.com/skills/github-pages

Examples of GitHub Pages in Action:

- Programming Historian: https://programminghistorian.org/ on GitHub: https://github.com/programminghistorian/jekyll
- Integrated Music Theory Textbook: https://intmus.github.io/inttheory/ on GitHub: https://github.com/intmus/inttheory
- Collection Builder: https://collectionbuilder.github.io/collectionbuilder-gh/ on GitHub: https://github.com/CollectionBuilder/collectionbuilder-gh/
- Library Carpentry Lessons!: https://librarycarpentry.github.io/lc-marcedit/
- Library Carpentry Workshop Websites: https://abigailsparling.github.io/2025-11-10-msu-online/ / GitHub template: https://github.com/carpentries/workshop-template