About this Site

Contents

Syllabus

Notes

Activities

- KWL Chart
- Prepare for the next class
- More Practice
- Deeper Explorations
- Project Information

FAQ

- Syllabus and Grading FAQ
- Git and GitHub

Resources

- Glossary
- Language Specific References
- Cheatsheet
- General Tips and Resources
- How to Study in this class
- Getting Help with Programming
- Getting Organized for class
- Advice from Dr. Brown's Data Science Students

Welcome to the course manual for Introduction to Computer Systems in Spring 2022 with Professor Brown.

This class meets TuTH 12:30-1:45 in Engineering Building Room 040.

This website will contain the syllabus, class notes, and other reference material for the class.

Course Calendar on BrightSpace

> **Tip**
>
> subscribe to that calendar in your favorite calendar application

Navigating the Sections

The Syllabus section has logistical operations for the course broken down into sections. You can also read straight through by starting in the first one and navigating to the next section using the arrow navigation at the end of the page.

This site is a resource for the course. We do not follow a text book for this course, but all notes from class are posted in the notes section, accessible on the left hand side menu, visible on large screens and in the menu on mobile.

The resources section has links and short posts that provide more context and explanation. Content in this section is for the most part not strictly the material that you'll be graded on, but it is often material that will help you understand and grow as a programmer and data scientist.

Reading each page

All class notes can be downloaded in multiple formats, including as a notebook. Some pages of the syllabus and resources are also notebooks, if you want to see behind the curtain of how I manage the course information.

> **Try it Yourself**
>
> Notes will have exercises marked like this

Question from Class

Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Long answers will be in the main notes

Further reading

Notes that are mostly links to background and context will be highlighted like this. These are optional, but will mostly help you understand code excerpts they relate to.

Question from class

Questions that are asked in class, but unanswered at that time will be answered in the notes and marked with a box like this. Short questions will be in the margin note

Hint

Both notes and assignment pages will have hints from time to time. Pay attention to these on the notes, they'll typically relate to things that will appear in the assignment.

Think Ahead

Think ahead boxes will guide you to start thinking about what can go into your portfolio to build on the material at hand.

Syllabus

Welcome to CSC302: Introduction to Computer Systems.

In this syllabus you will find an overview of the course, information about your instructor, course policies, restatements of URI policies, reminders of relevant resources, and a schedule for the course.

This is a live document that will change over time, but a pdf copy is available for direct download or to view on GitHub. Note that this will become outdated over time.

Basic Facts

Introduction to Computer Systems

This new course links together different ideas that you have encountered but not covered deeply in other courses. We'll learn about tools used in programming and how they work. The goal of this course is to help you understand how your computer and programming environment work so that you can debug and learn independently more confident.

Quick Facts

- **Course time:** Spring 2022, TuTh 12:30PM - 1:45PM
- **Credits:** 4

To request a permission number complete this google form **you must be signed into your URI google account to access the form**

Why Take this course

1. use and understand git/ GitHub
2. make sense of cryptic compiler messages
3. understand how file organization impacts programming
4. fulfill your 300 level CSC elective requirement
5. preview ideas that will be explored in depth in 411 & 412

Topics covered

this is a partial list

- git and other version control
- bash and other shell scripting
- filesystems
- basics of hardware
- what happens when you compile code
- what are the different types of software on your computer

Catalog Description

How the history and context of computing impacts the practice of computing today. Tools used in programming and computational problem solving. How programming works from high level languages to hardware. Survey of computer hardware and representation of information. Pre: CSC110, any 200 level CSC course, or equivalent.

Learning Outcomes

By the end of the semester, students will be able to:

1. Differentiate the different classes of tools used in computer science in terms of their features, roles, and how they interact and justify positions and preferences among popular tools
2. Identify the computational pipeline from hardware to high level programming language
3. Discuss implications of choices across levels of abstraction
4. Describe the context under which essential components of computing systems were developed and explain the impact of that context on the systems.

About this syllabus

You can get notification of changes from GitHub by “watching” the repository You can view the date of changes and exactly what changes were made on the Github commit history page.

Creating an issue is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section. That will be linked when sovle and you will get a notification at that time.

About your instructor

Name: Dr. Sarah M Brown Office hours: TBA via zoom, link on BrightSpace

Dr. Sarah M Brown is a second year Assistant Professor of Computer Science, who does research on how social context changes machine learning. Dr. Brown earned a PhD in Electrical Engineering from Northeastern University, completed a postdoctoral fellowship at University of California Berkeley, and worked as a postdoctoral research associate at Brown University before joining URI. At Brown University, Dr. Brown taught the Data and Society course for the Master’s in Data Science Program. You can learn more about me at my website or my research on my lab site.

You can call me Professor Brown or Dr. Brown, I use she/her pronouns.

The best way to contact me is e-mail or an issue on an assignment repo. For more details, see the Communication Section

Tools and Resources

We will use a variety of tools to conduct class and to facilitate your programming. You will need a computer with Linux, MacOS, or Windows. It is unlikely that a tablet will be able to do all of the things required in this course. A Chromebook may work, especially with developer tools turned on. Ask Dr. Brown if you need help getting access to an adequate computer.

All of the tools and resources below are either:

- paid for by URI **OR**
- freely available online.

BrightSpace

> **Note**
>
> Seeing the BrightSpace site requires logging in with your URI SSO and being enrolled in the course

This will be the central location from which you can access all other materials. Any links that are for private discussion among those enrolled in the course will be available only from our course Brightspace site .

This is also where your grades will appear and how I will post announcements.

For announcements, you can customize how you receive them.

Prismia chat

Our class link for Prismia chat is available on Brightspace. Once you've joined once, you can use the link above or type the url: prismia.chat. We will use this for chatting and in-class understanding checks.

On Prismia, all students see the instructor's messages, but only the Instructor and TA see student responses.

Course Manual

The course manual will have content including the class policies, scheduling, class notes, assignment information, and additional resources.

Links to the course reference text and code documentation will also be included here in the assignments and class notes.

GitHub

You will need a GitHub Account. If you do not already have one, please create one by the first day of class. If you have one, but have not used it recently, you may need to update your password and login credentials as the Authentication rules changed in Summer 2021. In order to use the command line with https, you will need to create a Personal Access Token for each device you use. In order to use the command line with SSH, set up your public key.

Programming Environment

In this course, we will use several programming environments. In order to complete assignments you need the items listed in the requirements list. The easiest way to meet these requirements is to follow the recommendations below. I will provide instruction assuming that you have followed the recommendations. We will add tools throughout the semester, but the following will be enough to get started.

> **Warning**
>
> This is not technically a *programming* class, so you will not need to know how to write code from scratch in specific languages, but we will rely on programming environments to apply concepts.

Requirements:

- Python with scientific computing packages (numpy, scipy, jupyter, pandas, seaborn, sklearn)
- Git
- A bash shell
- A web browser compatible with Jupyter Notebooks
- nano text editor

> **Note**
>
> all Git instructions will be given as instructions for the command line interface and GitHub specific instructions via the web interface. You may choose to use GitHub desktop or built in IDE tools, but the instructional team may not be able to help.

> **Warning**
>
> Everything in this class will be tested with the up to date (or otherwise specified) version of Jupyter Notebooks. Google Colab is similar, but not the same, and some things may not work there. It is an okay backup, but should not be your primary work environment.

Recommendation:

- Install python via Anaconda
- if you use Windows, install Git and Bash with GitBash (video instructions).
- if you use MacOS, install Git with the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this by trying to run git from the Terminal the very first time.`git --version`
- if you use Chrome OS, follow these instructions:

1. Find Linux (Beta) in your settings and turn that on.
2. Once the download finishes a Linux terminal will open, then enter the commands: `sudo apt-get update` and `sudo apt-get upgrade`. These commands will ensure you are up to date.
3. Install tmux with:

```
sudo apt -t stretch-backports install tmux
```

4. Next you will install nodejs, to do this, use the following commands:

```
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash
sudo apt-get install -y nodejs
sudo apt-get install -y build-essential.
```

5. Next install Anaconda's Python from the website provided by the instructor and use the top download link under the Linux options.
6. You will then see a .sh file in your downloads, move this into your Linux files.
7. Make sure you are in your home directory (something like home/YOURUSERNAME), do this by using the `pwd` command.
8. Use the `bash` command followed by the file name of the installer you just downloaded to start the installation.
9. Next you will add Anaconda to your Linux PATH, do this by using the `vim .bashrc` command to enter the .bashrc file, then add the `export PATH=/home/YOURUSERNAME/anaconda3/bin/:$PATH` line. This can be placed at the end of the file.
10. Once that is inserted you may close and save the file, to do this hold escape and type `:x`, then press enter. After doing that you will be returned to the terminal where you will then type the source .bashrc command.
11. Next, use the `jupyter notebook –generate-config` command to generate a Jupyter Notebook.
12. Then just type `jupyter lab` and a Jupyter Notebook should open up.

Video install instructions for Anaconda:

- Windows

- Mac

On Mac, to install python via environment, this article may be helpful

- I don't have a video for linux, but it's a little more straight forward.

Zoom (backup only & office hours only, Spring 2022 is in person)

This is where we will meet if for any reason we cannot be in person. You will find the link to class zoom sessions on Brightspace.

URI provides all faculty, staff, and students with a paid Zoom account. It *can* run in your browser or on a mobile device, but you will be able to participate in class best if you download the Zoom client on your computer. Please log in and configure your account. Please add a photo of yourself to your account so that we can still see your likeness in some form when your camera is off. You may also wish to use a virtual background and you are welcome to do so.

For help, you can access the instructions provided by IT.

Schedule

Overview

The following is a rough outline of topics in an order, these things will be filled into the concrete schedule above as we go. These are, in most cases bigger quetions than we can tackle in one class, but will give the general idea of how the class will go.

This plan accounts for 1 less week than we actually have. We will either go over somewhere or we'll use the last week for sharing projects, reflection, or an additional topic that comes up during the semester.

How does this class work?

one week

We'll spend the first two classes introducing some basics of GitHub and setting expectations for how the course will work. This will include how you are expected to learn in this class which requires a bit about how knowledge production in computer science works and a bit of the history.

How do all of these topics relate?

approximatley two weeks

> **Tip**
>
> We will integrate history throughout the whole course. Connecting ideas to one another, and especially in a sort of narrative form can help improve retention of ideas. My goal is for you to learn.
> We'll also come back to different topics over and over again with a slightly different framing each time. This will both connect ideas, give you chance to practice recalling (more recall practice improves long term rentention of things you learn), and give you a chance to learn things in different ways.

We'll spend a few classes doing an overview where we go through each topic in a litte more depth than an introduction, but not as deep as the rest of the semester. In this section, we will focus on how the different things we will see later all relate to one another more than a deep understanding of each one. At the end of this unit, we'll work on your grading contracts.

We'll also learn more key points in history of computing to help tie concepts together in a narrative.

Topics:

- bash
- man pages (built in help)
- terminal text editor
- git
- survey of hardware
- compilation
- information vs data

What tools do Computer Scientists use?

approximately four weeks

Next we'll focus in on tools we use as computer scientists to do our work. We will use this as a way to motivate how different aspects of a computer work in greater detail.

Topics:

- linux
- git
- i/o
- ssh and ssh keys
- number systems
- file systems

What Happens When I run code?

approximately five weeks

Finally, we'll go in really deep on the compilation and running of code. In this part, we will work from the compilation through to assembly down to hardware and then into machine representation of data.

Topics:

- software system and Abstraction
- programming languages
- cache and memory
- compiliation
- linking
- basic hardware components

Finalized Order

Content from above will be expanded and slotted into specific classes as we go. This will always be a place you can get reminders of what you need to do next and/or what you missed if you miss a class as an overivew. More Details will be in other parts of the site, linked to here.

Date	Key Question	Prepation	Activities
2021-01-25	What are we doing this semester?	Create GitHub and Prismia accounts, take stock of dev environments	introductions, tool practice
2021-01-27	How does knowledge work in computing?	Read through the class site, notes, reflect on a thing you know well	course FAQ, knowledge discussion
2021-02-01	How do I use git offline?	review notes, reflect on issues, check environment, map cs knowledge	cloning, pushing, terminal basics
2021-02-03	Why do I need to use a terminal?	review notes, practice git offline 2 ways, update kwl	bash, organizing a project
2021-02-08	What are the software parts of a computer system?	practice bash, contribute to the course site, examine a software project	hardware simulator
2021-02-10	What are the hardware parts of a computer system?	practice, install h/w sim, review memory	hardware simulation
2021-02-15	How does git really work?	practice, begin contract, understand git	grading contract Q&A, git diff, hash
2021-02-17	What happens under the hood of git?	things	git plumbing and more bash (pipes and find)
2021-02-22	Why are git commit numbers so long?	review, map git	more git, number systems
2021-02-24	How can git help me when I need it?	reveiw numbers and hypothesize what git could help with	git merges
2021-03-01	How do programmers build documentation?	review git recovery, practice with rebase, merge, revert, etc; confirm jupyterbook is installed	templating, jupyterbook
2021-03-03	How do programmers auotmate mundane tasks?	convert your kwlrepo	shell scripting, pipes, more redirects, grep
2021-03-08	How do I work remotely ?	install reqs, reflect on grade, practice scrip	ssh/ ssh keys, sed/ awk, file permissions
2021-03-10	How do programmers keep track of all these tools?	summarize IDE reflections	IDE anatomy
2021-03-22	Skipped		
2021-03-24	How do Developers keep track of all these tools?	[compare languages you know]	
2021-03-29	How do we choose among different programming languages?	[install c compiler]	
2021-03-31	What happens when I compile code?		
2021-04-05	Why is the object file unreadable?	[what are operators]	bits, bytes, and integers/character representtion
2021-04-07	What about non integer numbers?		floating point representation
2021-04-12	Where do those bitwise operations come from?	[review simulator]	gates, registers, more integer
2021-04-14	What actually is a gate?		physics, history
2021-04-19	How do components work together?		memmory, IO, bus, clocks,
2021-04-21			
2021-04-26			
2021-04-28			

Table 1 Schedule

Grading

This section of the syllabus describes the principles and mechanics of the grading for the course.

Learning Outcomes

The goal is for you to learn and the grading is designed to as close as possible actually align to how much you have learned. So, the first thing to keep in mind, always is the course learning outcomes:

By the end of the semester, students will be able to:

1. Differentiate the different classes of tools used in computer science in terms of their features, roles, and how they interact and justify positions and preferences among popular tools
2. Identify the computational pipeline from hardware to high level programming language
3. Discuss implications of choices across levels of abstraction
4. Describe the context under which essential components of computing systems were developed and explain the impact of that context on the systems.

These are what I will be looking for evidence of to say that you met those or not.

Principles of Grading

Learning happens through practice and feedback. My goal as a teacher is for you to learn. The grading in this course is based on your learning of the material, rather than your completion of the activities that are assigned.

This course is designed to encourage you to work steadily at learning the material and demonstrating your new knowledge. There are no single points of failure, where you lose points that cannot be recovered. Also, you cannot cram anything one time and then forget it. The material will build and you have to demonstrate that you retained things.

- Earning a C in this class means you have a general understanding; you will know what all the terms mean and could follow along if in a meeting where others were discussing systems concepts.
- Earning a B means that you could apply the course concepts in other programming environments; you can solve basic common errors without looking much up.
- Earning an A means that you can use knowledge from this course to debug tricky scenarios and/or design aspects of systems; you can solve uncommon error while only looking up specific syntax, but you have an idea of where to start.

No Grade Zone

At the beginning of the course we will have a grade free zone where you practice with both course concepts and the tooling and assingment types to get used to expectations. You will get feedback on lots of work and begin your Know, Want to know, Learned (KWL) Chart in this period.

Grading Contract

In about the third week you will complete, from a provided template, a grading contract. In that you will state what grade you want to earn in the class and what work you are going to do to show that. If you complete all of that work to a satisfactory level, you will get that grade. The grade free zone is a chance for you to get used to the type of feedback in the course and the grading contract template will have example specifications to meet.

The finalized grading contract will include the specification that each piece of work has to adhere to.

All contracts will include maintaining a KWL Chart for the duration of the semester and consistent responses in class.

Notes

- Keep your deeper explorations and more practice task content in your KWL chart repository.
- Link approved PRs to your grading contract for record keeping.

Grading Contract Instructions

> **Important**
>
> this includes minor corrections relative to the readme in the template provided

Grading Policies

Late Work

You will get feedback on items at the next feedback period.

Regrading

Re-request a review on your Feedback Pull request.

For general questions, post on the conversation tab of your Feedback PR with your request.

For specific questions, reply to a specifc comment.

If you think we missed *where* you did something, add a comment on that line (on the code tab of the PR, click the plus (+) next to the line) and then post on the conversation tab with an overview of what you're requesting and tag @brownsarahm

Support

Academic Enhancement Center

Academic Enhancement Center (for undergraduate courses): Located in Roosevelt Hall, the AEC offers free face-to-face and web-based services to undergraduate students seeking academic support. Peer tutoring is available for STEM-related courses by appointment online and in-person. The Writing Center offers peer tutoring focused on supporting undergraduate writers at any stage of a writing assignment. The UCS160 course and academic skills consultations offer students strategies and activities aimed at improving their studying and test-taking skills. Complete details about each of these programs, up-to-date schedules, contact information and self-service study resources are all available on the AEC website.

- **STEM Tutoring** helps students navigate 100 and 200 level math, chemistry, physics, biology, and other select STEM courses. The STEM Tutoring program offers free online and limited in-person peer-tutoring this fall. Undergraduates in introductory STEM courses have a variety of small group times to choose from and can select occasional or weekly appointments. Appointments and locations will be visible in the TutorTrac system on September 14th, 2020. The TutorTrac application is available through URI Microsoft 365 single sign-on and by visiting aec.uri.edu. More detailed information and instructions can be found on the AEC tutoring page.
- **Academic Skills Development** resources helps students plan work, manage time, and study more effectively. In Fall 2020, all Academic Skills and Strategies programming are offered both online and in-person. UCS160: Success in Higher Education is a one-credit course on developing a more effective approach to studying. Academic Consultations are 30-minute, 1 to 1 appointments that students can schedule on Starfish with Dr. David Hayes to address individual academic issues. Study Your Way to Success is a self-guided web portal connecting students to tips and strategies on studying and time management related topics. For more information on these programs, visit the Academic Skills Page or contact Dr. Hayes directly at davidhayes@uri.edu.
- The **Undergraduate Writing Center** provides free writing support to students in any class, at any stage of the writing process: from understanding an assignment and brainstorming ideas, to developing, organizing, and revising a draft. Fall 2020 services are offered through two online options: 1) real-time synchronous appointments with a peer consultant (25- and 50-minute slots, available Sunday - Friday), and 2) written asynchronous consultations with a 24-hour turn-around response time (available Monday - Friday). Synchronous appointments are video-based, with audio, chat, document-sharing, and live captioning capabilities, to meet a range of accessibility needs. View the synchronous and asynchronous schedules and book online, visit uri.mywconline.com.

General URI Policies

Anti-Bias Statement:

We respect the rights and dignity of each individual and group. We reject prejudice and intolerance, and we work to understand differences. We believe that equity and inclusion are critical components for campus community members to thrive. If you are a target or a witness of a bias incident, you are encouraged to submit a report to the URI Bias Response Team at www.uri.edu/brt. There you will also find people and resources to help.

Disability Services for Students Statement:

Your access in this course is important. Please send me your Disability Services for Students (DSS) accommodation letter early in the semester so that we have adequate time to discuss and arrange your approved academic accommodations. If you have not yet established services through DSS, please contact them to engage in a confidential conversation about the process for requesting reasonable accommodations in the classroom. DSS can be reached by calling: 401-874-2098, visiting: web.uri.edu/disability, or emailing: dss@etal.uri.edu. We are available to meet with students enrolled in Kingston as well as Providence courses.

Academic Honesty

Students are expected to be honest in all academic work. A student's name on any written work, quiz or exam shall be regarded as assurance that the work is the result of the student's own independent thought and study. Work should be stated in the student's own words, properly attributed to its source. Students have an obligation to know how to quote, paraphrase, summarize, cite and reference the work of others with integrity. The following are examples of academic dishonesty.

- Using material, directly or paraphrasing, from published sources (print or electronic) without appropriate citation
- Claiming disproportionate credit for work not done independently
- Unauthorized possession or access to exams
- Unauthorized communication during exams
- Unauthorized use of another's work or preparing work for another student
- Taking an exam for another student
- Altering or attempting to alter grades
- The use of notes or electronic devices to gain an unauthorized advantage during exams
- Fabricating or falsifying facts, data or references
- Facilitating or aiding another's academic dishonesty
- Submitting the same paper for more than one course without prior approval from the instructors

URI COVID-19 Statement

The University is committed to delivering its educational mission while protecting the health and safety of our community. While the university has worked to create a healthy learning environment for all, it is up to all of us to ensure our campus stays that way.

As members of the URI community, students are required to comply with standards of conduct and take precautions to keep themselves and others safe. Visit web.uri.edu/coronavirus/ for the latest information about the URI COVID-19 response.

- Universal indoor masking is required by all community members, on all campuses, regardless of vaccination status. If the universal mask mandate is discontinued during the semester, students who have an approved exemption and are not fully vaccinated will need to continue to wear a mask indoors and maintain physical distance.
- Students who are experiencing symptoms of illness should not come to class. Please stay in your home/room and notify URI Health Services via phone at 401-874-2246.
- If you are already on campus and start to feel ill, go home/back to your room and self-isolate. Notify URI Health Services via phone immediately at 401-874-2246.

If you are unable to attend class, please notify me at brownsarahm@uri.edu. We will work together to ensure that course instruction and work is completed for the semester.

Office Hours & Comms

Help Hours

TBA

```
/tmp/ipykernel_2242/2146052215.py:1: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis='index')`
  help_df.style.hide_index()
```

Day	Time	Location	Host
Tuesday	4-5pm	online	Dr. Brown
Wednesday	1-2pm	online	Dr. Brown
Friday	11am-1pm	online	Mark

Online office hours locations are linked in the #help channel on slack

Tips

For assignment help

- **send in advance, leave time for a response** I check e-mail/github a small number of times per day, during work hours, almost exclusively. You might see me post to this site, post to BrightSpace, or comment on your assignments outside of my normal working hours, but I will not reliably see emails that arrive during those hours. This means that it is important to start assignments early.

Using issues

- use issues for content directly related to assignments. If you push your code to the repository and then open an issue, I can see your code and your question at the same time and download it to run it if I need to debug it
- use issues for questions about this syllabus or class notes. At the top right there's a GitHub logo that allows you to open a issue (for a question) or suggest an edit (eg if you think there's a typo or you find an additional helpful resource related to something)

For E-mail

- use e-mail for general inquiries or notifications
- Please include `[CSC392]` in the subject line of your email along with the topic of your message. This is important, because your messages are important, but I also get a lot of e-mail. Consider these a cheat code to my inbox: I have setup a filter that will flag your e-mail if you include that in subject to ensure that I see it.

1. Introduction

1.1. What is a System?

> **Tip**
>
> You can contribute or fix things on this page (and anywhere else in this site) by clicking on "suggest an edit" under the GitHub menu in the top right.

1.2. What are we going to learn? and Editing on GitHub

We initialized your KWL Chart. You will keep this chart up to date over the course of the semester. Mostly it will be prompted when you should fill it in, but you can add to it whenever you would like.

> **Further Reading**
>
> GitHub itself provides pretty good documentation, full of screenshots for things in browser. editing a file pull request

1.3. For next class

> **Note**
>
> This section will contain more detail, but a short list of what you need will always be in the schedule

- More practice with GitHub terminology. Accept this assignment, read through it, and follow the instructions at the end.
- Review these notes, bring any questions you have to class
- Read the syllabus, explore this whole website. Bring questions about the course. Be prepared for a scavenger hunt that asks you not to recall every fact about the course, but to know where to find informatio.
- Think about one thing you've learned really well (computing or not) and how do you know that you know it? (bring your example)

1.4. Questions After Class

1.4.1. What physical code will we be writing this semester?

Shell scripts, tiny bits of C, small examples of python. Mostly, we'll be focused on the glue that connects the pieces of a regular code project more than developing a lot of code. We will not, for example, study any algorithms.

1.4.2. How would committing, pull requests, branches, etc work if you wanted to work on something on your computer?

We will learn how to work with git offline next week. You can use git (and GitHub) via a terminal, specialized desktop applications, or built into many IDEs.

1.4.3. Why does github seem so simple on a surface level? but to actually use it requires much deeper knowledge…

GitHub provides a user friendly wrapper around git, which is an open source version control tool that is designed to be very powerful for and primarily used by, experienced programmers with a good knowledge of files systems. Git is in fact a special file system. GitHub makes the most common actions easier so that more people can do them, but the underlying tool can do so many things, so it can be complicated.

1.4.4. Will I be able to view my answers on prismia chat for the future? Will it erase?

Yes, the prismia history will always be available. You can also download the transcript from each class.

1.4.5. What about the feedback page? How to merge it or not merge it at all?

We will specifically advise you on when you should merge the Feedback PR.

2. How does learning and knowledge work in computing?

2.1. Git review

- Make sure you get GitHub terminology down and use this "assignment" to practice.

2.2. Scavenger Hunt

> **Note**
>
> The goal here is to make sure you know where to find basic things, not that you have memorized every bit of information about the course

- Where can you find when office hours are?
- Where can you find the detailed list of what to prepare for today's class?
- Where is the regrading policy?
- There's a term you don't recognize in an activity, where should you look?
- Something went wrong in an assignment repo on GitHub, what should you check before asking for help?

2.3. Recall, Systems

"Systems" in computing often refers to all the parts that help make the "more exciting" algorithmic parts work. Systems is like the magic that helps you get things done in practice, so that you can shift your attention elsewhere.

In intro courses, we typically give you an environment to hide all the problems that could occur at the systems level.

Systems programming is how to look at the file system, the operating system, etc.

2.4. Mental Models and Learning

2.4.1. What is it like to know something really well?

When we know something well, it is easier to do, we can do it multple ways, it is easy to explain to others and we can explain it multiple ways. we can do the task almost automatically and combine and create things in new ways. This is true for all sorts of things.

a mental model is how you think about a concept and your way of relating it. Novices have sparse mental models, experts have connected mental models.

When we first learn new things, we first get the basic concepts down, but we may not know how they relate.

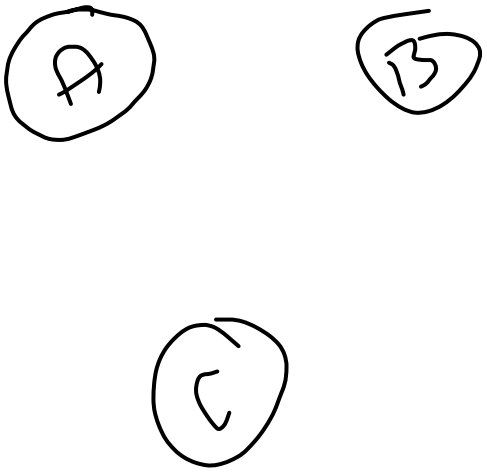


Fig. 2.1 a novice mental model is disconnected and has few concepts

As we learn more, they become more connected.

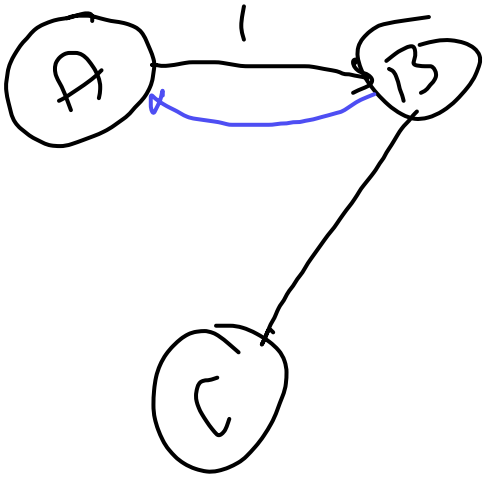


Fig. 2.2 a compententmental model starts to have some connections, with relationships between the concepts.

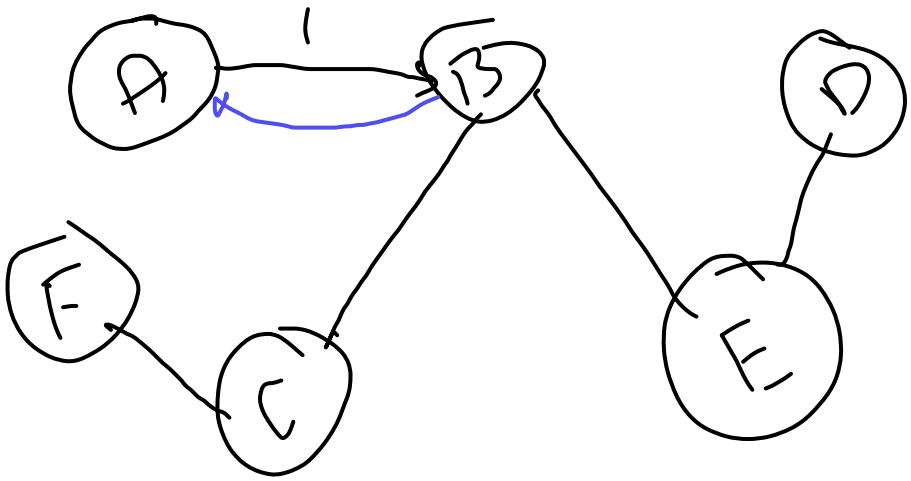


Fig. 2.3 an expert mentla model is densley connected and has more concepts in it.

We can visualize with concept maps. Which connect the ideas using relationships on the arrows.

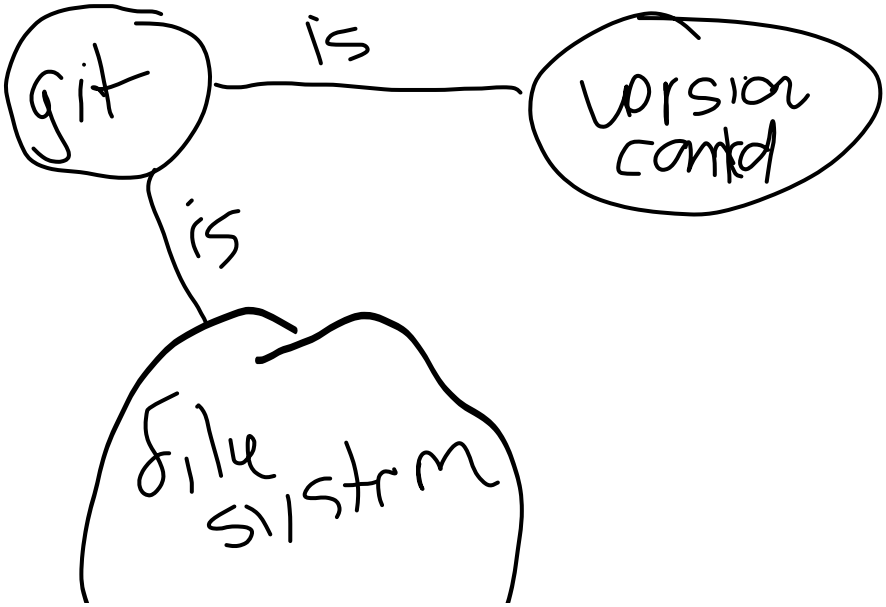


Fig. 2.4 a small concept map showing that git is an instance of both a file system and a version control system.

2.5. Why do we need this for computer systems?

> **Attention**
>
> This section contain points added here that were not discussed directly in class, but are important and will come back up

2.5.1. Systems are designed by programmers

Computer Science is not a natural science like biology or physics where we try to understand some aspect of the world that we live in. Computer Science as a discipline, like algorithms, mostly derives from Math.

Historically, Computer Science Departments were often initally formed by professors in math creating a new department or, sometimes, making a new degree programs without even creating a new department at first. In some places, CS degree programs also grew within or out of Electrical Engineering. At URI, CS grew out of math.

So, when we study computer science, while parts of it are limited by physics[1], most of it is essentially an imaginary world that is made by people. Understanding how people think, both generally, and common patterns within the community of programmers[2] understand how things work and why they are the way they are. The why can also make it easier to remember, or, it can help you know what things you can find alternatives for, or even where you might invent a whole new thing that is better in some way.

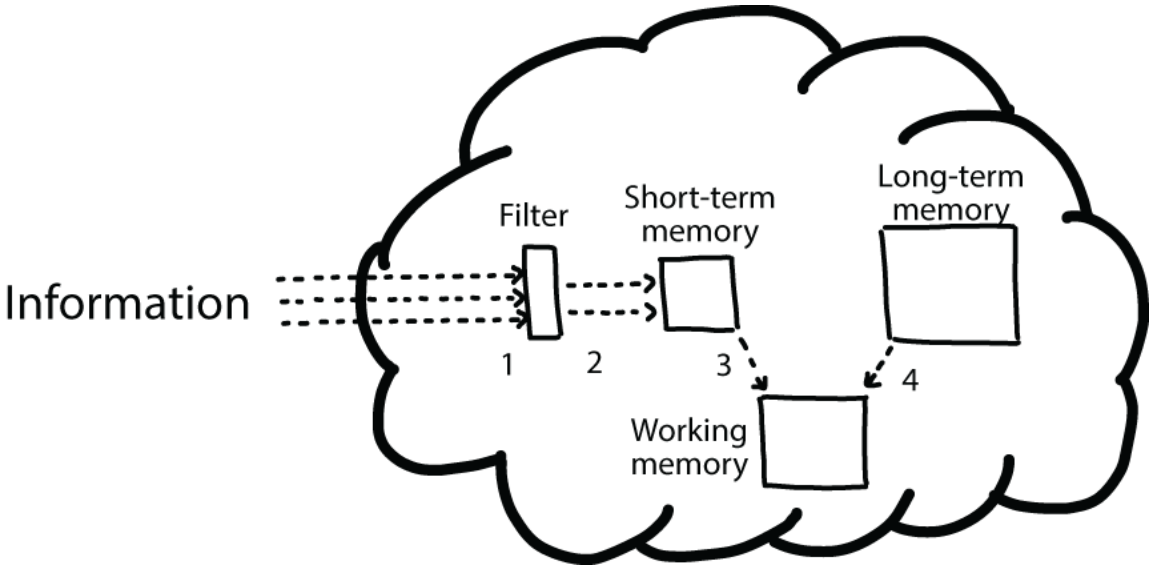


Fig. 2.5 An overview of the three cognitive processes that this book covers: STM, LTM, and working memory. The arrows labeled 1 represent information coming into your brain. The arrows labeled 2 indicate the information that proceeds into your STM. Arrow 3 represents information traveling from the STM into the working memory, where it's combined with information from the LTM (arrow 4). Working memory is where the information is processed while you think about it.

2.5.2. Context Matters

This context of how things were developed can influence how we understand it. We will also talk about the history of computing as we go through different topics in class so that we can build that context up.

2.5.3. Optimal is relative

The "best" way to do something is always relative to the context. "Best" is a vague term. It could be most computationally efficient theoretically, fastest to run on a particular type of hardware, or easiest for another programmer to read.

We will see how the best choice varies a lot as we investigate things at different levels of abstraction.

2.6. How I expect this to work

2.7. For next class

> **Note**
>
> This is what is required, before the next class and will be checked or if you don't do it you will have trouble participating in class

1. Review these notes, both rendered as html and the raw markdown in the repository.
2. find 2-3 examples of things in programming you have got working, but did not really understand. this could be errors you fixed, or something you just know you're supposed to do, but not why
3. map out your computing knowledge and add it to your kwl chart repo. this can be an image that you upload or a text-based outline.
4. Make sure you have a working environment for next week. Use slack to ask for help.
    - check that you have Python installed with Jupyter, ideally with Anaconda
    - install jupyter book
    - install GitBash on windows (optional for others)
    - make sure you have Xcode on MacOS
    - install the GitHub CLI on all OSs

2.8. More Practice

Note

Activities in this section are optional, but things that may help you prepare, or (in future classes) extend the idea.

- (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later, or try getting it to render on your own.
- (optional) read chapter 1 the programmmer's brain. Some of the ideas we talked about today are mentioned there, and it relates to where you're suppoed to be looking for things that you have done, but didn't really understand.
- try adding something to this page or the glossary of the course site or link a glossary term to an occurence of it on the site.

Hint

terms on this page that could be added to the glossary include filesystem and operating system. The jupyter book docs show how to add to a glossary and link to the glossary from another page.

2.9. Questions After Class

2.9.1. How would I learn more about version control systems?

We will talk more about this in in the tools section of the course

2.9.2. How to use github to make more meaningful repositories, instead of just a mess of files that are not properly uploaded?

We will talk a bit about how git works to get some of this. Some of this will also be dependent on the *type* of code that you are working on. For example, organizing data science examples would be different than a data science library which would be different than a backend API, even if they were all in Python.

2.9.3. Are there any benefits to using git offline vs using it only in conjunction with github?

offline you can use more powerful code editing tools and run your code more frequently. Even offline, you can use it with GitHub.

Advantages to using git without GitHub, but with a different host could be privacy reasons- if you own your own server; that you want to track changes, but do not need a backup at all; that you want different hosting for their interface that provides different features; or that you don't trust

Microsoft(who bought GitHub). Some people also do not use GitHub because they have ICE as a customer.

2.9.4. When will we be establishing the grade contracts?

We will work on grading contracts in *approximately* the third week. They will be due on or after Feburary 13th. You will have at least one week from when you get the template to the first review.

2.9.5. How can I be better at communicating documentation

We are going to focus on the tools to produce documentation sites and for distributing documentation. To learn more about documentation, in terms of what to write and where, I have an outline here. I also learned a lot from this tutorial. In general, open source community resources tend to be where you can learn more about this. Also, most langauges' official documentation will have information eg for python.

2.9.6. How often are we supposed to update our KWL Charts?

You will be prompted (either in class or as a class preparation) when I expect it at a minimum, but you can also add to it when you think of things.

2.9.7. Do we get a notification when you post the notes, or do we just check periodically?

The #siteupdates channel in slack shows when commits are pushed to the repository for the site, you can also create a more custom version of notification using either the GitHub watch or the GitHub bot in a slack DM to yourself, using `/github help`.

---

[1] when we are *really* close to the hardware

[2] Of course, not *all* programmers think the same way, but when people spend time together and communicate, they start to share patterns in how they think. So, while you do **not** have to think the same way as these patterns, knowing what they are will help you reading code, and understanding things.

3. How do I use git offline

3.1. Todays Goals

- just enough bash
- offline git basics
- practice with issues as something to *do* while we work with git offline

3.2. Closing an Issue with a commit

We can close issues with commits, we'll first review making commits in browser to see how that works, then we will do it offline again.

Use the create a test repo for today's class it will have some issues it in upon creation.

Notice what happened:

- the file is added and the commit has the the message
- the issue is closed
- if we go look at the closed issues, we can see on the issue that it was linked to the commit
- from the issue, we can see what the changes were that made are supposed to relate to this

> **Note**
>
> we can still comment on an issue that is already closed.

> **Try it Yourself**
>
> We can also re-open issues. Try that out and then make a new commit to close it again. Why is this a useful feature for GitHub?

3.3. Getting Set up Locally

Opening different terminals

- default terminal on mac, ue the `bash` command to use bash (zsh will be mostly the same; it's derivative, but to ensure exactly the same as mine use bash)
- use gitbash on Windows

To change directory

```
cd path/to/go/to
```

To make a directory (folder) for things in this course (or in my case for inclass time)

```
mkdir sysinclass
```

Then we have to `cd` into that new folder:

```
cd sysinclass/
```

To view where we are, we **p**rint **w**orking **d**irectory

```
pwd
```

View files

```
ls
```

It's empty for now, but we will change that soon.

3.4. Using Git and GitHub locally

3.4.1. Authenticating with GitHub

There are many ways to authenticate securely with GitHub and other git clients. We're going to use *easier* ones for today, but we'll come back to the third, which is a bit more secure and is a more general type of authentication.

1. GitHub CLI: enter the following and follow the prompts.

   ```
   gh auth login
   ```

2. personal access token. This is a special one time password that you can use like a password, but it is limited in scope and will expire (as long as you choose settings well)
3. ssh keys

3.4.2. Cloning a repository

Cloning a repository makes a local copy of a remote git repository.

We can clone in two different ways, with git only or with the GitHub CLI tools.

> **Warning**
>
> My repository, like yours, is private so copying these lines directly will not work. You will have to replace my GitHub username with your own.

with the GitHub CLI:

```
gh repo clone introcompsys/github-in-class-brownsarahm
```

with git only:

```
git clone https://github.com/introcompsys/github-in-class-brownsarahm.git
```

> **Important**
>
> the git only version can be used with git repositories that are hosted anywhere, for example on BitBucket or GitLab

Either way we will see something like this:

```
Cloning into 'github-in-class-brownsarahm'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 15 (delta 2), reused 5 (delta 1), pack-reused 0
Receiving objects: 100% (15/15), done.
Resolving deltas: 100% (2/2), done.
```

Now we can check what happened using `ls`

```
github-in-class-brownsarahm
```

When we clone a repository, it creates a new directory and downloads all of the contents and the repository information, including where it came from so that we can send our new changes back there.

3.4.3. Adding new files to a Repository Locally

We first go into that folder.

```
cd github-in-class-brownsarahm/
```

We can see what is there.

```
ls
```

```
README.md
touch about.md
```

```
ls
```

and then we see the list of files

```
README.md       about.md
```

```
git status
```

which gives us the following output

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        about.md

nothing added to commit but untracked files present (use "git add" to track)
```

```
ls -a
.               .git            README.md
..              .github         about.md
```

```
git add .
```

again, we can check what git knows about

```
git status
```

and take note of the key differences from before.

```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   about.md

git commit -m 'create empty about'
[main b81cf15] create empty about
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 about.md
```

3.4.4. Text editing on the terminal

```
nano about.md
```

then we can edit the file, adding some content and then write out (to save) and then exit nano

3.4.5. Commiting Changes to a file

```
git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   about.md

no changes added to commit (use "git add" and/or "git commit -a")
```

```
git add about.md
```

```
git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   about.md
```

```
git commit -m 'complete about closes #2
> '

[main 17320fc] complete about closes #2
 1 file changed, 4 insertions(+)
```

```
git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

3.4.6. Sending Changes to GitHub

```
git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 535 bytes | 535.00 KiB/s, done.
Total 6 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/introcompsys/github-in-class-brownsarahm.git
   f707186..17320fc  main -> main
```

Notice on GitHub that the issue is now closed. and the commit is referenced and shows the changes.

3.5. A third way to close an issue

See the `classmate` issue:

```
owner:
- [ ] give a class mate access to the repo
- [ ] assign this issue to them

classmate:
- [ ] add `classmate.md` with your name and expected graduation on  a bracn `classmate`
- [ ] open a PR that will close this issue
```

1. Do the `owner` list in your repo (the on that ends with your user name)
2. Do the classmate actions in *another person's repo*
3. In *your own* repo, on the PR made by your class mate, tag `@sp21instructors` in a comment and then merge the PR.

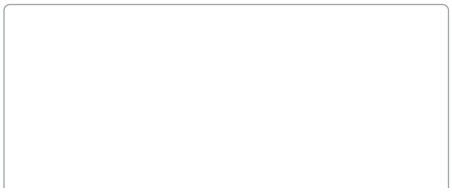
3.5.1. Controlling Access

When you are a repository owner or organization level admin, you can change who has access to a repository on the settings tab.

Since the course repositories are in an organization, you get to choose the role for each collaborator or team. The GitHub Docs define the roles and permissions, so you can always refer t that to choose the right one.

> **Note**
>
> I seeded these notes by using the `Export text` option from the mac terminal app. Other terminals have similar options and you can always get *only* the list of commands you have run with `history`

3.6. Prepare for next class



> **Important**
>
> If you do the last two after Thursday that's okay but the rest are important practice to reinforce what you just learned and to prepare for what we will do in class on Thursday.

> **Tip**
>
> The "prepare for class" section is the minimum, the "more practice" is more like what will be reqiured for a B, but there's ramp up time at the beginning.
> "Prepare for class" will generally be due at the time of the next class because we will use that stuff in the next class session. "More practice" you can come back to as fits in your schedule, (though I don't recommend too big of a lag).

1. Complete the classmate issue in your in-class repository.
2. read the notes PR, add or comment on a tip, resource, a bit of history in a sidebar or additional end of class question
3. try using git in your IDE of choice, log any challenges you have on the practice repo (`github-in-class-username`), and tag @sp22instructors on GitHub. You can use either repo we have made in class, or one for an assignment in another course.
4. using your terminal, download your KWL repo and update your 'learned' column on a new branch
5. answer the questions below in a new markdown file, `gitoffline.md` in your KWL on your new branch and push the changes to GitHub
6. Create a PR from your new branch to main **do not merge this until instructed**
7. add your programming challenge(s) you have had as issues to our private repo or to the course website repo if you like. Put one 'challenge/question' per issue so that we can close them as addressed. See last class notes for prompt.
8. Create or comment on a discussion thread in the private repo about the part of CS/ type of programming you like best/what you want to do post graduation.

> **Tip**
>
> remember you can copy text from here directly into your file

Questions:

```
## Reflection
1. Describe the staging area (what happens after git add) in your own words. Can you think of an analogy for it? Is there anything similar in a hobby you have?
2. what step is the hardest for you to remember?
3. Compare and contrast using git on the terminal and through your IDE. when would each be better/worse?
4. Describe the commit that closed the `classmate` issue, who does it attribute the fix to?
```

3.7. More Practice

1. Find the "Try it yourself" boxes in these notes, try them and add notes/ responses under a `## More Practice` heading in your `gitoffline.md` file of your KWL repo.
2. Download the course site repo via terminal.
3. Explore the difference between `git add` and `git commit` try committing and pushing without adding, then add and push without committing. Describe what happens in each case in your `gitoffline.md`

3.8. Questions After class

3.8.1. Can we push the file before commit? And if so, what will happen?

●●●

3.8.2. what is the difference between add and commit? They seem to do the same thing to me.

●●●

3.8.3. Is there a point to doing multiple commits before a push.

●●●

3.8.4. Can I just use GitHub Desktop if I have it already?

●●●

3.9. Resources:

What is Git?

Interactive Git Cheat Sheet

4. Why Do I Need to Use a terminal?

We will go back to the same repository we worked with on Tuesday, for me that was

```
cd Documents/teaching/sysinclass/github-in-class-brownsarahm/
```

We can use `touch` that we saw Tuesday to create many files at once:

```
touch abstract_base_class.py helper_functions.py important_classes.py alternative_classes.py README.md LICENSE.md
CONTRIBUTING.md setup.py tests_abc.py test_help.py test_imp.py test_alt.py overview.md API.md _config.yml
_toc.yml philosophy.md example.md Untitled.ipynb Untitled01.ipynb Untitled02.ipynb
```

we got an error from this:

```
-bash: _toc.yml: command not found
```

we can intermet this, bash thought `_toc.yml` was a command. That means there was a hard to see accidental line break in the text above.

If we didn't know what that meant, we could also investigate further using `ls` to list.

```
ls
```

we see we have most of the files actually created,

```
API.md                  abstract_base_class.py  test_alt.py
CONTRIBUTING.md         alternative_classes.py  test_help.py
LICENSE.md              helper_functions.py     test_imp.py
README.md               important_classes.py    tests_abc.py
_config.yml             overview.md
about.md                setup.py
```

we can use the up arrow key to get back the last line.

```
_toc.yml philosophy.md example.md Untitled.ipynb Untitled01.ipynb Untitled02.ipynb
```

and add `touch` at the start of it to create those last few files.

```
touch _toc.yml philosophy.md example.md Untitled.ipynb Untitled01.ipynb Untitled02.ipynb
```

and confirm

```
ls
```

4.1. Scenario

> **Note**
>
> a few of you asked about learning how to organize projects. While our main focus in this class session is the `bash` commands to do it, the *task* that we are going to do is to organize a hypothetical python project

Now we have all of these files, named in abstract ways to signal hypothecial contents and suggest how to organize them.

```
API.md                  _toc.yml                philosophy.md
CONTRIBUTING.md         about.md                setup.py
LICENSE.md              abstract_base_class.py  test_alt.py
README.md               alternative_classes.py  test_help.py
Untitled.ipynb          example.md              test_imp.py
Untitled01.ipynb        helper_functions.py     tests_abc.py
Untitled02.ipynb        important_classes.py
_config.yml             overview.md
```

First we're goign to paste some contents (shared from prismia, view below) in to the readme with nano

```
nano README.md
```

We can view the contents of the file using `cat` to print the contents to the terminal output.

```
cat README.md
```

and we see:

```
# GitHub Practice

Name: sarah
|file | contents |
| ------| ------- |
| abstract_base_class.py | core abstract classes for the project |
| helper_functions.py | utitly funtions that are called by many classes |
| important_classes.py | classes that inherit from the abc |
| alternative_classes.py | classes that inherit from the abc |
| LICENSE.md | the info on how the code can be reused|
| CONTRIBUTING.md | instructions for how people can contribute to the project|
| setup.py | file with function with instructions for pip |
| tests_abc.py | tests for constructors and methods in abstract_base_class.py|
| tests_helpers.py | tests for constructors and methods in helper_functions.py|
| tests_imp.py | tests for constructors and methods in important_classes.py|
| tests_alt.py | tests for constructors and methods in alternative_classes.py|
| API.md | jupyterbook file to generate api documentation |
| _config.yml | jupyterbook config for documentation |
| _toc.yml | jupyter book toc file for documentation |
| philosophy.md | overview of how the code is organized for docs |
| example.md | myst notebook example of using the code |
| Untitled*.ipynb | jupyter notebook from dev, not important to keep |
```

this explains each file a little bit more than the name of it does. We see there are sort of 5 groups of files:

- about the project/repository
- code that defines a python module
- test code
- documentation
- extra files that “we know” we can delete.

4.2. Making Directories

First we will make directories. We saw `mkdir` on Tuesday

```
mkdir docs/
```

This doesn’t return anything, but we can see the effect with `ls`

```
ls
```

```
API.md                  _toc.yml                overview.md
CONTRIBUTING.md         about.md                philosophy.md
LICENSE.md              abstract_base_class.py  setup.py
README.md               alternative_classes.py  test_alt.py
Untitled.ipynb          docs                    test_help.py
Untitled01.ipynb        example.md              test_imp.py
Untitled02.ipynb        helper_functions.py     tests_abc.py
_config.yml             important_classes.py
```

We might not want to make them all one at a time. Like with `touch` we can pass multiple names to mkdir with spaces between to make multiple at once.

```
mkdir tests mymodule
```

and again use `ls` to see the output

```
API.md                  about.md                philosophy.md
CONTRIBUTING.md         abstract_base_class.py  setup.py
LICENSE.md              alternative_classes.py  test_alt.py
README.md               docs                    test_help.py
Untitled.ipynb          example.md              test_imp.py
Untitled01.ipynb        helper_functions.py     tests
Untitled02.ipynb        important_classes.py    tests_abc.py
_config.yml             mymodule
_toc.yml                overview.md
```

4.3. Moving files

we can move files with `mv`. We’ll first move the philosophy.md file into docs and check that it worked.

```
mv philosophy.md docs/
ls
```

```
API.md                  _toc.yml                mymodule
CONTRIBUTING.md         about.md                overview.md
LICENSE.md              abstract_base_class.py  setup.py
README.md               alternative_classes.py  test_alt.py
Untitled.ipynb          docs                    test_help.py
Untitled01.ipynb        example.md              test_imp.py
Untitled02.ipynb        helper_functions.py     tests
_config.yml             important_classes.py    tests_abc.py
```

4.3.1. Getting help in bash

To learn more about the mv command, we can use the man(ual) file.

```
man mv
```

use enter/return or arrows to scroll and `q` to quit

If we type something wrong, the error message also provides some help

```
mv ls
usage: mv [-f | -i | -n] [-v] source target
       mv [-f | -i | -n] [-v] source ... directory
```

We can use man on any bash command to see the options so we do not need to remember them all, or go to the internet every time we need help. We have high quality help for the details right in the shell, if we remember the basics.

```
man ls
```

```
ls -hl
```

```
total 16
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 API.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 CONTRIBUTING.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 LICENSE.md
-rw-r--r--  1 brownsarahm  staff   1.2K Feb  3 12:56 README.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:52 Untitled.ipynb
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:52 Untitled01.ipynb
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:52 Untitled02.ipynb
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 _config.yml
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:52 _toc.yml
-rw-r--r--  1 brownsarahm  staff    14B Feb  1 13:23 about.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 abstract_base_class.py
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 alternative_classes.py
drwxr-xr-x  3 brownsarahm  staff    96B Feb  3 13:04 docs
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:52 example.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 helper_functions.py
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 important_classes.py
drwxr-xr-x  2 brownsarahm  staff    64B Feb  3 13:01 mymodule
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 overview.md
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 setup.py
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 test_alt.py
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 test_help.py
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 test_imp.py
drwxr-xr-x  2 brownsarahm  staff    64B Feb  3 13:01 tests
-rw-r--r--  1 brownsarahm  staff     0B Feb  3 12:51 tests_abc.py
```

In some versions of bash we can use:

```
mv --help
```

4.3.2. Moving multiple files with patterns

let's look at the list of files again.

```
cat README.md
```

```
# GitHub Practice

Name: sarah
|file | contents |
| ------| ------- |
| abstract_base_class.py | core abstract classes for the project |
| helper_functions.py | utitly funtions that are called by many classes |
| important_classes.py | classes that inherit from the abc |
| alternative_classes.py | classes that inherit from the abc |
| LICENSE.md | the info on how the code can be reused|
| CONTRIBUTING.md | instructions for how people can contribute to the project|
| setup.py | file with function with instructions for pip |
| test_abc.py | tests for constructors and methods in abstract_base_class.py|
| test_helpers.py | tests for constructors and methods in helper_functions.py|
| test_imp.py | tests for constructors and methods in important_classes.py|
| tests_alt.py | tests for constructors and methods in alternative_classes.py|
| API.md | jupyterbook file to generate api documentation |
| _config.yml | jupyterbook config for documentation |
| _toc.yml | jupyter book toc file for documentation |
| philosophy.md | overview of how the code is organized for docs |
| example.md | myst notebook example of using the code |
| scratch.ipynb | jupyter notebook from dev |
```

> **Note**
>
> this is why good file naming is important even if you have not organized the whole project yet, you can use the good conventions to help yourself later.

We see that the ones with similar purposes have similar names.

We can use `*` as a wildcard operator and then move will match files to that pattern and move them all. We'll start with the two `yml` (yaml) files that are both for the documentation.

```
mv *.yml docs/
```

4.3.3. Renaming a single file with mv

We see that most of the test files start with `test_` but one starts with `tests_`. We could use the pattern `test*.py` to move them all without conflicting with the directory `tests/` but we also want consistent names.

We can use `mv` to change the name as well. This is because "moving" a file and is really about changing its path, not actually copying it from one location to another and the file name is a part of the path.

```
mv tests_abc.py test_abc.py
ls
```

now that it's fixed

```
API.md                  abstract_base_class.py  setup.py
CONTRIBUTING.md         alternative_classes.py  test_abc.py
LICENSE.md              docs                    test_alt.py
README.md               example.md              test_help.py
Untitled.ipynb          helper_functions.py     test_imp.py
Untitled01.ipynb        important_classes.py    tests
Untitled02.ipynb        mymodule
about.md                overview.md
```

We can use the pattern `test_*` to move them all.

```
mv test_* tests/
ls
```

```
API.md                  Untitled02.ipynb        helper_functions.py
CONTRIBUTING.md         about.md                important_classes.py
LICENSE.md              abstract_base_class.py  mymodule
README.md               alternative_classes.py  overview.md
Untitled.ipynb          docs                    setup.py
Untitled01.ipynb        example.md              tests
```

Now we can move all of the other .py files to the module

```
mv *.py mymodule/
ls
```

```
API.md                  Untitled01.ipynb        mymodule
CONTRIBUTING.md         Untitled02.ipynb        overview.md
LICENSE.md              about.md                tests
README.md               docs
Untitled.ipynb          example.md
```

4.4. Working with relative paths

Let's review our info again

```
cat README.md
# GitHub Practice

Name: sarah
|file | contents |
| ------| ------- |
| abstract_base_class.py | core abstract classes for the project |
| helper_functions.py | utitly funtions that are called by many classes |
| important_classes.py | classes that inherit from the abc |
| alternative_classes.py | classes that inherit from the abc |
| LICENSE.md | the info on how the code can be reused|
| CONTRIBUTING.md | instructions for how people can contribute to the project|
| setup.py | file with function with instructions for pip |
| tests_abc.py | tests for constructors and methods in abstract_base_class.py|
| tests_helpers.py | tests for constructors and methods in helper_functions.py|
| tests_imp.py | tests for constructors and methods in important_classes.py|
| tests_alt.py | tests for constructors and methods in alternative_classes.py|
| API.md | jupyterbook file to generate api documentation |
| _config.yml | jupyterbook config for documentation |
| _toc.yml | jupyter book toc file for documentation |
| philosophy.md | overview of how the code is organized for docs |
| example.md | myst notebook example of using the code |
| scratch.ipynb | jupyter notebook from dev |
```

We've made a mistake, `setup.py` is actually instructions that need to be at the top level, not inside the module's sub directory.

We can get it back using the relative path to the file and then using `.` to move it to where we "are" sicne we are in the top level directory still.

```
mv mymodule/setup.py .
ls
```

```
API.md                  Untitled01.ipynb        mymodule
CONTRIBUTING.md         Untitled02.ipynb        overview.md
LICENSE.md              about.md                setup.py
README.md               docs                    tests
Untitled.ipynb          example.md
```

Or, if we put it back temporarily

```
mv setup.py mymodule/
```

We can cd to where we put it

```
cd mymodule/
ls
```

```
abstract_base_class.py  helper_functions.py     setup.py
alternative_classes.py  important_classes.py
```

and move it up a level using `..`

```
mv setup.py ..
ls
```

```
abstract_base_class.py  helper_functions.py
alternative_classes.py  important_classes.py
```

then the `..` to go up a level gets us back to where we were.

```
cd ..
ls
```

```
API.md                  Untitled01.ipynb        mymodule
CONTRIBUTING.md         Untitled02.ipynb        overview.md
LICENSE.md              about.md                setup.py
README.md               docs                    tests
Untitled.ipynb          example.md
```

Now we'll move the last few docs files.

```
mv API.md docs/
mv example.md docs/
mv overview.md docs/
ls
```

4.5. Removing files

We still have to deal with the untitled files that we know we don't need any more.

```
CONTRIBUTING.md         Untitled01.ipynb        mymodule
LICENSE.md              Untitled02.ipynb        setup.py
README.md               about.md                tests
Untitled.ipynb          docs
```

we can delete them with `rm` and use `*` to delet them all.

```
rm Untitled*
```

```
ls
```

now we have a nice clean repository.

```
CONTRIBUTING.md README.md       docs            setup.py
LICENSE.md      about.md        mymodule        tests
```

4.6. Copying

The typical contents of the README we would also want in the documentation website. We might add to the file later, but that's a good start. We can do that by copying.

When we copy we designate the file to copy and a path/name for the copy we want to make.

```
cp README.md docs/index.md
cd docs/
ls
```

```

API.md          _toc.yml        index.md        philosophy.md
_config.yml     example.md      overview.md
```

we can check the contents of the file too:

```
cat index.md
```

```
# GitHub Practice

Name: sarah
|file | contents |
| ------| ------- |
| abstract_base_class.py | core abstract classes for the project |
| helper_functions.py | utitly funtions that are called by many classes |
| important_classes.py | classes that inherit from the abc |
| alternative_classes.py | classes that inherit from the abc |
| LICENSE.md | the info on how the code can be reused|
| CONTRIBUTING.md | instructions for how people can contribute to the project|
| setup.py | file with function with instructions for pip |
| tests_abc.py | tests for constructors and methods in abstract_base_class.py|
| tests_helpers.py | tests for constructors and methods in helper_functions.py|
| tests_imp.py | tests for constructors and methods in important_classes.py|
| tests_alt.py | tests for constructors and methods in alternative_classes.py|
| API.md | jupyterbook file to generate api documentation |
| _config.yml | jupyterbook config for documentation |
| _toc.yml | jupyter book toc file for documentation |
| philosophy.md | overview of how the code is organized for docs |
| example.md | myst notebook example of using the code |
| scratch.ipynb | jupyter notebook from dev |
```

it matches .

4.7. More relative paths

We need a `__init__.py` in the mymodule directory but we are in the docs directory currently. No problem!

```
touch ../mymodule/__init__.py
```

```
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ../README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../CONTRIBUTING.md
        ../LICENSE.md
        ./
        ../mymodule/
        ../setup.py
        ../tests/

no changes added to commit (use "git add" and/or "git commit -a")
```

note we have both changes and untracked files… but wherre is the docs folder?

we're in there so all of the files are listed relative to there, so it's the `./` line to say that we are currently in an untracked directory. Git status doesn't look inside directories it doens't know if it should track or not.

if we go back to the top level

```
cd ..
```

```
git status
```

> **Tip**
>
> Compare these two outputs carefully. Getting used to noticing these details will help you get yourself unstuck!

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        CONTRIBUTING.md
        LICENSE.md
        docs/
        mymodule/
        setup.py
        tests/

no changes added to commit (use "git add" and/or "git commit -a")
```

Then we can add the changes and push

```
git add .
git commit -m 'insclass 2-3'
```

```
[main c15cf43] insclass 2-3
 20 files changed, 41 insertions(+)
 create mode 100644 CONTRIBUTING.md
 create mode 100644 LICENSE.md
 create mode 100644 docs/API.md
 create mode 100644 docs/_config.yml
 create mode 100644 docs/_toc.yml
 create mode 100644 docs/example.md
 create mode 100644 docs/index.md
 create mode 100644 docs/overview.md
 create mode 100644 docs/philosophy.md
 create mode 100644 mymodule/__init__.py
 create mode 100644 mymodule/abstract_base_class.py
 create mode 100644 mymodule/alternative_classes.py
 create mode 100644 mymodule/helper_functions.py
 create mode 100644 mymodule/important_classes.py
 create mode 100644 setup.py
 create mode 100644 tests/test_abc.py
 create mode 100644 tests/test_alt.py
 create mode 100644 tests/test_help.py
 create mode 100644 tests/test_imp.py
```

```
git push
```

```
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 1.22 KiB | 1.22 MiB/s, done.
Total 7 (delta 0), reused 1 (delta 0), pack-reused 0
To https://github.com/introcompsys/github-in-class-brownsarahm.git
   17320fc..c15cf43  main -> main
```

> **Important**
>
> if your push gets rejected, read the hints, it probably has the answer. We will come back to that error though

4.8. Git order of operations

above since we didn't make a branch we pushed to main.

```
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

We could make the file changes first and then make the branch we want to commit them too as well. it's best to make the branch first so you don't forget, but it is an option

```
touch test_file.md
```

```
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test_file.md

nothing added to commit but untracked files present (use "git add" to track)
```

```
git checkout -b test
```

```
git status
```

```
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test_file.md

nothing added to commit but untracked files present (use "git add" to track)
```

4.9. Recap

Why do I need a terminal

1. replication/automation
2. it's always there and doesn't change
3. it's faster one you know it (also see above)

> So, is the shell the feature that interacts with the operating system and then the terminal is the gui that interacts with the shell?

This week we saw two really important tools. Next week we're going to take a sort of archealogical look at computer systems, first software then hardware to wrap up our overivew and exploration of what all these topics we're going to cover in class are and how they relate.

4.10. Prepare for the next class

1. Review the notes
2. Reorganize a folder on your computer (good candidate may be desktop or downloads folder), using only a terminal to make new directories, move files, check what's inside them, etc. Answer reflection questions (will be in notes) in a new file, `terminal.md` in your kwl repo.
3. Add a glossary to the site to define a term or cheatsheet entry to describe a command that we have used so far.
4. Examine a large project you have done or by finding an open source project on GitHub. Answer the reflection questions in `software.md` in your kwl repo. (will be in notes)

4.10.1. Terminal File moving reflection

Start with a file explorer open, but then try to close it and use only command line tools to explore and make your choices

```
1. Did this get easier toward the end?
1. Use the history to see which commands you used and how many times each, make a table below.
1. Did you have to look up how to do anything we had not done in class?
1. When do you think that using the terminal will be better than using your GUI file explorer?
1. What questions/challenges/ relfections do you have after this?
1. What kinds of things might you want to write a bash script for given what you know in bash so far? come up with 1-2
scenarios
```

4.10.2. Software Reflection

```
1. link to public repo if applicable or title of your project
1. What types of files are there that are not code?
1. What different types of code files are in the project? Do they serve different goals?
1. Is it all in one language or are there multiple?
1. Try to figure out (remember) how the project works. What types of things, without running the code can you look at at a
high level?
```

4.11. More Practice

1. Try to do as many things as possible on the terminal for a whole week.
2. Make yourself a bash cheatsheet (and/or contribute to one on the course site)
3. Read through part 1 of the programmer's brain and try the exercises, especially in chapter 4.

4.12. Questions at the end of class

4.12.1. what makes a file be able to stage? In other words, what does “staging” actually mean?

●●●

4.12.2. how do you make a branch on the GitHub site?

●●●

4.12.3. how do we write a bash script?

●●●

4.12.4. Where can I learn more about the GitHub flow?

●●●

4.12.5. How do we link a project we already have made to a new git repository

●●●

4.12.6. what can happen if I moved a file but I had another file pointing to the old address

●●●

4.12.7. how in depth will our bash scripting go in this class? clearly it can be used for a lot of different things

●●●

4.12.8. How might you go about re-instantiating a repo? I.e. starting back from whatever the origin is

●●●

4.13. Resources

Bash Cheat Sheet

Alternative Shells

5. Review and Abstraction

5.1. Can I reset a Git repository?

1. Find the hash number for the first commit of your in-class repo.
2. On your terminal, navigate to that repo.
3. Check out that commit `git checkout <paste hash here>`
4. Look back at what happened, using `ls`
5. Make a new branch called 'reset' and push that branch to GitHub.
6. Switch back to the current version of the repo
7. In browser, compare the two branches, visually.

5.2. Moving Files Requires Care

A question from last week was what happens if we move a file to an address where there already is one?

```
touch fa
echo "file one" > fa
cat fa
```

```
echo "file two" > fb
cat fb
```

```
mv fa fb
cat fb
```

5.3. Standard In, Out, and Error

We have been using bash to move files around and explore the system so far. In doing so we have also seen `cat` that we saw would display the contents of a file.

What it actually does is a little bit different. Let's try `cat` without putting a file name after it.

```
cat
```

It waits for us to type, if we type and then press enter, what we typed is displayed and it keeps waiting.

Use control/command + d to exit.

`cat` actually looks at standard input, a special file in our computer that gets the input from the keyboard if we don't tell it otherwise.

```
cat fa
```

is a shortcut basically for

```
cat < fa
```

which says explicitly, get ready to the contents of standard in to standard out and then put the contents of `fa` and put it on standard in. The arrow is called a redirect.

We used `echo` to write to a file above in the little experiment.

```
echo "some text" > a_file
cat a_file
```

and we get output as before

```
some text
```

That line has two new parts both `echo` and the `<` syntax. Let's try `echo` by itself.

```
echo "hello world"
```

and we see

```
hello world
```

Echo puts content on standard out, which is a special file that is by default linked to the display of the terminal. It could have been set elsewhere, and that's what the redirect does.

```
echo "some text" > a_file
cat a_file
```

This sends that text to standard out and redirects standard out to the file `a_file`

```
some text
```

if we use two arrows it will append instead of overwriting.

```
echo "some more text" >> a_file
cat a_file
```

```
some text
some more text
```

```
man echo
```

Name	File descriptor	Description	Abbreviation
Standard input	0	The default data stream for input, for example in a command pipeline. In the terminal, this defaults to keyboard input from the user.	stdin
Standard output	1	The default data stream for output, for example when a command prints text. In the terminal, this defaults to the user's screen.	stdout
Standard error	2	The default data stream for output that relates to an error occurring. In the terminal, this defaults to the user's screen.	stderr

Important

GitBash does not support man the reasons athe developer does not want to are also visible. You can use the help option `-help` try the help command.
The help is slightly different from the man pages overall.

Alternatively, you can modify your environment further. Enabling the Windows subsystem for Linux is one option. So is booting into Linux for example ubuntu that is installed on a flash drive. This uses the flas drive as the hard drive for the operating system. This option creates 2 whole "computers" at the software level, that use the same hardware.

5.4. Layers of a Computer System

1. Application
2. Algorithm
3. Programming Language
4. Assembly Language
5. Machine Code
6. Instruction set Architecture
7. Micro Architecture
8. Gates/registers
9. Devices (transistors)
10. Physics

5.5. Prepare for Next Class

1. install h/w simulator
2. Add a glossary, cheatsheet entry, or historical context/facts about the things we have learned to the site.
3. Review past classes prep/more practice and catchup if appropriate
4. Map out how you think about data moving through a small program using the levels of abstraction. Add this to a markdown table in your KWL chart repo called `abstraction.md`. If you prefer a different format than a table, that is okay, but put it in your KWL repo. It is okay if you are not sure, the goal is to think through this.

5.6. More Practice

1. Once your PRs in your KWL are merged so that main and feedback match, pull to updates your local copy. In a new terminal window, navigate there and then move the your KWL chart to a file called `chart.md`. Create a new README files with a list of all the files in your repo. Use `history N` (N is the number of past commands that history will return) and redirects to write the steps you took to `reorg.md`. Review that file to make sure it doesn't have extra steps in it and remove any if needed using nano then commit that file to your repo.
2. find a place where there is a comment in the course notes indicating content to add and submit a PR adding that content. This could be today's notes or a past day's.
3. Add a new file to your KWL repo called `stdinouterr.md` Try the following one at a time in your terminal and describe what happens and explain why or list questions for each in the file. What tips/reminders would you give a new user (or yourself) about using redirects and `echo`?
    - `echo "hello world" > fa > fb`
    - `echo "a test" > fc fd`
    - `> fe echo "hi there"`
    - `echo "hello " > ff world`
    - `<ff echo hello`
    - `fa < echo hello there`
    - `cat`

Tip

pay attention to how many steps you do to know what value of N to use. You should be able to do all of number 1 in your terminal.

Hint

redirects (>) can be used with other commands, not only echo

Note

If you get stuck on any of these create an issue and tag @sp22instructors

5.7. Questions After class

5.7.1. What happens if I don't meet the requirements for the grade I contract for?

You will be able to revise the contract if you choose to earn a different grade. The revision will also have to get approved, but it is an option. If you do not fulfill your contract in the form it is stated at the end of the semester, you will get an incomplete and then we will make a plan to change that to a letter.

5.7.2. When can we expect approved pull requests?

Feedback hours mostly, which are 5-6pm on Tuesday and 4-5pm on Thursday

5.7.3. Can you echo multiple files at the same time?

Echo sends only to stdout, we can redirect stdout to a different file, but echo specifically goes to the one place. The question then becomes can we redirect std out to two places at once, which we cannot do with redirects alone either. However, we can send output multiple places using a few more commands. We'll come back to this one next week. Also try the last exercise under more practice.

5.7.4. does this character "<" do something different than the redirect character ">"?

They are both redirections < is less common.

5.7.5. What's under the hood with >?

The official documentation for redirections describes some. We'll come back to it after we talk through the overview of hardware a bit more in the next class.

5.7.6. What level of understanding of the abstraction stack is typical for a programmer?

This is going to depend on their training. A person who programs after a short coding bootcamp or a scientist who codes in order to do their scientific research may only understnd the application and algorithm layers and be perfectly content and able to fulfill their goals. With a Computer Science degree we hope that by the end you have down to Assembly very strong and the basic ideas down to gates/ registers. Someone with a Computer Engineering degree will have more understanding at the lower levels but maybe less in programming languages and algorithms than a CS degree. Plus

anyone can go learn more and forget things they once knew. Also how much you actively use this knowledge is going to vary. Someone who writes firmware is going to be focused on a very different point in that stack that someone who develops new machine learning applications for example.

5.8. Why you would want to override the name/path of a file?

You probably would not want to do this very often, but for example, I do this when I download a newer version of a file and my browser names the new version something like `file_name (1).ext` I actually wanted to overwrite but it assumes that I do not, so I use `mv file_name\ (1).ext file_name.ext` to overwrite the new, updated content into the better file name (without the space and (1))

5.8.1. How does Authentication on GitHub work?

GitHub provides a whole page on that with advice and links. We will also come back to this in order to talk about ssh keys and ssh generally next week.

5.9. Resources

Interactive Git Cheat Sheet

6. Survey of Hardware

6.1. Where does assembly come from?

In order to watch what happens in hardware when a program runs, which is our goal today, we will execute a compiled program. It is written in assembly code. Typically, we do not write assembly, but instead it is produced by the compiler.

Technically assembly instructions and the values we operate on are represented in binary on hardware, but these more readable level instructions, though basic, are a useful abstraction. These instructions are also hardware nonspecific.

6.2. Using the simulator

On MacOS:

```
cd path/nand2tetris/tools
bash CPUEmulator.sh
```

On Windows: Double click on the CPUEmulator.bat file

We're going to use the test cases from the book's project 5:

1. Load Program
2. Navigate to nand2tetris/projects/05

We're not going to *do* project 5, which is to build a CPU, but instead to use the test.

For more on how the emulator works see the CPU Emulator Tutorial.

For much more detail about how this all works chapter 4 of the related text book has description of the machine code and assembly language.

6.3. Adding Constants

We'll use add.hack first.

This program adds constants, 2+3.

It is a program, assembly code that we are loading to the simulator's ROM, which is memory that gets read only by the CPU.

Run the simulator and watch what each line of the program does.

Notice the following:

- to compute with a constant, that number only exists in ROM in the instructions
- to write a value to memory the address register first has to be pointed to what where in the memory the value will go, then the value can be sent there

> **Try it yourself**
>
> Write code in a high level language that would compile into this program. Try writing two different variations.

```
for itme in list:
```

6.4. Using a variable

Next use the max.hack.

This one compares the values at two memory locations. In order to use it, you have to write values to the RAM 0 and RAM 1 manually.

It first takes the value from each location and passes the first value to D, then uses the ALU to assign the difference between the two values to D. Then, if the value is greater than 0, it jumps to line to line 10 in the ROM (of the instructions). Line 10, sets A to 0 and 11 sets D to the value from RAM 0. If, instead, the value is less than 0, A is set 1, then and D is set to that value. Then the program points A to 2 and writes the value from D there.

> **Try it yourself**
>
> Write code in a high level language that would compile into this program. Try writing multiple different versions.
>
> What does this program assume has happened that it doesn't include in its body.

6.5. Using output

The rect.hack program writes to output, by using a specific memory location that is connected to the output.

> **Try it yourself**
>
> Try working through this program using the tools to understand what it does

6.6. Prepare for Next Class

1. Read these notes and practice with the hardware simulator, try to understand its assembly and walk through what other steps happen. Make notes on what you want to remember most or had the most trouble with in `hardwaresurvey.md`
2. Review and update your listing of how data moves through a program in your `abstraction.md`. Answer reflection questions below.
3. Review the commit history and git blame of a repo in browser- what must be in the .git directory for GitHub to render all of that information? (be prepared to discuss this in class)
4. Fill in the Know and Want to know columns for the new KWL chart rows below.
5. Begin your grading contract, bring questions to class Tuesday.

6.6.1. Abstraction reflection

```
1. Did you initially get stuck anywhere?
1. How does what we saw with the hardware simulators differ from how you had thought about it before?
1. Are there cases where what you previously thought was functional for what you were doing but not totally correct? Describe them.
```

6.6.2. New KWL chart rows

```
|file system | _ | _ |_ |
|bash | _ | _ | _ |
|abstraction | _ | _ | _ |
|programming languages | _ | _ | _ |
```

6.7. More Practice

1. Complete the `Try it Yourself` blocks above in your `hardwaresurvey.md`.
2. Expand on your update to `abstraction.md`: Can you reconcile different ways you have seen memory before?

6.8. Questions After Class

6.8.1. Is assembly then converted to binary and if so, why isn't the code translated straight to binary instead of from code to assembly to binary?

> Yes, assembly is converted to binary in order to write it to the hardware. The Emulator that we used today showed the numbers in RAM and the instructions both in higher level representations than binary. The advantage to assembly is that it is hardware independent and human readable. It is low level and limited to what the hardware *can* do, but it is a version of that that can be run on different hardware.

6.8.2. What does A mean in CPU Emulator?

6.8.3. When the prepare for next class says things like "organize your thoughts on …" is it expected that we make a .md file somewhere on GitHub to show this work?

> For things I expect to review, the instructions will tell you where it goes. Other things, we will bring up in a future class, so you should have it handy, but you can put it in a repo or a private set of notes.

7. What actually *is* git?

7.1. Grading Contract Q & A

Grading contract information is added to the syllabus

and the FAQ section

7.2. Git is a File System with a Version Control user interface

git is fundamentally a content-addressable filesystem with a VCS user interface written on top of it.

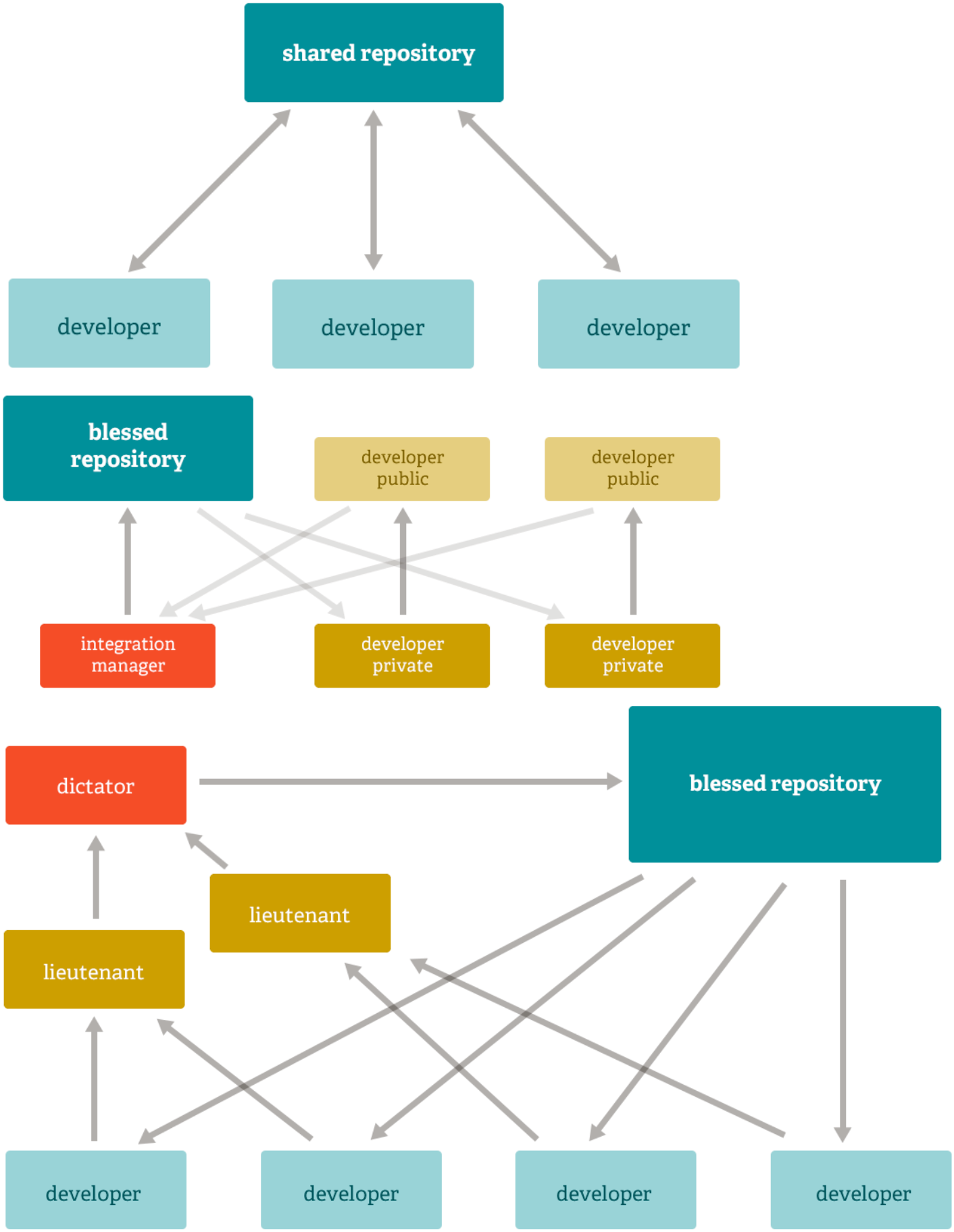
Porcelain: the user friendly VCS

Plumbing: the internal workings- a toolkit for a VCS

We have so far used git as a version control system. A version control system, in general, will have operations like commit, push, pull, clone. These may work differently under the hood or be called different things, but those are what something needs to have in order to

7.3. Git is distributed

Git can have different workflows



7.4. What are the parts of git?

and we're going to start by examinging our familiar github inclass repo

```
cd path/to/sysinclass/github-in-class-brownsarahm/
ls -a
```

```
.               .github         README.md       b_file          setup.py
..              CONTRIBUTING.md a_file          docs            test_file.md
.git            LICENSE.md      about.md        mymodule        tests
```

We are going to look inside the git folder

```
cd .git
ls
```

```
COMMIT_EDITMSG  description     info            packed-refs
HEAD            hooks           logs            refs
config          index           objects
```

The most important parts:

name	type	purpose
objects	directory	the content for your database
refs	directory	pointers into commit objects in that data (branches, tags, remotes and more)
HEAD	file	points to the branch you currently have checked out
index	file	stores your staging area information.

7.4.1. Git HEAD

We can look at the head file

```
cat HEAD
```

we see

```
ref: refs/heads/main
```

This tells us where in the git history the current status of the repository is.
This is what git uses when we call `git status`. It does more after reading that file, but that is the first thing it does.

```
cd ..
git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        a_file
        b_file
        test_file.md

nothing added to commit but untracked files present (use "git add" to track)
```

Note that the current branch is main and the HEAD file has a path to a file named main in the `refs` directory.

7.4.2. Git Refs

We can loook at that

```
cat .git/refs/heads/main
```

we see the most recent commit hash.

```
cea6a93d576ecd042823fca24553a58a6cd6565b
```

We can verify this with `git log`

when we run this it opens the log interactively, so we see something like:

```
commit cea6a93d576ecd042823fca24553a58a6cd6565b (HEAD -> main, origin/main, origin/HEAD)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 16:42:12 2022 -0500

    try to prevernt repeated running
```

In parenthesis that is what branches are pointed to that commit. Use enter/return to scroll and press `q` to exit.

> **Try it Yourself**
>
> use git log to draw a map that shows where the different branches are relative to one another

```
cd ..
cd refs/
```

```
ls
heads   remotes tags
```

```
cd heads/
ls
main    reset   test
```

this has one directory for each branch. we can confirm this with `git branch` at the top level.

```
cd ..
ls remotes/
```

```
origin
```

```
cd remotes/origin/
ls
```

```
HEAD    main    reset
```

```
(base) brownsarahm@origin $ cat HEAD
ref: refs/remotes/origin/main
(base) brownsarahm@origin $ cd main
-bash: cd: main: Not a directory
(base) brownsarahm@origin $ cat main
c15cf43b6807e172aaba7cf3b57adc7214b91082
(base) brownsarahm@origin $ cd ..
(base) brownsarahm@remotes $ cd ..
cd ..
cat config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/introcompsys/github-in-class-brownsarahm.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[branch "reset"]
        remote = origin
        merge = refs/heads/reset
ls
COMMIT_EDITMSG  config          info            refs
FETCH_HEAD      description     logs
HEAD            hooks           objects
ORIG_HEAD       index           packed-refs
cat ORIG_HEAD
c15cf43b6807e172aaba7cf3b57adc7214b91082
pwd
/Users/brownsarahm/Documents/sysinclass/github-in-class-brownsarahm/.git
cd ../../
pw
```

7.4.3. Git Config and branch naming

```
cd .git
cat config
```

and we see

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/introcompsys/github-in-class-brownsarahm.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[branch "reset"]
        remote = origin
        merge = refs/heads/reset
```

This file tracks the different relationships between your local copy and remots that it knows. This repository only knows one remote, named origin, with a url on GitHub. A git repo can have multiple remotes, each with its own name and url.

it also maps each local branch to its corresponding origin and the local place you would merge to when you pull from that remote branch.

> **Warning**
>
> I remoeved looking at the index here, we're going to come back to it with more time to inspect it more carefully on Thursday

7.4.4. Git Objects

```
cd objects/
ls
```

```
0c      35      55      87      b6      c1      pack
17      45      79      b5      b8      info
```

7.5. Starting a Git Repository from Scratch

We'll create clear repo at the top levle of our inclass directory so that it is not inside another repo

```
cd ..
pwd
```

```
/path/to/Documents/sysinclass
```

then ceate the repo with

```
git init test
```

> **Note**
>
> official statement on git branch naming

GitHub moved a little faster and used a repo to share info about their process
this change was complex and spurred a lot of discussion

```
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/brownsarahm/Documents/sysinclass/test/.git/
```

```
git branch -m main
fatal: not a git repository (or any of the parent directories): .git
```

```
cd test
git branch -m main
```

and we can confirm it works with

```
git status
```

to see that it is now on branch main.

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
ls
```

7.6. Prepare for next Class

1. review the notes and ensure that you have a new, empty repository named test with its branch renamed to main from master.
2. Add the following to your kwl:

```
|git workflows | _ | _ | _ |
| git branches | _ | _ | _ |
| bash redirects | _ | _ | _ |
```

3. Practice with git log and redirects to write the commit history for your kwl chart to a file gitlog.txt and commit that file to your repo.

7.7. More Practice

1. ¸ Read about different workflows in git and add responses to the below in a `workflows.md` in your kwl repo. Git Book atlassian Docs
2. Contribute either a glossary term, cheatsheet item, additional resource/reference, or history sidebar to the course website.

```
## Workflow Reflection

1. What advantages might it provide that git can be used with different workflows?
1. Which workflow do you think you would like to work with best and why?
1. Describe a scenario that might make it better for the whole team to use a
workflow other than the one you prefer.
```

7.8. Questions after class

7.8.1. when should the grading contract be turned in?

First draft is due Thursday, Feb 17 at 4pm as in the README

7.8.2. Can you create multiple remotes that have the same name?

no each remote has to have a different name, like each variable in a program has to be unique if you try to create a new remote with the same name it would overwrite the old one

7.8.3. what does it mean when a branch is both x commits ahead of main, but also y commits behind?

that means that there have been y commits to main that are not included on the other branch **and** that there are x commits to the other branch that are not on main.

7.8.4. Should we be working with Git entirely from the terminal for classwork or is it our preference?

In class, we will use the terminal because it is the most consistent and transparent. For other work, you can edit it however you like. You can work in browser, with the GitHub desktop client, using the GitHub CLI, or your favorite IDE's tools. I recommend giving the terminal a serious try because it will have all of the features of git and other tools may not, but your preference is fine. It could actually be really good for your general understanding to practice with git at least two ways.

7.8.5. Will we be using the github cli to publish our repo to github?

We are going to mostly use the base git CLI because it is more general and usable in more contexts (for example, your remote could be on GitLab or BitBucket or an internal private server)

7.8.6. Questions we will answer in the next couple of classes

1. What is the purpose of the index file?

2. How do you add your local repo to github?
3. What would happen if we would change this hexadecimal code which is in the files in .git?
4. Are any more efficient ways to navigate through repositories and git files

7.9. Resources

- git docs
- git book this includes other spoken languages as well if that is helpful for you.

8. How does git work?

8.1. Review

How can you write the history of a repo to a file?

We'll do this inside our in-class repo.

```
cd github-in-class-brownsarahm/
```

Recall, the `git log` command

```
git log
```

displays it in a text editor:

```
commit cea6a93d576ecd042823fca24553a58a6cd6565b (HEAD -> main, origin/main, origin/HEAD)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 16:42:12 2022 -0500

    try to prevernt repeated running

commit c15cf43b6807e172aaba7cf3b57adc7214b91082 (test)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 13:38:42 2022 -0500

    insclass 2-3

commit 17320fc6f26806eb7d1ffc62c23b3bf1361b58b2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 13:32:46 2022 -0500

    complete about closes #2

commit b81cf1525e96782e868e96a20eacf6eb26e882b7
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 13:20:05 2022 -0500
```

use the `q` key to exit

We can use a redirect `>` to send that to a file instead of to where it normally goes.

```
git log > gitlog.txt
```

We can confirm this is the same as we saw before with `cat`

```
cat gitlog.txt
```

```
commit cea6a93d576ecd042823fca24553a58a6cd6565b
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 16:42:12 2022 -0500

    try to prevernt repeated running

commit c15cf43b6807e172aaba7cf3b57adc7214b91082
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 13:38:42 2022 -0500

    insclass 2-3

commit 17320fc6f26806eb7d1ffc62c23b3bf1361b58b2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 13:32:46 2022 -0500

    complete about closes #2

commit b81cf1525e96782e868e96a20eacf6eb26e882b7
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 13:20:05 2022 -0500

    create empty about

commit f707186cd978072d2888b0d2112023b5618b6414
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 12:52:28 2022 -0500

    create readme closes #3

commit 611180dd89ffd8977dd9e5c502bcd4147c4980be
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Tue Feb 1 17:45:49 2022 +0000

    Setting up GitHub Classroom Feedback

commit 3fbd9ee19ff64241b102fa61f279e9c4683d0e69
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Tue Feb 1 17:45:49 2022 +0000

    GitHub Classroom Feedback

commit 270a43daebe1ba7bb719f141b327ca35e8e187ad
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Tue Feb 1 17:45:47 2022 +0000

    Initial commit
```

8.2. Review: Anatomy of git

Recall that the `HEAD` file contains a pointer to the place in the git database that matches the current status of the directory

```
cat .git/HEAD
```

in this case it points to the head ref called main

```
ref: refs/heads/main
```

This is what `git status` uses, we can see that using our first git “plumbing” command [1]

```
git cat-file -p refs/heads/main
```

> **Note**
>
> note that it’s sort of like the bash command cat, but it *only* works on git *objects* the HEAD file is a plain text file that we can view with `cat`

Before we look at the output of this, let’s examine the command:

- git cat-file: displays git content
- -p (pretty print) option figure out the type of content and display it appropriately

```
tree 4cbe76bea90531a07dbc8cc413de26f39994478b
parent c15cf43b6807e172aaba7cf3b57adc7214b91082
author Sarah Brown <brownsarahm@uri.edu> 1643924532 -0500
committer GitHub <noreply@github.com> 1643924532 -0500
gpgsig -----BEGIN PGP SIGNATURE-----

 wsBcBAABCAAQBQJh/Ew0CRBK7hj4Ov3rIwAATFIIAAeEB38i/5hNr1OUfYMJ/1PA
 RKdcQYspkvi70ISXpkLTEvTMwkFfmJO6Y12QqKdy7WJHIeD0qRsigmOv9oUCv2LJ
 YIdd1d6m9gfe6lwrxh5QG3itURfXyhVIff0j8YHYzq0HN7gAjw+s6UHGGjJiR7u4
 Jq01ekrQJ/OrXWmQnr6UWJNqwBx/7rSgeh0yWpx+5f8z+dFp5N57nk7MGZE04YBn
 huJG6lDPRxTHifYwDx1Ib5C0BvrLKQO0j9TvI/86LhcFSyoDBG3/diHokrQO5bdU
 okm+JsNZAx4prZebZ8eKcMEu5YiAX0wllQxrAEBKEUCc3EU2W4MMRFeiCOY3UNI=
 =oLlf
 -----END PGP SIGNATURE-----


try to prevernt repeated running
```

This object has the hash for the tree that this commit is in, it also has the hash of the parent commit to this commit, then author information and the last commit message

8.3. Git Plumbing in a Fresh repo

Recall we used `git init test` to create a new repo on Tuesday, let's got back to that.

```
cd ../test/
```

We can use `git status` to confirm that it is completely blank

```
git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

The directory is also empty

```
ls -a
```

Except for the `.git` database

```
.       ..      .git
```

And recall the different files.

```
ls .git/
HEAD            description     info            refs
config          hooks           objects
```

> **Try it yourself**
>
> Try to remember what each of those files/directories does, make a table and try to write each one's role. After, check your answers using Tuesday's notes

8.4. Git Objects

we noted above that the `git cat-file` only works to git objects. Lets see what objects there are. Today we will see three types today:

- blob objects: the content of your files (data)
- tree objects: stores file names and groups files together (organization)
- Commit Objects: stores information about the sha values of the snapshots

First, lets examine the `objects` directory. This time, instead of using `ls` we'll use the `bash` command `find`. This pattern matches and looks for files or directories and lists the results

```
find .git/objects/
```

We see that it finds, the `object` directory itself and two subdirectories.

```
.git/objects/
.git/objects//pack
.git/objects//info
```

Let's look at another option of this command, the `-type` option filters the results and only returns the ones of the desired type. We'll look at only files using `f`

```
find .git/objects/ -type f
```

In this empty repository, there are no files.

8.4.1. Hashing

Let's create an object. We have mentioned breifely before that git stores data by hashing it, so the plumbing command `git hash-object` does exactly that.

```
echo 'text content' | git hash-object -w --stdin
```

Let's examine this command:

- `git hash-object` by default hashes the return the unique key to that particular content
- the `-w` option then tells git to alsow write that object to the database
- the `--stdin` option tells `git hash-object` to get the content to be processed from stdin instead of a file
- the `|` is called a pipe (what we saw before was a redirect) it pipes a *process* output into the next command
- echo would write to stdout, with the pipe it passes that to stdin for the `git hash-object` to use

> **Note**
>
> the key is *unique* in that there is a one to one mapping from any particular content to a single key. However, as we saw in class, if we all do the same content, we all get the key same key back

and we see it returns the unique key to us:

```
c182a93374d6b18a87e1f1e8a9a18812639c58c8
```

We can then view the file that was added to the database with `git cat-file` again. Recall the `-p` option, uses information about the file type to display it in an appropriate manner.

```
git cat-file -p c182a93374d6b18a87e1f1e8a9a18812639c58c8
text content
```

So far, we added "text content" to the git database, but nothing in the directory:

```
ls
```

shows us that its an empty directory.

Lets create a file and add that to our git database. We will create the file with echo and a redirect:

```
echo 'version 1' > test.txt
```

then we can hash it and write to the database from the file this time.

```
git hash-object -w test.txt
83baae61804e65cc73a7201a7252750c76066a30
```

we can list the directory so and note that more directories have been made:

```
ls .git/objects/
83      c1      info    pack
```

Even better, we can use the `find` command filtered by type that we saw earlier.

```
find .git/objects/ -type f
.git/objects//c1/82a93374d6b18a87e1f1e8a9a18812639c58c8
.git/objects//83/baae61804e65cc73a7201a7252750c76066a30
```

We see that this created an additional file for each of the two has objects that we have created.

Let's append more text to the file, recall two `>>` redirects to append instead of overwrite.

```
echo 'version 2' >> test.txt
```

We can confirm this worked as we expected:

```
cat test.txt
version 1
version 2
```

And then has the file

```
git hash-object -w test.txt
```

and view the hashed content using the key that was returned.

```
git cat-file -p 0c1e7391ca4e59584f8b773ecdbbb9467eba1547
version 1
version 2
```

So far, we have been adding a new hash of the whole file each time. We will see later that git *can* be more efficient than this because this would begin to take a lot of space very quickly.

8.4.2. Tree Objects

The next type of object to inspect is the tree, since the tree is a git object, we can use `git cat-file` again.

```
git cat-file -p main^{tree}
```

but this test repo that we are working with does not have a tree yet, so it

```
fatal: Not a valid object name main^{tree}
```

We can confirm this using git status

```
git status
```

```
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

> **Important**
>
> This syntax works in `bash` but has common problems in other shells:
> - CMD on Windows: the `^` character is used for escaping, so you have to double it:`git cat-file -p main^^{tree}`.
> - PowerShell: parameters using `{}` characters have to be quoted to parse correctly, so use: `git cat-file -p 'master^{tree}'`.
> - ZSH: the ^ character is used for globbing, so enclose the whole expression in quotes: `git cat-file -p "master^{tree}"`.

In a more complete repo, we can view the tree:

```
git cat-file -p main^{tree}
```

```
040000 tree c91892d93170077fea81f1d8009c4ccc91af1c29    .github
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    CONTRIBUTING.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    LICENSE.md
100644 blob 7987a001e70d28376129bfe0538f98f9aa281a55    README.md
100644 blob b5f4fcb8f875fe33781256ebf6fdde7547188d6f    about.md
040000 tree 55651ac0763db741988f02a45842aa020a77c14d    docs
040000 tree 3581dc38ca3929efcbbc6f7ce510ded2c7dd7309    mymodule
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    setup.py
040000 tree 45fcb1dd311e5e45af759cb3627dca5f47f58f04    tests
```

We see a blob for each file and a tree for each directory. The blob objects refer to the last hashed version of the file added to this branch (main)

```
cat HEAD
ref: refs/heads/main
```

8.5. Creating a Commit manually

> **Warning**
>
> this is revised relative to in class

In class we tried to add directly to the commit tree:

```
echo 'first commit' | git commit-tree c182a9
```

but this failed:

```
fatal: c182a93374d6b18a87e1f1e8a9a18812639c58c8 is not a valid 'tree' object
```

because we had skipped a step. We need to create the tree first.

You use this command to artificially add the earlier version of the test.txt file to a new staging area.

```
git update-index --add --cacheinfo 100644 \
  83baae61804e65cc73a7201a7252750c76066a30 test.txt
```

the `\` allows us to continue typing on a second visual line in one long command. this command puts the hashed object 83baae with file named `test.txt`

- this the plumbing command `git update-index` updates (or in this case creates an index, the staging area of our repository)
- the `--add` option is because the file doesn't yet exist in your staging area (you don't even have a staging area set up yet)
- `--cacheinfo` because the file you're adding isn't in your directory but is in your database. Then, you specify the mode, SHA-1, and filename

n this case, you're specifying a mode of 100644, which means it's a normal file.

```
git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

```
git add test.txt
```

```
git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.txt
```

> **Note**
>
> We tried viewing the index but it did not work. the documentation shows the specification for the index, it is not readable

```
git commit -m 'first commit'
[main (root-commit) 2948028] first commit
 1 file changed, 2 insertions(+)
 create mode 100644 test.txt
```

```
find .git/objects/ -type f
.git/objects//0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects//c1/82a93374d6b18a87e1f1e8a9a18812639c58c8
.git/objects//29/480288d80e3850d6bf7ae170bd3bea5b3164c7
.git/objects//83/baae61804e65cc73a7201a7252750c76066a30
.git/objects//25/8231c1cee8048eef3a8057cfbdab76261277c6
```

```
git cat-file -p main^{tree}
100644 blob 0c1e7391ca4e59584f8b773ecdbbb9467eba1547    test.txt
```

```
git cat-file -p 2948028
tree 258231c1cee8048eef3a8057cfbdab76261277c6
author Sarah M Brown <brownsarahm@uri.edu> 1645121714 -0500
committer Sarah M Brown <brownsarahm@uri.edu> 1645121714 -0500

first commit
```

```
git cat-file -p 258231
100644 blob 0c1e7391ca4e59584f8b773ecdbbb9467eba1547    test.txt
```

8.6. How is git efficient?

```
git gc
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
ls
test.txt
```

```
find .git/objects -type f
.git/objects/c1/82a93374d6b18a87e1f1e8a9a18812639c58c8
.git/objects/pack/pack-07b0c08cc0268d55d02ea62ea880c61f810956d8.idx
.git/objects/pack/pack-07b0c08cc0268d55d02ea62ea880c61f810956d8.pack
.git/objects/info/commit-graph
.git/objects/info/packs
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

```
git verify-pack
usage: git verify-pack [-v | --verbose] [-s | --stat-only] <pack>...

    -v, --verbose         verbose
    -s, --stat-only       show statistics only
    --object-format <hash>
                          specify the hash algorithm to use
```

```
git verify-pack -s
usage: git verify-pack [-v | --verbose] [-s | --stat-only] <pack>...

    -v, --verbose         verbose
    -s, --stat-only       show statistics only
    --object-format <hash>
                          specify the hash algorithm to use
```

8.7. Why didn’t git cat-file work on HEAD?

We tried:

```
git cat-file -p HEAD
```

and saw an error

```
fatal: Not a valid object name HEAD
```

To confirm what it was doing wrong, we checked another repository:

```
git init test2
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/brownsarahm/Documents/sysinclass/test2/.git/
```

```
git cat-file -p HEAD
```

```
fatal: not a git repository (or any of the parent directories): .git
```

```
cd test2
```

```
git cat-file -p HEAD
```

```
fatal: Not a valid object name HEAD
```

We confirmed that this does not work.

This was because I made a mistake, the HEAD file does not need `git cat-file` it is a plain text file, not a git blob (type) object.

8.7.1. Could the lack of pushing be why?

```
cat .git/config
```

lets compare the results of this across two repos first for the small test repo:

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
```

and for github-inclass:

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/introcompsys/github-in-class-brownsarahm.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[branch "reset"]
        remote = origin
        merge = refs/heads/reset
```

This is all of the difference that is caused by the lack or configuring a remote.

We can view the heads by listing that directory

```
ls .git/refs/heads
```

```
main    reset   test
```

8.8. Prepare for next Class

1. Review the notes
2. For the core "Porcelain" git commands we have used (add, commit), make a table of which git plumbing commands they use in `gitplumbing.md` in your KWL repo
3. In a `gitunderstanding.md` list 3-5 items from the following categories (1) things you have had trouble with in git in the past and how they relate to your new understandign (b) things that your understanding has changed based on today's class (c) things about git you still have questions about
4. Make notes on *how* you use IDEs for the next week or so using the template `idethoughts.md` file in the course notes.

8.9. More Practice

1. Add to your `gitplumbing.md` file explanations of the main git operations we have seen (add, commit, push) in your own words in a way that will either help you remember or behow you would explain it to someone else at a high level. This might be analogies or explanations using other progrmaming concepts or concepts from a hobby. Add this under a subheading `##` with a descriptive title (for example "Git In terms of ")
2. For one thing your understanding changed or an open question you, look up or experiment to find the answer

> **Further Reading**
>
> The goal of this exercise is to take an ethnographic approach to understanding the IDE(s) you use most often. We will combine this with a more formal study of them soon. Approaching a topic through multiple lenses can help you understand it better and presenting you, as a group, with multiple ways is a strategy of mine to help make sure that every one of you finds *at least* one way that works for you.
> More on ethnography in CS

```
# IDE Thoughts

## Actions Accomplished
<!-- list what things you do: run code/ edit code/ create new files/ etc; no need to comment on what the code you write does -
->


## Features Used
<!-- list features of it that you use, like a file explorer, debugger, etc -->
```

8.10. Questions After Class

8.10.1. when does the grade-free zone end?

It has ended, starting with the feedback given today, there will be grading implications, but remember there are flexibility systems with most of it.

8.10.2. Are there other uses for the hashes besides identifying git files in our repositories

for these particular hashes, no, they are the keys that mathc to values of items in our git database, but we can do other things with other hashes.

8.10.3. what does print pretty / -p mean?

this option of the git cat-file command displays the contents of a blob in a human readable format.

8.10.4. will we ever need to use the plumbing tools in a bind?

Some of them, yes! For example, today, it was helpful to use the plumbing command `git update-ref` to create a branch from a particular past commit. They could also help you to view a commit in greater detail and some other beyond what we saw today could help you edit a commit or alter the “history” of your repo if you need to.

The porcelain commands will get you through your day to day everything is going right and the most common mistakes.

8.10.5. Questions addressed above integrated into the narrative

• I am still confused about the idea of the objects and what they do exactly/their purpose

• so all of the plumbing commands we used today are all the parts that are wrapped up in the processes like adding/committing/pushing things to GitHub?

8.10.6. Questions to practice with

- What if we would change the file again (added 3rd line) and committed again. How many hash objects would be created? What connection would there be between the commits?

8.10.7. Questions we will come back to next

- how are hash number formed?
- From init, first commit, to first push, what is the exact sequence of events that happens under the hood of git?

---

[1] low level git commands that allow the user to access the inner workings of git.

9. How do hashes work in git?

In this class we will cover:

- what type of hashes are used in git
- what a hash is
- number systems

9.1. Git Plumbing Q&A

Plumbing commands work with the .git directory as a file system directly. Porcelain commands provide higher level interactions with git as a version control system.

9.2. What *is* a hash?

- a hash is a fixed size value that can be used to represent data of arbitrary sizes
- the *output* of a hashing function
- often fixed to a hash table

A hasing function could be really simple, to read off a hash table, or it can be more complex.

For example:

Hash	content
0	Success
1	Failure

If we want to represent the status of a program running it has two possible outcomes: success or failure. We can use the following hash table and a function that takes in the content and returns the corresponding hash. Then we could pass around the 0 and 1 as a single bit of information that corresponds to the outcomes.

This lookup table hash works here.

In a more complex scenario, imagine trying to hash all of the new terms you learn in class. A table would be hard for this, because until you have seen them all, you do not know how many there will be. A more effective way to hash this, is to derive a *hashing function* that is a general strategy.

9.3. When does hashing occur in git?

In git we hash both the content directly to store it in the database (.git) directory and the commit information.

Recall, when we were working in our toy repo we created an empty repository and then added content directly, we all got the same hash, but when we used git commit our commits had different hashes because we have different names and made the commits at different seconds. We also saw that *two* entries were created in the `.git` directory for the commit.

In git, 40 characters that uniquely represent either the content or a commit.

Mostly, a shorter version of the commit is sufficient to be unique, so we can use those to refer to commits by just a few characters:

- minimum 4
- must be unique

> **Note**
>
> You can view commits shorter with options to `git log`
>
> ```
> git log --abbrev-commit --pretty=oneline
> ```

> **Important**
>
> git commits only need to be unique is **per repository** the git program is not searching all git repositories when we run commands that use commits, it is looking only in the local `.git` directory. You could even have two different projects on your computer with an identical hash and that would not be a conflict because git only looks within the current directory.

9.4. What hashing function does git use?

Git uses SHA-1. See a generator

This is a Secure Hashing Algorithm that is derived from cryptography. Because it is secure, no set of mathematical options can directly decrypt an SHA-1 hash. It is designed so that any possible content that we put in it returns a unique key. It uses a combination of bit level operations on the content to produce the unique values.

The SHA-1 Algorithm hashes content into a fixed length of 160 bits. This means it can produce \(2^160\) different hashes. Which makes the probability of a collision very low.

> The number of randomly hashed objects needed to ensure a 50% probability of a single collision is about \(2^{80}\) (the formula for determining collision probability is p = (n(n-1)/2) * (1/2^160)). \(2^{80}\) is 1.2 x 1024 or 1 million billion billion. That's 1,200 times the number of grains of sand on the earth.
>
> —A SHORT NOTE ABOUT SHA-1 in the Git Documentation

This output, 160 bits (a bit is a unit of information in base 2; a 0 or 1) can be interpreted as a number and represented in diffrent ways. Since 160 characters is really long, git represents it as 40 characters in hexadecimal.

However, SHA-1 is subjec to collision attacks.

> We have broken SHA-1 in practice.
>
> …
>
> It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file.
>
> …
>
> GIT strongly relies on SHA-1 for the identification and integrity checking of all file objects and commits. It is essentially possible to create two GIT repositories with the same head commit hash and different contents, say a benign source code and a backdoored one. An attacker could potentially selectively serve either repository to targeted users. This will require attackers to compute their own collision. — shattered it

Git switched to hardended SHA-1 in response to a collision

> In that case it adjusts the SHA-1 computation to result in a safe hash. This means that it will compute the regular SHA-1 hash for files without a collision attack, but produce a special hash for files with a collision attack, where both files will have a different unpredictable hash. from

and they will change again soon

9.5. What is a number?

a mathematical object used to count, meaure and label

a mathematical object used to count, meaure and label

What types of numbers are you familiar with?

9.6. Number Representation

Numbers are a cultural artifact: We use a base 10 system most commonly becuase we count with our hands and have 10 fingers. Computers use base 2 because they are digital:on & off. The current representation system we use (0,1, 2, 3, 4,5,6,7,8,9) is called hindu-arabic.

9.6.1. Decimal

To represent larger numbers than we have digits on we have a base (10) and then.
\[10 ≡ 10*1 + 1*0\]
\[22 ≡ 10*2 + 1*2 \]
we have the ones (\(10^0) place, tens (\)10^1\() place, hundreds (\)10^2) place etc.

9.6.2. Binary

uses base 2, but the same characters. So the place values are the ones(\(2^0\)), the twos(\(2^1\)), the fours(\(2^2\)), the eights(\(2^3\)), etc.
\[10 => 2*1 + 1*0 = 2\]
so this 10 in binary is 2 in decimal
\[1001 => 8*1 + 4*0 + 2*0 + 1*1 = 9\]
Binary numbers have been discovered in ancient egyptian, chinese and Indian texts.

9.6.3. Octal

uses base 8, but the same characters. So the place values are the ones(\(8^0\)), the eights(\(8^1\)), the 64s (\(8^2\)), etc.
\[10 = > 8*1 + 1*0 = 8\]
so 10 in octal is 8 in decimal
\[401 => 64*4 + 8*0 + 1*1 = 257\]
This numbering system was popular in 6 bit and 12 bit computers, but is has origins before that. Native Americans using the Yuki Language (based in what is now California)used an octal system because they count using the spaces between fingers and speakers of the Pamean languages in Mexico count on knuckles in a closed fist. Europeans debated using decimal vs octal in the 1600-1800s for various reasons because 8 is better for math mostly. It is also found in Chinese texts dating to 1000BC.

9.6.4. Hexadecimal

uses base 16, but the same characters plus letters A-F. The letters fill in for the numbers aft 9 so A is 10, B is 11, etc. So the place values are the ones(\(16^0\)), the sixteens(\(16^1\)), the two hundred fifty sixes(\(16^2\)),etc.
\[10 = > 16*1 + 1*0 = 16\]
so 10 in hex is 16 in decimal
\[E => 1*14 = 14\]
\(CD => 16*12 + 1*13 = 205\)$

There was debate a number of different proposals for how the characters beyond 9 should be represented.

9.6.5. Roman numerals

Use different representation completely. It doesn't have places the same way. But it is still a way to represent numbers.

I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M =1000

This repesentation concatenates symbols and adds them if they are in desceding order and if a smaller digit before a larger then it is subtracted.

- III = 1 + 1 + 1= 3
- IX = 10 -1 = 9
- XL = 50 - 10 = 40

> **Note**
>
> On a lighter note: passWordle

> **Learn more**
>
> Learning more about other number systems or the history of these number systems is a good topic for a deeper exploration.

9.7. Prepare for next Class

1. review the past two classes of notes
2. find 3 more examples of using other number systems, list them in `numbers.md`
3. Read about hexpeak from Wikipedia for an overview and one additional source. In your kwl repo in `hexspeak.md` summarize it in your own words, one interesting fact from your additional source and link to the source you found. Come up with a word or two on your own.
4. (priority) Bring to class a scenario where you think git could help, but we haven't covered yet how to do it. (be prepared to post it to Prismia at the start of class)

9.8. More Practice

1. Read more about git's change from SHA-1 to SHA-256 and reflect on what you learned (questions provided)
2. (priority) In a language of your choice or pseudocode, write a short program to convert, without using libraries, between all pairs of (binary, decimal, hexidecimal) in `numbers.md`. Test your code, but include in the markdown file enclosed in three backticks so that it is a "code block" write the name of the language after the ticks like:

````
```python
# python code
```
````

9.8.1. transition questions

1. Why is the switch important?
2. Summarize one vulnerability of SHA-1.
3. What impact will the swith have on how git works?
4. If you have scripts that operate on git repos, what might you do to future proof them so that the switch won't break your code.

9.9. Questions After Class

9.9.1. Why did git use SHA-1 isntead of SHA 256 from the start?

The shorter the hash the more efficient the system is and git was 12 years old when SHA-1 was determined to be meaningfully vulnerable to collsion attack.

9.9.2. If there is ever a class cancellation or any kind of issue with the notes again, will that be emailed to us or only sent through Slack?

It will be announced on slack, but with an @channel mention, you can set your notification preferences so that you get e-mails for direct messages and @ mentions and the @channel counts as an @ mention for all members of the slack workspace (in this case our class). You coudl also opt for mobile notifications if you prefer. The goal with slack is to make it so that you all can better customize how you are notified and I can post to one place.

9.9.3. Are both git and github switchig from sha1

GitHub *uses* git, it is not an alternative implementation or a fork, so yes it will switch too. The developers at GitHub an other git hosts are among the most impacted by the change since they write code that directly interacts with git objects.

9.9.4. Are octal numbers ever used in computing?

We will see it when we talk about file permissions in unix systems.

More broadly, this would be a good topic for a deeper exploration report.

9.9.5. Questions left as an exercise

> **Warning**
>
> these are for you to look up/hypothesize about and then we will discuss with you in your KWL repo

- What are the possible dangers of git switching to a better encryption scheme immediately?

10. How can git help me?

So far we have seen git add a bunch of steps to a workflow and worked to understand what it is doing. I've *told* you tracks versions and can help you out because keeping track of versions is good.

Today I will *show* you how it can help.

we are going to work in out github-inclass repo today. Let's reveiw its status.

```
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        a_file
        b_file
        gitlog.txt
        test_file.md

nothing added to commit but untracked files present (use "git add" to track)
```

In mine, I have these extra files I don't need from testing things. How can I get rid of them since I do not want to keep them?

We see that they are in the files along with the other files that we planned to be here. Remember the "scenario" of this repo is that it has a number of mostly empty files to represent a python project.

```
ls
CONTRIBUTING.md README.md       about.md        docs            mymodule        test_file.md
LICENSE.md      a_file          b_file          gitlog.txt      setup.py        tests
```

Could it be git pull?

```
git pull
```

```
Already up to date.
```

No git pull, gets, from the origin, the most up to date version of the git database (`.git` directory) from the origin (in this case github). It does not, however do anything to untracked files.

We can remove them:

```
rm *_file
rm gitlog.txt
rm test_file.md
```

Then we can confirm the git status of the repo.

```
git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

10.1. What if I make a file that I don't want to commit?

Imagine we made a configuration file that contained some text (for example a personal key of some sort) that we need in the directory to make our code work locally, but that we do not want to push to GitHub?

```
echo "secrets" >> config.txt
```

The way to do that is to not put it into git at all, so that when we push it will not go.

As usual, before we do anything with git, we check the status

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        config.txt

nothing added to commit but untracked files present (use "git add" to track)
```

In this case, we see that git has the file here, but it is untracked.

Recall, git status has the git program check the `HEAD` file to determine what in the databaes to look at and then compares the current directory status to what the corresponding last hash contained.

Git provides a special file that it will check when we add, and then not add any files that match the files in.

```
nano .gitignore
```

We will add the following to the file:

```
config.*
```

Then write and exit from nano.

Now our status has changed:

```
git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

We see the config file is not in the untracked anymore (that means it will not be added if we do git add) but the gitignore file is.

We want to commit this file becuase we would want collaborators to share the same ignored patterns.

```
git add .
```

```
git commit -m 'ignore configs'
[main 3c46e01] ignore configs
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

Now, what is the advantage of using the pattern?

If we make another file that starts with “config”

```
echo "more secrets" >> config.yml
```

and then check the status

```
git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

git does not see it at all.

we can see though that the files are boht there in our directory on disk.

```
ls
```

```
CONTRIBUTING.md README.md       config.txt      docs            setup.py
LICENSE.md      about.md        config.yml      mymodule        tests
```

What happens if we try to explicitly add it?

```
git add config.txt
```

Let's see:

```
The following paths are ignored by one of your .gitignore files:
config.txt
hint: Use -f if you really want to add them.
hint: Turn this message off by running
hint: "git config advice.addIgnoredFile false"
```

git tries to warn us.

> **Important**
>
> In general, if something you are trying to do requires `-f` or `-hard` in git or bash (or others) do not do it unless you are **very** confident that it is the right thing to do.
>
> These options are generally only required for things are either hard or impossible to undo.

10.2. What if I break my code and commit the bad code?

Let's imagine we wrote some code that does not actually work

```
echo "bad code" >> mymodule/important_classes.py
```

but we forgot to test it, so we add

```
git add .
```

and commit it:

```
git commit -m 'best feature ever'
```

now it is in our repository

```
[main 2c1c3e2] best feature ever
 1 file changed, 1 insertion(+)
```

Now we test the code, find out that it does not work. but we remember that it worked before this code we just added. We can view the commit history to get the hash of the last commit that worked.

```
git log
commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:00:50 2022 -0500

    best feature ever

commit 3c46e01c5b29ffa2ed9a76ff4e8c8f334c0c1bf2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 12:54:45 2022 -0500

    ignore configs

commit cea6a93d576ecd042823fca24553a58a6cd6565b (origin/main, origin/HEAD)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 16:42:12 2022 -0500

    try to prevernt repeated running

commit c15cf43b6807e172aaba7cf3b57adc7214b91082 (test)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 13:38:42 2022 -0500

    insclass 2-3

commit 17320fc6f26806eb7d1ffc62c23b3bf1361b58b2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Feb 1 13:32:46 2022 -0500
```

and then we can look at the difference between that version of `important_classes.py` and the current version:

```
git diff 3c46e0 important_classes.py
```

Git diff is what GitHub uses on the PR files changed tab. it takes the commit reference first and then the file to look at just the one files changes.

```
diff --git a/mymodule/important_classes.py b/mymodule/important_classes.py
index e69de29..90baf46 100644
--- a/mymodule/important_classes.py
+++ b/mymodule/important_classes.py
@@ -0,0 +1 @@
+bad code
```

From this, we can confirm that the only change in the most recent commit is the bad code, there is not any other good code we want to keep.

We can look back our git log above to get the hash for that last commit and we can use git revert to “undo” it.

```
git revert 2c1c3e
```

Then git asks us to confirm a commit message, and we get output like this, which looks like after a commit.

```
[main 8bb2df9] Revert "best feature ever"
 1 file changed, 1 deletion(-)
```

To see how git revert works lets look at our commit history again.

```
git log
```

```
commit 8bb2df91d371ab3f9245d49aee5bd0e5f95a9294 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:09:56 2022 -0500

    Revert "best feature ever"

    This reverts commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4.

commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:00:50 2022 -0500

    best feature ever

commit 3c46e01c5b29ffa2ed9a76ff4e8c8f334c0c1bf2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 12:54:45 2022 -0500

    ignore configs

commit cea6a93d576ecd042823fca24553a58a6cd6565b (origin/main, origin/HEAD)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 16:42:12 2022 -0500

    try to prevernt repeated running

commit c15cf43b6807e172aaba7cf3b57adc7214b91082 (test)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 3 13:38:42 2022 -0500

    insclass 2-3

...
```

git did not go back in history in this case, it looked at what changes were applied in the commit we were reverting, applied the inverse of them to the current place the head points and added a new commit for that.

To illustrate this more clearly, lets make two commits. First we'll commit some code we will later realize is not good ot helper functions and then something good to alternative classes.

```
echo "heleper code" >> helper_functions.py
git add .
git commit -m 'bad code'
```

we see the message for that

```
[main e9c132f] bad code
 1 file changed, 1 insertion(+)
```

and the second change

```
echo "something good" >> alternative_classes.py
git add .
git commit -m 'some code'
```

```
[main a224c42] some code
 1 file changed, 1 insertion(+)
```

Now we view the commit history again and the code we want to change is one commit back.

```
git log
commit a224c42a9fd5ba19b55d1e5e4f34c411c8d519f3 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:47 2022 -0500

    some code

commit e9c132f6922a4bfae5b21793a9982fe837db6643
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:17 2022 -0500

    bad code

commit 8bb2df91d371ab3f9245d49aee5bd0e5f95a9294
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:09:56 2022 -0500

    Revert "best feature ever"

    This reverts commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4.

commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:00:50 2022 -0500

    best feature ever

commit 3c46e01c5b29ffa2ed9a76ff4e8c8f334c0c1bf2
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 12:54:45 2022 -0500

    ignore configs
```

We can use revert again

```
git revert e9c132
```

it behaves just as before, having us write a commit message and then commiting the “new” changes

```
[main b6437b4] Revert "bad code"
 1 file changed, 1 deletion(-)
```

Now when we view the commit history we see the bad code, the commit we wanted to keep after it and hten the reverted bad code.

```
git log
commit b6437b4174f69d79e769e296c5e0db5f38c78a55 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:14:58 2022 -0500

    Revert "bad code"

    This reverts commit e9c132f6922a4bfae5b21793a9982fe837db6643.

commit a224c42a9fd5ba19b55d1e5e4f34c411c8d519f3
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:47 2022 -0500

    some code

commit e9c132f6922a4bfae5b21793a9982fe837db6643
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:17 2022 -0500

    bad code

commit 8bb2df91d371ab3f9245d49aee5bd0e5f95a9294
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:09:56 2022 -0500

    Revert "best feature ever"

    This reverts commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4.

commit 2c1c3e2d7bebdbd238e517e07142151cdf5256f4
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:00:50 2022 -0500
```

Important

What does this functionality tell us about how we should commit?

we want to commit:

- for each “thought” that is cohesive and is a unit we might want to undo
- we want to use descriptive commit messages so that we can use them to pick out what to undo

10.3. What if I try a bunch of stuff and it doesn’t work, but I notice before I commit?

Imagine, you tried out a new idea:

```
echo "test idea" >> mymodule/important_classes.py
```

and then you test and it does not work. You want to put your working directory back to match the last commit in order to start from scratch with a new idea.

```
git status
```

We can see that our new file is not staged for commit and the git status tells us how to put it back.

```
On branch main
Your branch is ahead of 'origin/main' by 6 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   mymodule/important_classes.py

no changes added to commit (use "git add" and/or "git commit -a")
```

We can use `git restore` with the file name (path) of what to restore

```
git restore mymodule/important_classes.py
```

Now if we try something new again

```
echo "test idea again" >> mymodule/important_classes.py
```

10.4. How can git help me remember what I'm working on?

Imagine you worked some then walked away and forget what you were working on.
A diff can help you see what changes you have made since your las commit.

```
git diff mymodule/important_classes.py
diff --git a/mymodule/important_classes.py b/mymodule/important_classes.py
index e69de29..a8343bc 100644
--- a/mymodule/important_classes.py
+++ b/mymodule/important_classes.py
@@ -0,0 +1 @@
+test idea again
```

Now if we add the file, to commit.

```
git add .
```

If you are about to commit and forget what work you are about to commit or have trouble thinking of what to write as your commit message, you can use `git diff --staged` to see exactly what changes you are about to add.

```
git diff --staged
```

this outputs a diff between the index and the last commit.

```
diff --git a/mymodule/important_classes.py b/mymodule/important_classes.py
index e69de29..a8343bc 100644
--- a/mymodule/important_classes.py
+++ b/mymodule/important_classes.py
@@ -0,0 +1 @@
+test idea again
```

Note that it shows you what you are comparing `e69de29` to `a8343b` and lists the files and both a summary `@@ -0,0 +1 @@`

and the actual line by line changes.

This can be more useful after we ahve made more changes after we staged some.

```
echo "doc new feature" >> README.md
```

and check the diff again

```
git diff --staged
```

we get the same output

```
diff --git a/mymodule/important_classes.py b/mymodule/important_classes.py
index e69de29..a8343bc 100644
--- a/mymodule/important_classes.py
+++ b/mymodule/important_classes.py
@@ -0,0 +1 @@
+test idea again
```

Thisis the same becuase it comares the *staging* area to the last commmit, not the current directory to the last commit.

```
git status
```

This shows that only the important_classes file is staged, not the readme chagnes.

```
On branch main
Your branch is ahead of 'origin/main' by 6 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mymodule/important_classes.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
```

10.5. How can update my feature branch when main gets updated?

Imagine you are working on a new feature for a the project, so you create a new branch

```
git checkout -b new_feature
Switched to a new branch 'new_feature'
```

and add your work.

```
echo "new feature" >> mymodule/important_classes.py
```

```
git status
On branch new_feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   mymodule/important_classes.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
        modified:   mymodule/important_classes.py
```

and we will commit the code here.

```
git add mymodule/important_classes.py
git commit -m 'new feature'
```

```
[new_feature e7a92d2] new feature
 1 file changed, 2 insertions(+)
```

Then you remember you had done work on the README that was supposed to be on main already, so we go back there and commit that file. Imagine this work is a bug fix that is unrelated to new feature and you want users and your co-workers to have before you finish the new feature.

```
git checkout main
M       README.md
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 6 commits.
  (use "git push" to publish your local commits)
```

```
git status
On branch main
Your branch is ahead of 'origin/main' by 6 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

and commit the readme here.

```
git add .
```

```
git commit -m 'update readme'
[main 07dbc9e] update readme
 1 file changed, 1 insertion(+)
```

We can view commit history to see that these two branches have similar, but not the same history.

```
git log
commit 07dbc9e484b92cdbc7b055883fb5e1bfaee645f1 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:27:33 2022 -0500

    update readme

commit b6437b4174f69d79e769e296c5e0db5f38c78a55
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:14:58 2022 -0500

    Revert "good code"

    This reverts commit e9c132f6922a4bfae5b21793a9982fe837db6643.

commit a224c42a9fd5ba19b55d1e5e4f34c411c8d519f3
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:47 2022 -0500

    some code

commit e9c132f6922a4bfae5b21793a9982fe837db6643
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:17 2022 -0500

    good code

commit 8bb2df91d371ab3f9245d49aee5bd0e5f95a9294
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:09:56 2022 -0500

    Revert "best feature ever"
```

then to the other branch

```
git checkout new_feature
```

```
Switched to branch 'new_feature'
```

and commit history again

```
git log
```

```
commit e7a92d24de2d70c6122e28b55cd9066b504ae8c8 (HEAD -> new_feature)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:26:40 2022 -0500

    new feature

commit b6437b4174f69d79e769e296c5e0db5f38c78a55
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:14:58 2022 -0500

    Revert "good code"

    This reverts commit e9c132f6922a4bfae5b21793a9982fe837db6643.

commit a224c42a9fd5ba19b55d1e5e4f34c411c8d519f3
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:47 2022 -0500

    some code

commit e9c132f6922a4bfae5b21793a9982fe837db6643
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:12:17 2022 -0500

    good code

commit 8bb2df91d371ab3f9245d49aee5bd0e5f95a9294
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Feb 24 13:09:56 2022 -0500

    Revert "best feature ever"
```

Now we want to get the changes from main into the new feature branch because those fixes impact our ability to test the new feature to see if it works right.

We can update the new_feature branch with rebase.

```
git rebase main
Successfully rebased and updated refs/heads/new_feature.
```

Rebase gets the current status of the main branch (which has a mostly shared commit history with `new_feature` and then applies all of the commits that are on `new_feature` but not `main` on top of the most recent version of main.

We can see than now the history of new_feature is everything on main +1 one commit, where before they had shared except the last commit, where each one had a different most recent commit.

Now, if we're done with the new feature and it works, we can merge it into main.

```
git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 7 commits.
  (use "git push" to publish your local commits)
```

On main we have no current thigns to commit:

```
git status
On branch main
Your branch is ahead of 'origin/main' by 7 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Git `merge` allows us to apply the commits from new_feature to main.

```
git merge new_feature
Updating 07dbc9e..19d30d1
Fast-forward
 mymodule/important_classes.py | 2 ++
 1 file changed, 2 insertions(+)
```

This applies the commits to new_feature that are not on main to the end of main.

```
git status
On branch main
Your branch is ahead of 'origin/main' by 8 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

10.6. What if the content diverges

Let's make a new branch

```
git checkout -b hotfix
Switched to a new branch 'hotfix'
```

Then switch back to main

```
git checkout main
Switched to branch 'main'
```

and add more content to our main branch

```
echo "bold commit to main" >> mymodule/important_classes.py
echo "slow thoughtful fix" >> mymodule/important_classes.py
```

We can look at the version of the file we have here

```
cat mymodule/important_classes.py
```

we have the old content and our new two lines

```
test idea again
new feature
bold commit to main
slower more thoughtful fix
```

and commit.

```
git add .
git commit -m "working project"
[main b832b2a] working project
 1 file changed, 2 insertions(+)
```

Now we'll go back to the feature branch

```
git checkout -b hotfix
Switched to branch 'hotfix'
```

and look at what's there.

```
cat mymodule/important_classes.py
```

we don't have those two new lines.

```
test idea again
new feature
```

Now we fix the old bug that's not the new things. (maybe this was a different person working on this branch)

```
echo "fix older bug" >> mymodule/important_classes.py
```

and commit that

```
git add .
```

```
git commit -m 'fix other bug"
> '
[hotfix 3664aa0] fix other bug"
 1 file changed, 1 insertion(+)
```

Now we can compare the two versions.

```
cat mymodule/important_classes.py
```

we have 3 lines on the hotfix branch

```
test idea again
new feature
fix older bug
```

```
git checkout main
```

```
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 9 commits.
  (use "git push" to publish your local commits)
```

```
cat mymodule/important_classes.py
```

and 4 on the main branch and the 3rd line is different on each.

```
test idea again
new feature
bold commit to main
slower more thoughtful fix
```

Now, we want to have all of those so we try to merge.

```
git merge hotfix
Auto-merging mymodule/important_classes.py
CONFLICT (content): Merge conflict in mymodule/important_classes.py
Automatic merge failed; fix conflicts and then commit the result.
```

but we see an error

```
nano mymodule/important_classes.py
```

in nano now we see the file looks different

```
test idea again
new feature
>>>>> HEAD
bold commit to main
slower more thoughtful fix
=======
fix older bug
<<<<<<< hotfix
```

Git did not know how to merge the two files together, so it makred the part it did not know how to handle. There's the part from the HEAD (the branch we have checked out, in this case main)

to resolve the merge conflict, we edit the file to be what we want. In this casewe want both sets of changes. SO we edit to look as follows:

```
test idea again
new feature
bold commit to main
slower more thoughtful fix
fix older bug
```

Now git tells us that we need to fix the conflicts and commit ecause we have unmerged paths.

```
git status
On branch main
Your branch is ahead of 'origin/main' by 9 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   mymodule/important_classes.py

no changes added to commit (use "git add" and/or "git commit -a")
```

```
git commit -m "keep all changes"
U       mymodule/important_classes.py
error: Committing is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
```

We have to add as usual

```
git add .
```

then we can commit.

```
git commit -m "keep all changes"
```

```
[main 62b943f] keep all changes
```

10.7. Prepare for next Class

1. Check that jupyter book is installed on your computer and the windows advice
2. Fix any open PRs you have that need to be rebased.
3. In your github in class repo, create a series of commits that tell as story of how you might have made a mistake and fixed it. Use git log and redirects to write that log to a file, `gitstory.md` in your KWL repo and then annotate your story to add in any narrative that didn't fit in the commit messages. This could be that you made a mistake in your code and used git to recover or that you got your git database to an undesirable state and got it back on track.

10.8. More Practice

1. Find an open source repository and look at the .gitignore file. In `donotcommit.md` reflect on what types of content typically get ignored and why you think they are ignored. If you can't figure out why, try to look it up.
2. Create a second git story for the other type form your first (code mistake or git mistake) in `gitstory2.md`
3. add something to the glossary, cheatsheet or a history sidebar to the notes.

11. How do we build Documentation?

11.1. Review from getting unstuck with git

11.1.1. Add a results folder and prevent its content from being committed.

Recall, this is our GitHub inclass repo:

```
cd github-in-class-brownsarahm
ls
```

```
CONTRIBUTING.md README.md       config.txt      docs            setup.py
LICENSE.md      about.md        config.yml      mymodule        tests
```

We saw last week to use the `.gitignore` file to ignore files that match a pattern.

```
cat .gitignore
```

we used a very simple pattern

```
config.*
```

to ignore any config file in this repository.

If we push

```
git push
```

> **Note**
>
> Recall here that it is compressing before pushing

```
Enumerating objects: 44, done.
Counting objects: 100% (42/42), done.
Delta compression using up to 8 threads
Compressing objects: 100% (32/32), done.
Writing objects: 100% (38/38), 2.96 KiB | 1.48 MiB/s, done.
Total 38 (delta 22), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (22/22), completed with 3 local objects.
To https://github.com/introcompsys/github-in-class-brownsarahm.git
   cea6a93..62b943f  main -> main
```

and view online, we can see that the two config files we have locally are not pushed, becuase they were not added to the .git

> **Important**
>
> This is another example of how `git push` does not push the files in the directory, but instead the `.git` directory (compressed) and then GitHub (and other git hosts) use git operations to show us the contents of the files based on the repository database.

Now, back to our task we first create the directory and then append a new line to it.

```
mkdir results
echo "results/" >> .gitignore
```

Note that the two `>` is needed, not one so tht we append instead of erase.

> **Try it yourself**
>
> What could you do if you had used only one and overwritten the .gitignore file instead of appending to it?

How can we tell that worked? As usual, `git status` is a good place to start.

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

When we do git status, we see that we changed the `.gitignore` file but this does not show us that what we put in there actually matches the directory we want to ignore *yet*.

> **Important**
>
> git does not track empty directories.

> **Tip**
>
> If we want to track an "empty" directory a common practice is to add an hidden, often empty file (`.gitkeep` is also hidden) so that it is not really empty anymore, but is empty-looking when we use, eg `ls`

If we create another empty directory we can see that this is also not showing as an untracked file.

```
mkdir res2
```

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

We can add an empty file to test that our ignore actually worked.

```
touch results/test1
```

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

and compare to what hapens if we add an empty file to the directory that does not match our pattern.

```
touch res2/test1
```

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        res2/

no changes added to commit (use "git add" and/or "git commit -a")
```

> **Note**
>
> a directory where all files are untracked stays as just the directory no matter how many files it is, until you add at least one.

```
touch res2/test2
```

```
git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        res2/

no changes added to commit (use "git add" and/or "git commit -a")
```

> **Note**
>
> merge conflict file explanation to be added later

11.2. What does it mean to build documentation?

> **Note**
>
> For now, see the prismia transcript for an outline here

11.2.1. What is a build?

11.2.2. Why is documentation so important?

documentation types table

ethnography of docuemtnation data science

11.3. Building Documentation with Jupyter book

There are many Documenation Tools because we (programmers) don't like

linux kernel uses sphinx

why and how it works

Jupyterbook wraps sphinx and uses markdown instead of restructured text

11.3.1. Creating a Jupyterbook from the template

We'll naviate out of the github in class repo

```
cd ..
```

and use jupyterbook to create a new book from the template

```
jupyter-book create tiny-book
```

We see this

```
================================================================================

Your book template can be found at

    tiny-book/

================================================================================
```

and that it created a new directory

```
ls
```

```
github-in-class-brownsarahm     test                            tiny-book
nand2tetris                     test2
```

If we navigate to that directory we can create a new repo here as well

```
cd tiny-book/
git init .
```

```
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/brownsarahm/Documents/sysinclass/tiny-book/.git/
```

and again we'll change our default branch to main

```
git branch -m main
```

```
git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        _config.yml
        _toc.yml
        intro.md
        logo.png
        markdown.md
        notebooks.ipynb
        references.bib
        requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
```

and we will commit the template.

```
git add .
```

```
git commit -m 'empty template'
[main (root-commit) a8a489e] empty template
 8 files changed, 362 insertions(+)
 create mode 100644 _config.yml
 create mode 100644 _toc.yml
 create mode 100644 intro.md
 create mode 100644 logo.png
 create mode 100644 markdown.md
 create mode 100644 notebooks.ipynb
 create mode 100644 references.bib
 create mode 100644 requirements.txt
```

11.3.2. Strucutre of a Jupyter book

```
ls
_config.yml             intro.md                markdown.md             references.bib
_toc.yml                logo.png                notebooks.ipynb         requirements.txt
```

A jupyter book has two required files (`_config.yml` and `_toc.yml`)

- config defaults
- toc file formatting rules

the extention (`.yml`) is yaml, which stands for “YAML Ain’t Markup Language”. It consists of key, value pairs and is deigned to be a human-friendly way to encode data for use in any programming language.

```
cat _config.yml
```

The configuration file, tells it basic iformation about the book, it provides all of the settings that jupyterbook and sphinx need to render the content as whatever output format we want.

```
# Book settings
# Learn more at https://jupyterbook.org/customize/config.html

title: My sample book
author: The Jupyter Book Community
logo: logo.png

# Force re-execution of notebooks on each build.
# See https://jupyterbook.org/content/execute.html
execute:
  execute_notebooks: force

# Define the name of the latex output file for PDF builds
latex:
  latex_documents:
    targetname: book.tex

# Add a bibtex file so that we can create citations
bibtex_bibfiles:
  - references.bib

# Information about where the book exists on the web
repository:
  url: https://github.com/executablebooks/jupyter-book  # Online location of your book
  path_to_book: docs  # Optional path to your book, relative to the repository root
  branch: master  # Which branch of the repository should be used when creating links (optional)

# Add GitHub buttons to your book
# See https://jupyterbook.org/customize/config.html#add-a-link-to-your-repository
html:
  use_issues_button: true
  use_repository_button: true
```

The table of contents file describe how to put the other files in order.

```
cat _toc.yml
```

```
# Table of contents
# Learn more at https://jupyterbook.org/customize/toc.html

format: jb-book
root: intro
chapters:
- file: markdown
- file: notebooks
```

> **Try it yourself**
>
> Which files created by the template are not included in the rendered output? How could you tell?

The other files are optional, but common. Requirements.txt is the format for pip to install python depndencies. There are different standards in other languages for how

```
cat requirements.txt
jupyter-book
matplotlib
numpy
```

the .bib file allows you to put structured data in bibtex format and then jupyterbook can write a bibliography for you.

11.3.3. How do I get output?

```
jupyter-book build .
```

```
Running Jupyter-Book v0.11.1
Source Folder: /Users/brownsarahm/Documents/sysinclass/tiny-book
Config Path: /Users/brownsarahm/Documents/sysinclass/tiny-book/_config.yml
Output Path: /Users/brownsarahm/Documents/sysinclass/tiny-book/_build/html
Running Sphinx v3.2.1
making output directory... done
[etoc] Changing master_doc to 'intro'
checking for /Users/brownsarahm/Documents/sysinclass/tiny-book/references.bib in bibtex cache... not found
parsing bibtex file /Users/brownsarahm/Documents/sysinclass/tiny-book/references.bib... parsed 5 entries
myst v0.13.7: MdParserConfig(renderer='sphinx', commonmark_only=False, dmath_allow_labels=True, dmath_allow_space=True,
dmath_allow_digits=True, update_mathjax=True, enable_extensions=['colon_fence', 'dollarmath', 'linkify', 'substitution'],
disable_syntax=[], url_schemes=['mailto', 'http', 'https'], heading_anchors=None, html_meta=[], footnote_transition=True,
substitutions=[], sub_delimiters=['{', '}'])
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 3 source files that are out of date
updating environment: [new config] 3 added, 0 changed, 0 removed
Executing: notebooks in: /Users/brownsarahm/Documents/sysinclass/tiny-book

looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] notebooks
generating indices...  genindexdone
writing additional pages...  searchdone
copying images... [100%] _build/jupyter_execute/notebooks_2_0.png
copying static files... ... done
copying extra files... done
dumping search index in English (code: en)... done
dumping object inventory... done
build succeeded.

The HTML pages are in _build/html.
[etoc] missing index.html written as redirect to 'intro.html'

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

Finished generating HTML for book.
Your book's HTML pages are here:
    _build/html/
You can look at your book by opening this file in a browser:
    _build/html/index.html
Or paste this line directly into your browser bar:
    file:///Users/brownsarahm/Documents/sysinclass/tiny-book/_build/html/index.html

================================================================================
```

Then we can see in browser what the output looks like.

we can also see that a _build directory was created and examine that

```
cd _build/
ls
```

it has some cache (jupyter_execute) and our output (html). Building to html is the default, but there are many types of builds output

```
html           jupyter_execute
```

We can see in the html folder

```
cd html/
ls
```

that there are `html` files for each input file, plus some extra (genindex) and ther is javascript for searching the site

```
_images         _sources        genindex.html   intro.html      notebooks.html  search.html
_panels_static  _static         index.html      markdown.html   objects.inv     searchindex.js
```

the static folder contains the rest of the formatting

```
ls _static/
```

```
__init__.py
__pycache__
basic.css
clipboard.min.js
copy-button.svg
copybutton.css
copybutton.js
copybutton_funcs.js
css
doctools.js
documentation_options.js
file.png
images
jquery-3.5.1.js
jquery.js
js
language_data.js
logo.png
minus.png
mystnb.css
panels-main.c949a650a448cc0ae9fd3441c0e17fb0.css
panels-variables.06eb56fa6e07937060861dad626602ad.css
plus.png
pygments.css
searchtools.js
sphinx-book-theme.12a9622fbb08dcb3a2a40b2c02b83a57.js
sphinx-book-theme.acff12b8f9c144ce68a297486a2fa670.css
sphinx-book-theme.css
sphinx-thebe.css
sphinx-thebe.js
togglebutton.css
togglebutton.js
underscore-1.3.1.js
underscore.js
vendor
webpack-macros.html
```

We didn't have to write any html and we got a responsive site!

If you wanted to change the styling with sphinx you can use built in themes which tell sphinx to put different files in the `_static` folder when it builds your site, but you don't have to change any of your content! If you like working on front end things (which is great! it's just not alwasy the goal) you can even build your own theme that can work with sphinx.

Jupyterbook is pretty opinionated on the styling, but for more general documenation websites the options are good

```
cd ../../
```

```
ls
_build                  intro.md                notebooks.ipynb
_config.yml             logo.png                references.bib
_toc.yml                markdown.md             requirements.txt
```

We're going to ignore the built files and use this later to automatically build them on GitHub.

```
echo "_build/" > .gitignore
```

```
git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

and then add and commit

```
git add .
```

```
git commit -m 'ignroe build'
[main 3e249ce] ignroe build
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

11.4. How do I push a repo that I made locally to GitHub

For today, create an empty github repo shared with me.

More generally, you can create a repo

That default page for an empty repo if you do not initiate it with any files will give you the instructions for what remote to add.

THen we add the remote

```
git remote add origin https://github.com/introcompsys/tiny-book-brownsarahm.git
```

and push to origin main.

```
git push -u origin main
\Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (13/13), 16.07 KiB | 8.03 MiB/s, done.
Total 13 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/introcompsys/tiny-book-brownsarahm.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
(base) brownsarahm@tiny-book $
```

11.5. Prepare for next Class

1. review the notes
2. (priority) Convert your KWL repo to a jupyter book. Ignore the builds from your reposiotry. Use the documenation to choose your settings: link it to your repo, do not serve it publicly on github. Be sure that it can build and if you encounter problems, create a `jupyterbooktroubleshooting.md` file and log them with solutions if you find them.
3. complete Feedback survey if you have not alredy.

11.6. More Practice

1. build your kwl repo to a pdf or one other format locally.
2. Add `templating.md` to your KWL repo and explain templating in your own words using other programming concepts you have learned so far. Include in a markdown (same as HTML `<!-- comment -->`) comment the list of CSC courses you have taken for context while we give you feedback.
3. Learn about the documentation ecosystem in another language that you know. In `docs.md` include a summary of your findings and compare and contrast it to jupyter book/sphinx.

11.7. Questions after class

11.7.1. Could I use jupyterbook to put a resume online?

In theory, yes. However there are other tools that work similar in the sense of separating content from HTML/CSS/javascript in order to generate a website. Sphinx is more customizable, as noted above, but also, Static site generators are a whole category of software tools.

GitHub pages is free hosting, which by default uses jekyll however, that is no longer very active, due to the death of one of the core maintainers.

Some are centered on blog type sites (like jekyll) others are not. There are even ways to use a javascript slide tool to write in markdown but render for the web.

12. Shell Scripting

12.1. Feedback

12.2. Request: posted videos

- videos are a lot of work for them to be high quality and accessible
- even just a basic zoom is one more thing to manage
- the notes have a *complete* transcript of every action on the terminal
- I will pilot using webcaptioner to produce a transcript of more detail. I will then use these to fill in the notes and/or post them on slack (in a dedicated channel)

12.2.1. Request: more clear instructions

- Sorry, I'm adapting based on how far we get, so I don't have time to get iterative feedback on them.
- I will try to expand on them and add examples, also some patterns will emerge so you'll get used to the type of thing that's expected.
- also *always* feel free to ask more questions (slack or github issue to class repo)
- you *always* can revise your work, so it's also okay, espeically for more practice to put in the place of a response a question or try re-interpreting the instructions to check if they're correct.
- recent updates commit

12.2.2. Slack can be annoying

- check your notification settings!!
- use e-mail notifications if you prefer

12.2.3. Installations are hard

- this is sort of generally true, sorry
- the download part makes them not good for in class
- always ask in slack
- we might be done with most for a while
- I will try to link to more detaled instructions if i can find them.

12.2.4. Request: More direction for extensions

- instructions on site
- Will start to add notes with a marker

12.2.5. Multiple repositories is hard to manage

- try to think of them as separate directories and have an offline copy of each. We sort files often.
- The reason for multiple is because that's actually somewhat more realistic, partially and partially becuase of how templates work

12.2.6. This should be a 200 level course/ required

- the goal is for it to eventually be required and for most students to take it at the same time as 212. I'm glad (and other faculty will be too!) that you seem to agree

12.2.7. Some things only work on mac

- I *think* everything essential can be done on any OS, but there are different ways.
- I try to check everything and we are through the most tricky parts.
- If i miss something

12.2.8. More lecture/slides would be helpful

- I probably shouldn't have offered "slides" I will use prismia so that I can stay adaptive and swap the order in response
- I will try to prepare diagrams for topics in advance and/or use the prismia drawing tool,
- *feel free to ask me to draw or explain* either by raising your hand or on prismia during class

12.3. Jupyter Book

To make your KWL repo a jupyterbook, you need to:

- add the required files (`_toc.yml` and `_config.yml`)
- add your files to `_toc.yml`
- customize meta-data and settings in `_config.yml`

To add images, you add the file to the repository *and* mention it in a markdown file using the following syntax.

```
![mandatory alt text](relative/path/to/image)
```

also, myst cheatsheet from jupyterbook docs

12.4. Bash has programming language features

Bash includes most common basic programming language features:

- conditional expressions and constructs
- loops
- variables and builtin varialbes
- comments `#`

12.4.1. Variables

```
echo $SHELL
```

```
/bin/bash
```

12.4.2. Loops

The syntax for a loop is:

```
for name [ [in [words …] ] ; ] do commands; done
```

or, with more spacing for readability

```
for item in [LIST]
do
  [COMMANDS]
done
```

Note in this syntax, we do not need a counter variable. In each iteration through the loop the next item from `[LIST]` becomes the value of `item`.

In our case, we can use this like:

```
for file in README.md gitoffline.md terminal.md software.md abstraction.md chart.md reorg.md stdinouterr.md hardwaresurvey.md
gitlog.txt workflows.md gitplumbing.md gitunderstanding.md idethoughts.md numbers.md hexspeak.md gitstory.md gitstory2.md
donotcommit.md jupyterbooktroubleshooting.md templating.md docs.md
> do
> echo $file
> done
```

To send all of the file names to STDOUT:

```
README.md
gitoffline.md
terminal.md
software.md
abstraction.md
chart.md
reorg.md
stdinouterr.md
hardwaresurvey.md
gitlog.txt
workflows.md
gitplumbing.md
gitunderstanding.md
idethoughts.md
numbers.md
hexspeak.md
gitstory.md
gitstory2.md
donotcommit.md
jupyterbooktroubleshooting.md
templating.md
docs.md
```

12.4.3. Conditionals

```
for file in README.md gitoffline.md terminal.md software.md abstraction.md chart.md reorg.md stdinouterr.md hardwaresurvey.md
gitlog.txt workflows.md gitplumbing.md gitunderstanding.md idethoughts.md numbers.md hexspeak.md gitstory.md gitstory2.md
donotcommit.md jupyterbooktroubleshooting.md templating.md docs.md
> do
> if test -f $file; then
> echo $file
> fi
> done
```

12.5. Shell Scripts

So, now we have checked your repository, but for you to use this on a regular basis there are two challenges.

1. You would have to type (or copy paste) those lines every time
2. the list of files will expand but the list in that will not update.

We will address the first with a bash script. Bash scripts allow you to make programs, including small utiltiies that help you with little tasks.

```
nano checker.sh
```

And we can paste this into the file:

```
for file in README.md gitoffline.md terminal.md software.md abstraction.md chart.md reorg.md stdinouterr.md hardwaresurvey.md
gitlog.txt workflows.md gitplumbing.md gitunderstanding.md idethoughts.md numbers.md hexspeak.md gitstory.md gitstory2.md
donotcommit.md jupyterbooktroubleshooting.md templating.md docs.md
> do
> if ! test -f $file; then
> echo $file
> fi
> done
```

12.6. Installing from source

We have used programs that we installed from their installation tools. Today we will use a small program I wrote for class, that we install from source.

```
git clone https://github.com/introcompsys/courseutils
```

```
Cloning into 'courseutils'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 16 (delta 5), reused 6 (delta 2), pack-reused 0
Receiving objects: 100% (16/16), 4.03 KiB | 4.03 MiB/s, done.
Resolving deltas: 100% (5/5), done.
```

First we can look at what this contains.

```
cd courseutils/
ls
```

It is very lightweight:

```
LICENSE         README.md       kwltracking.py  setup.py
```

Let's review the purpose of each

- This repo is public, so it has a license to describe how it is pertmitted to be used
- The README is information about the repository; currently it has the documentation
- kwltracking.py is the actual code of the program
- setup.py is instructions for python's `pip` on how to install the program.

```
pip install .
```

> **Note**
>
> This output is truncated

If it works, there will be along output that ends like:

```
...
Successfully installed syscourseutils-0.1.0
```

After you install, you will need to open a new terminal window (or otherwise restart bash, which varies on a lot of parameters)

Making it *installable* changes how we can run and use the program. In particular, it makes it a bash command.

So, on the terminal, we can use it like:

```
kwlfilecheck
```

This program scrapes the course website and echos to stdout the list of files expected in the KWL chart repo.

```
README gitoffline.md terminal.md software.md abstraction.md chart.md reorg.md stdinouterr.md hardwaresurvey.md gitlog.txt
workflows.md gitplumbing.md gitunderstanding.md idethoughts.md numbers.md hexspeak.md gitstory.md gitstory2.md donotcommit.md
jupyterbooktroubleshooting.md templating.md docs.md
```

Now, back in your KWL repo, we want to change our script to use this.

```
cd ..
```

We can edit the script, with nano

```
nano checker.sh
```

We can edit it so that it calls that process, assigns its output to a variable, and then uses that variable as our list output like this:

```
for file in $(kwlfilechecker)
do
 if ! test -f $file; then
  echo $file
 fi
done
```

Now you can use:

```
bash checker.sh
```

To see what, if any files you are missing.

12.7. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Review the notes (to reinforce ideas and improve your memory of them)
2. Update your KWL to include all rows listed on the KWL page of the course site. (because tracking your learning deepens your learning)
3. **priority** Make an issue on your grading contract repo that includes: a self-assessment of your progress on your contract so far and a plan going forward. Tag @sp22instructors so that we can help you meet your goals. (so that you get the grade you want)
4. **priority** On Windows, install Putty (we will use this Monday)
5. Write a bash script that checks only if the number of files in your kwl meets or exceeds the minimum (number from list + 2 for toc and config) OR that does something else of your choosing. Add it to your KWL repo in a utils directory. Include Comments. (to practice so that you remember)

12.8. More Practice

1. **priority** add or link a glossary or cheatsheet, OR add a box with some historical context or extra reading links to the notes from any past class (in order to practice git/github and using jupyter book features, which are similar to other documentation tools, so even if you do not work in a python centric environment the concepts will translate)

> **Hint**
>
> See documentation on shell arithmetic for counting

12.9. Extension Ideas

1. document or add course scripts to the courseutils repo
2. write a similar util for another context and document it
3. turn your bash script above (if not the toy counting one) into a longer tutorial that explains its components piece by piece, how they work together.
4. use the jupyter book api referencing capability to document a side project or code from a different course, eg 110 (keep it in a separate private repo so as to not put solutions to a course on the public internet)

12.10. Questions after class

12.10.1. Will we create bash programs from scratch?

At least small ones, we probably will not write large programs though.

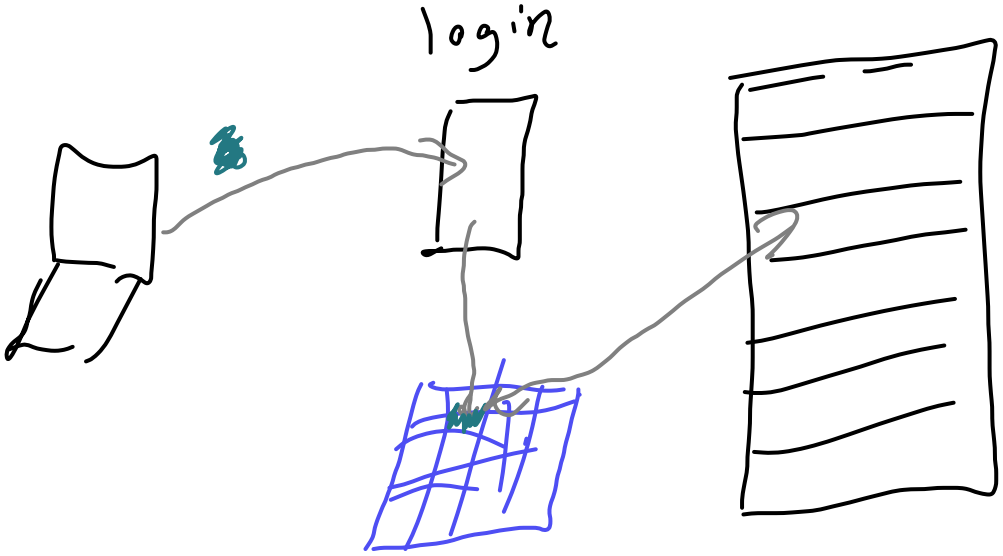
12.10.2. What are some applications of bash scripting?

We will see its use in HPC in the next class, but also for small utilities like this.

13. How can I work on a remote server?

Today we will connect to a remote server, use an HPC system with scheduler, and learn new bash commands for working with the *content* of files.

13.1. What are remote servers and HPC systems?



13.2. Connecting to Seawulf

We connect with secure shell or `ssh` from our terminal (GitBash or Putty on windows) to URI's teaching High Performance Computing (HPC) Cluster Seawulf.

> **Warning**
>
> This cluster is for course related purposes at URI, if you want to use a HPC system of some sort for a side project, consider Amazon Web Services, Google Cloud, or Microsoft Azul services, you can get some allocation for free a a student.
> If you are doing research supervised by a URI professor, there are other servers on campus and URI participates in a regional HPC resource as well.

Our login is the part of your uri e-mail address before the @ and I will tell you how to find your default password if you missed class.

```
ssh -l brownsarahm seawulf.uri.edu
```

When it logs in it looks like this and requires you to change your password. They configure it with a default and with it past expired.

```
The authenticity of host 'seawulf.uri.edu (131.128.217.210)' can't be established.
ECDSA key fingerprint is SHA256:RwhTUyjWLqwohXiRw+tYlTiJEbqX2n/drCpkIwQVCro.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added 'seawulf.uri.edu,131.128.217.210' (ECDSA) to the list of known hosts.
brownsarahm@seawulf.uri.edu's password:
You are required to change your password immediately (root enforced)
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for user brownsarahm.
Changing password for brownsarahm.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
Connection to seawulf.uri.edu closed.
```

after you give it a new password, then it logs you out and you have to log back in.

```
brownsarahm@~ $ ssh -l brownsarahm seawulf.uri.edu
```

```
brownsarahm@seawulf.uri.edu's password:
Last login: Tue Mar  8 12:52:38 2022 from 172.20.133.152
```

We have logged into our home directory which is empty

```
[brownsarahm@seawulf ~]$ ls
```

```
[brownsarahm@seawulf ~]$ pwd
/home/brownsarahm
```

```
[brownsarahm@seawulf ~]$ whoami
brownsarahm
```

Notice that the prompt says `uriusername@seawulf` to indicate that you are logged into the server, not working locally.

13.3. Downloading files

```
[brownsarahm@seawulf ~]$ wget http://www.hpc-carpentry.org/hpc-shell/files/bash-lesson.tar.gz
--2022-03-08 12:58:09--  http://www.hpc-carpentry.org/hpc-shell/files/bash-lesson.tar.gz
Resolving www.hpc-carpentry.org (www.hpc-carpentry.org)... 104.21.33.152, 172.67.146.136
Connecting to www.hpc-carpentry.org (www.hpc-carpentry.org)|104.21.33.152|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12534006 (12M) [application/gzip]
Saving to: ‘bash-lesson.tar.gz’

100%[======================================>] 12,534,006  4.19MB/s   in 2.9s

2022-03-08 12:58:12 (4.19 MB/s) - ‘bash-lesson.tar.gz’ saved [12534006/12534006]
```

```
[brownsarahm@seawulf ~]$ ls
bash-lesson.tar.gz
```

```
[brownsarahm@seawulf ~]$ tar -xvf bash-lesson.tar.gz
dmel-all-r6.19.gtf
dmel_unique_protein_isoforms_fb_2016_01.tsv
gene_association.fb
SRR307023_1.fastq
SRR307023_2.fastq
SRR307024_1.fastq
SRR307024_2.fastq
SRR307025_1.fastq
SRR307025_2.fastq
SRR307026_1.fastq
SRR307026_2.fastq
SRR307027_1.fastq
SRR307027_2.fastq
SRR307028_1.fastq
SRR307028_2.fastq
SRR307029_1.fastq
SRR307029_2.fastq
SRR307030_1.fastq
SRR307030_2.fastq
```

```
[brownsarahm@seawulf ~]$ ls
bash-lesson.tar.gz                          SRR307026_1.fastq
dmel-all-r6.19.gtf                          SRR307026_2.fastq
dmel_unique_protein_isoforms_fb_2016_01.tsv  SRR307027_1.fastq
gene_association.fb                         SRR307027_2.fastq
SRR307023_1.fastq                           SRR307028_1.fastq
SRR307023_2.fastq                           SRR307028_2.fastq
SRR307024_1.fastq                           SRR307029_1.fastq
SRR307024_2.fastq                           SRR307029_2.fastq
SRR307025_1.fastq                           SRR307030_1.fastq
SRR307025_2.fastq                           SRR307030_2.fastq
```

13.4. Working with large files

One of these files, contains the entire genome for the common fruitfly, let's take a look at it: \

```
[brownsarahm@seawulf ~]$ cat dmel-all-r6.19.gtf
```

```
X       FlyBase gene    19961297        19969323        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
2L      FlyBase 3UTR    782825  782885  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
```

> **Warning**
>
> this output is truncated for display purposes

We see that this actually take a long time to output and is way tooo much information to actually read. In fact, in order to make the website work, I had to cut that content using command line tools, my text editor couldn't open the file and GitHub was unhappy when I pushed it.

For a file like this, we don't really want to read the whole file but we do need to know what it's strucutred like in order to design programs to work with it.

`head` lets us look at the first 10 lines.

```
[brownsarahm@seawulf ~]$ head dmel-all-r6.19.gtf
```

```
X       FlyBase gene    19961297        19969323        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
X       FlyBase mRNA    19961689        19968479        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase 5UTR    19961689        19961845        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19961689        19961845        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19963955        19964071        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19964782        19964944        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19965006        19965126        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19965197        19965511        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19965577        19966071        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
X       FlyBase exon    19966183        19967012        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
```

We can use the `-n` parameter to change the number.

And, tails shows the last few.

```
[brownsarahm@seawulf ~]$ tail dmel-all-r6.19.gtf
```

which in this case looks mostly the same

```
2L      FlyBase exon    782124  782181  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase exon    782238  782441  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase exon    782495  782885  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase start_codon     781297  781299  .       +       0       gene_id "FBgn0041250"; gene_symbol "Gr21a";
transcript_id "FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase CDS     781297  782048  .       +       0       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase CDS     782124  782181  .       +       1       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase CDS     782238  782441  .       +       0       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase CDS     782495  782821  .       +       0       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase stop_codon      782822  782824  .       +       0       gene_id "FBgn0041250"; gene_symbol "Gr21a";
transcript_id "FBtr0331651"; transcript_symbol "Gr21a-RB";
2L      FlyBase 3UTR    782825  782885  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
```

We can also see how much content is in the file `wc` give a word count and with its `-l` parameter gives us the number of lines.

```
[brownsarahm@seawulf ~]$ wc -l dmel-all-r6.19.gtf
```

```
542048 dmel-all-r6.19.gtf
```

Over five hundred forty thousand lines is a lot.

How can we get the number of lines in each of the `.fastq` files?

```
[brownsarahm@seawulf ~]$ wc -l *.fastw
wc: *.fastw: No such file or directory
```

note that in my typo, it tells me no files matched my pattern.

```
[brownsarahm@seawulf ~]$ wc -l *.fastq
   20000 SRR307023_1.fastq
   20000 SRR307023_2.fastq
   20000 SRR307024_1.fastq
   20000 SRR307024_2.fastq
   20000 SRR307025_1.fastq
   20000 SRR307025_2.fastq
   20000 SRR307026_1.fastq
   20000 SRR307026_2.fastq
   20000 SRR307027_1.fastq
   20000 SRR307027_2.fastq
   20000 SRR307028_1.fastq
   20000 SRR307028_2.fastq
   20000 SRR307029_1.fastq
   20000 SRR307029_2.fastq
   20000 SRR307030_1.fastq
   20000 SRR307030_2.fastq
  320000 total
```

when it does work, we also get the total.

We can use redirects as before to save these to a file:

```
[brownsarahm@seawulf ~]$ wc -l *.fastq > linecounts.txt
```

```
[brownsarahm@seawulf ~]$ cat linecounts.txt
   20000 SRR307023_1.fastq
   20000 SRR307023_2.fastq
   20000 SRR307024_1.fastq
   20000 SRR307024_2.fastq
   20000 SRR307025_1.fastq
   20000 SRR307025_2.fastq
   20000 SRR307026_1.fastq
   20000 SRR307026_2.fastq
   20000 SRR307027_1.fastq
   20000 SRR307027_2.fastq
   20000 SRR307028_1.fastq
   20000 SRR307028_2.fastq
   20000 SRR307029_1.fastq
   20000 SRR307029_2.fastq
   20000 SRR307030_1.fastq
   20000 SRR307030_2.fastq
  320000 total
```

We can also search files, without loading them all into memory or displaying them, with `grep`:

```
[brownsarahm@seawulf ~]$ grep Act5c dmel-all-r6.19.gtf
```

```
[brownsarahm@seawulf ~]$ grep mRNA dmel-all-r6.19.gtf
```

this output a lot, so the output is truncated here

```
X       FlyBase mRNA    19961689        19968479        .       +       .       gene_id "FBgn0031081"; gene_symbol "Nep3";
transcript_id "FBtr0070000"; transcript_symbol "Nep3-RA";
2L      FlyBase mRNA    781276  782885  .       +       .       gene_id "FBgn0041250"; gene_symbol "Gr21a"; transcript_id
"FBtr0331651"; transcript_symbol "Gr21a-RB";
```

and we can combine `grep` with `wc` to count occurences.

```
[brownsarahm@seawulf ~]$ grep mRNA dmel-all-r6.19.gtf | wc -l
34025
```

13.5. File permissions

Let's make a small script, recalling what we have learned so far:

```
[brownsarahm@seawulf ~]$ echo "echo 'script works'" >> demo.sh
```

We can confirm that the script looks like a we expected

```
[brownsarahm@seawulf ~]$ cat demo.sh
echo 'script works'
```

One thing we could do is to run the script using `./`

```
[brownsarahm@seawulf ~]$ ./demo.sh
```

but we get a permission denied error

```
-bash: ./demo.sh: Permission denied
```

By default, files have different types of permissions: read, write, and execute for different users that can access them. To view the permissions, we can use the `-l` option of `ls`.

```
[brownsarahm@seawulf ~]$ ls -l
total 138452
-rw-r--r--. 1 brownsarahm spring2022-csc392 12534006 Apr 18  2021 bash-lesson.tar.gz
-rw-r--r--. 1 brownsarahm spring2022-csc392       20 Mar  8 13:12 demo.sh
-rw-r--r--. 1 brownsarahm spring2022-csc392 77426528 Jan 16  2018 dmel-all-r6.19.gtf
-rw-r--r--. 1 brownsarahm spring2022-csc392   721242 Jan 25  2016 dmel_unique_protein_isoforms_fb_2016_01.tsv
-rw-r--r--. 1 brownsarahm spring2022-csc392 25056938 Jan 25  2016 gene_association.fb
-rw-r--r--. 1 brownsarahm spring2022-csc392      447 Mar  8 13:07 linecounts.txt
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_2.fastq
```

For each file we get 10 characters in the first column that describe the permissions. The 3rd column is the username of the owner, the fourth is the group, then size date revised and the file name.

We are most interested in the 10 character permissions. The fist column indicates if any are directories with a `d` or a `-` for files. We have no directories, but we can create one to see this.

```
[brownsarahm@seawulf ~]$ mkdir results
```

```
[brownsarahm@seawulf ~]$ ls -l
total 138452
-rw-r--r--. 1 brownsarahm spring2022-csc392 12534006 Apr 18  2021 bash-lesson.tar.gz
-rw-r--r--. 1 brownsarahm spring2022-csc392       20 Mar  8 13:12 demo.sh
-rw-r--r--. 1 brownsarahm spring2022-csc392 77426528 Jan 16  2018 dmel-all-r6.19.gtf
-rw-r--r--. 1 brownsarahm spring2022-csc392   721242 Jan 25  2016 dmel_unique_protein_isoforms_fb_2016_01.tsv
-rw-r--r--. 1 brownsarahm spring2022-csc392 25056938 Jan 25  2016 gene_association.fb
-rw-r--r--. 1 brownsarahm spring2022-csc392      447 Mar  8 13:07 linecounts.txt
**drwxr-xr-x. 2 brownsarahm spring2022-csc392       10 Mar  8 13:16 results**
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_2.fastq
```

We can see in the bold line, that the first character is a d.

The next nine characters indicate permission to **R**ead, **W**rite, and e**X**ecute a file. With either the letter or a `-` for permissions not granted, they appear in three groups of three, three characters each for owner, group, anyone with access.

If we want to run the file, we *can* instead use `bash` directly, but this is limited relative to calling our script in other ways.

```
[brownsarahm@seawulf ~]$ bash demo.sh
script works
```

Instead, to add execute permission, we can use `chmod`

```
[brownsarahm@seawulf ~]$ chmod +x demo.sh
```

```
[brownsarahm@seawulf ~]$ ls -l
total 138452
-rw-r--r--. 1 brownsarahm spring2022-csc392 12534006 Apr 18  2021 bash-lesson.tar.gz
-rwxr-xr-x. 1 brownsarahm spring2022-csc392       20 Mar  8 13:12 demo.sh
-rw-r--r--. 1 brownsarahm spring2022-csc392 77426528 Jan 16  2018 dmel-all-r6.19.gtf
-rw-r--r--. 1 brownsarahm spring2022-csc392   721242 Jan 25  2016 dmel_unique_protein_isoforms_fb_2016_01.tsv
-rw-r--r--. 1 brownsarahm spring2022-csc392 25056938 Jan 25  2016 gene_association.fb
-rw-r--r--. 1 brownsarahm spring2022-csc392      447 Mar  8 13:07 linecounts.txt
drwxr-xr-x. 2 brownsarahm spring2022-csc392       10 Mar  8 13:16 results
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_2.fastq
```

```
[brownsarahm@seawulf ~]$ ./demo.sh
script works
```

```
[brownsarahm@seawulf ~]$ nano demo.sh
```

We can add a bit more to our script to make it more interesting

```
for VAR in *.gz
do
  echo $VAR
done


echo 'script works'
```

and note that tht does not change the permission.

```
[brownsarahm@seawulf ~]$ ls -l
total 138452
-rw-r--r--. 1 brownsarahm spring2022-csc392 12534006 Apr 18  2021 bash-lesson.tar.gz
-rwxr-xr-x. 1 brownsarahm spring2022-csc392       60 Mar  8 13:27 demo.sh
-rw-r--r--. 1 brownsarahm spring2022-csc392 77426528 Jan 16  2018 dmel-all-r6.19.gtf
-rw-r--r--. 1 brownsarahm spring2022-csc392   721242 Jan 25  2016 dmel_unique_protein_isoforms_fb_2016_01.tsv
-rw-r--r--. 1 brownsarahm spring2022-csc392 25056938 Jan 25  2016 gene_association.fb
-rw-r--r--. 1 brownsarahm spring2022-csc392      447 Mar  8 13:07 linecounts.txt
drwxr-xr-x. 2 brownsarahm spring2022-csc392       10 Mar  8 13:16 results
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625262 Jan 25  2016 SRR307023_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625376 Jan 25  2016 SRR307024_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625286 Jan 25  2016 SRR307025_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625302 Jan 25  2016 SRR307026_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625312 Jan 25  2016 SRR307027_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625338 Jan 25  2016 SRR307028_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625390 Jan 25  2016 SRR307029_2.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_1.fastq
-rw-r--r--. 1 brownsarahm spring2022-csc392  1625318 Jan 25  2016 SRR307030_2.fastq
```

13.6. Making a batch file

```
[brownsarahm@seawulf ~]$ nano my_job.sh
```

```
[brownsarahm@seawulf ~]$ sbatch my_job.sh
Submitted batch job 23950
```

```
[brownsarahm@seawulf ~]$ squeue --job 23950
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

13.7. Interactive mode

So far we have worked on the login node which has very little computational power and submitted a job to a compute node through the scheduler for it to run asynchronously. We can also use a compute node interactively if we have a computationally intensive task that we need to work with in real time, instead of it being fully automated.

We'll talk after the break about how this might look, but for example if our `grep` commands needed more power where we need the resutl back right away in order to then decide what the next calculation could be.

We can request a node to be allocated to our user and have a direct `ssh` connection to compute node using `interactive`. Interactive by default calls a specific script that is configured by the system administrators for the particular cluster that you are logged into.

For seawulf, interactive opens a default of 8 hgours, but you can use parameters to request more or less time.

```
[brownsarahm@seawulf ~]$ interactive
```

When we run this interactive submits a "job" to the schedule, but instead of then passing a script to the compute node it connects us to that node via ssh

```
salloc: Granted job allocation 23963
salloc: Waiting for resource configuration
salloc: Nodes n005 are ready for job
/usr/bin/id: cannot find name for user ID 1168
/usr/bin/id: cannot find name for group ID 1111
/usr/bin/id: cannot find name for user ID 1168
```

We can see that we are connected to a system with more resources using this:

```
[I have no name!@n005 ~]$ lshw
WARNING: you should run this program as super-user.
          configuration: driver=system
WARNING: output may be incomplete or inaccurate, you should run this program as super-user.
```

I truncataed the long output.

We can then exit the compute node and go back to the login node with `exit`

```
[I have no name!@n005 ~]$ exit
```

```
logout
salloc: Relinquishing job allocation 23963
```

Note tht it says the job has ended.

13.8. Logging off

You can log off with `exit` it is important to do so to not use resources you do not need. However, note tht if your laptop loses internet connection your session will also be disconnected. This is another advantage of the scheduler, you don't need to maintain a connection for it to run your code.

```
[brownsarahm@seawulf ~]$ exit
```

```
logout
Connection to seawulf.uri.edu closed.
```

Note that when you are back to your computer the prompt changes. And that using the up arrow, the last command your computer's shell knows is the `ssh` line.

```
brownsarahm@~ $ ssh -l brownsarahm seawulf.uri.edu
```

13.9. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Review the notes (to reinforce ideas and improve your memory of them)
2. **priority** add a `## summary` section to your IDE notes where you summarize what you observed in your IDE usage and what questions or what features (we will work with this after break and I want you to prepare before break)
3. Modify the sbatch job we made in class so that you get files back for any errors and output from the script. see the options (to review/practice what we did)

13.10. More Practice

> **Try it Yourself**
>
> use scp to download the files generated to your local computer and add them to your kwl chart. (a possible extenstion idea to practice reading the official documentation to learn a new tool that is similar to ones you have seen before)

1. **priority** write a script that outputs the first 3 lines of each `.fastq` line and sbatch script that runs it and saves the output to files and e-mails you when it is done. Submit the job. (to reinforce /practice what we saw in class)
2. File permissions are represented numerically often in octal, by transforming the permissions for each level (owner, group, all) as binary into a number. Add `octal.md` to your KWL repo and answer the following. Try to think through it on your own, but you may look up the answers, as long as you link to (or ideally cite using jupyterbook citations) a high quality source.

```
1. Transform the permissions [`r--`, `rw-`, `rwx`] to octal, by treating it as
three bits.
1. Transform the permission we changed our script to `rwxr-xr-x` to octal.
1. Which of the following would prevent a file from being edited by other
people 777 or 755?
```

3. Answer the following in `hpc.md` of your KWL repo: (to think about how the design of the system we used in class impacts programming and connect it to other ideas taught in CS)

```
1. What kinds of things would your code need to do if you were going to run it
on an HPC system?
1. What Sbatch options seem the most helpful?
1. How might you go about setting the time limits for a script? How could you
estimate how long it will take?
```

13.11. Questions After Class

13.11.1. When is sbatch used?

Sbatch is one type of scheduler, that manages the resource of a high performance computing system

13.11.2. will sbatch kick out your job after a certain amount of time? what if your job was indefinite like hosting a website.

High performance computing systems, like the cluster that we logged into in class are ued for computationally expensive programs that might require more RAM and/or more CPUs in order to run and/or to run for a longer time than your laptop makes sense to leave it in one place.

For example, this is often used in research data analysis, like the genomics data we used today or for training machine learning models.

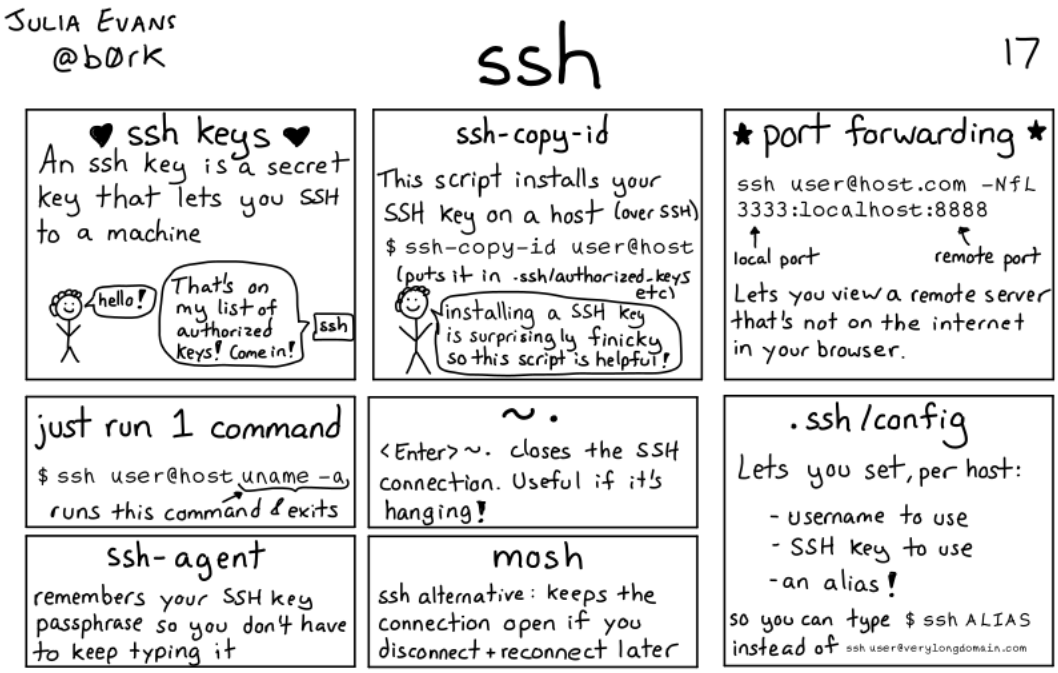
HPCs are a specifc type of remote resource, for indefinite jobs, we use different types of servers. However, `ssh` can still be used to log into those sytems.

14. How does ssh really work? how can I be more secure?

14.1. Using ssh keys to authenticated

1. generate a key pair
2. store the public key on the server
3. Request to login, tell server your public key, get back a session ID from the server
4. if it has that public key, then it generates a random string, encrypts it with your public key and sends it back to your computer
5. On your computer, it decrypts the message + the session ID with your private key then hashes the message and sends it back
6. the server then hashes its copy of the message and session ID and if the hash received and calculated match, then you are loggied in

a toy example



from wizardzines

Lots more networking detals in the full zine available for purchase or I have a copy if you want to borrow it.

this *free zine* describes networks at a lower level full zine but does not include ssh

14.2. Generating a Key Pair

We can use `ssh-keygen` to create a keys.

```
ssh-keygen -f ~/.ssh/seawulf -t rsa -b 4096
```

The `-f` option allows us to specify the file name of the keys.

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/brownsarahm/.ssh/seawulf.
Your public key has been saved in /Users/brownsarahm/.ssh/seawulf.pub.
The key fingerprint is:
SHA256:e1g1ZJA3nlIvMs0mpFG8Sca9z9DQl02N14/Z6Y9Xj70 brownsarahm@152.133.20.172.s.wireless.uri.edu
The key's randomart image is:
+---[RSA 4096]----+
|         +o+o. o*|
|        . Bo* o.*|
|         * OoB *o|
|        . B.X.=.o|
|        S .* =.  |
|         +    o..|
|        o .    ++|
|         .    o =|
|               E.|
+----[SHA256]-----+
```

14.3. Sending the public key to a server

```
ssh-copy-id -i ~/.ssh/seawulf brownsarahm@seawulf.uri.edu
```

and we see it working :

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/Users/brownsarahm/.ssh/seawulf.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
```

Then you have to authenticate with your password, because that is what was set up before.

```
Number of key(s) added:        1

Now try logging into the machine, with:   "ssh 'brownsarahm@seawulf.uri.edu'"
and check to make sure that only the key(s) you wanted were added.
```

To login without usng a password you have to tell ssh which key to use:

```
ssh -i ~/.ssh/seawulf brownsarahm@seawulf.uri.edu
```

Or you can add the following to your `~/.ssh/config` file

```
Host seawulf
  Hostname seawulf.uri.edu
  Username brownsarahm
  IdentityFile ~/.ssh/seawulf
```

and then use

```
ssh seawulf
```

14.4. Review: Undoing half a commit

We can undo a commit using revert, but that applies the opposite of all changes made with that commit. What can we do to undo one of the changes, but not the others?

We need a new commit that has only the changes we actually want to revert.

We can checkout the new commit and then reset the head. Then we have the changes in question in the staging area ready for commit. Now we can commit only the one we want to do in a new commit.

Then we can checkout the main branch, and use git revert to appy the opposite of the thing we want to undo.

14.5. Review: Grep, head, and tail to keep excerpts of a file

```
git add .
git commit -m 'export'
```

```
[nots38 3831319] export
g 1 file changed, 579569 insertions(+)
 create mode 100644 notes/2022-03-08.md
```

When I pushed it though, is where I realized my problem:

```
$ git push --set-upstream origin nots38
```

It looked normal at first

```
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 4.43 MiB | 4.17 MiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote: warning: See http://git.io/iEPt8g for more information.
remote: warning: File notes/2022-03-08.md is 78.68 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: warning: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.
remote:
remote: Create a pull request for 'nots38' on GitHub by visiting:
remote:      https://github.com/introcompsys/spring2022/pull/new/nots38
remote:
To https://github.com/introcompsys/spring2022.git
 * [new branch]      nots38 -> nots38
Branch 'nots38' set up to track remote branch 'nots38' from 'origin'.
```

but the message reminds me that I added a very large file and it took longer than normal. That file is 78.68 MB and GitHub prefers that you not track files larger than 50MB.

However, I knew that the large output was caused by a specific command, so I used `grep` to find what line of the file that occured.

```
grep -n "cat dmel-all-r6.19.gtf" notes/2022-03-08.md
```

```
76:[brownsarahm@seawulf ~]$ cat dmel-all-r6.19.gtf
```

So then I used `head` to take the lines above that plus one below it into a new file

```
head -n 77 notes/2022-03-08.md >> notes/2022-03-08-1.md
```

Then I found the first time we used `head` in class because I knew that was next

```
grep -n "head" notes/2022-03-08.md
```

```
542125:[brownsarahm@seawulf ~]$ head dmel-all-r6.19.gtf
542194:[brownsarahm@seawulf ~]$ head dmel-all-r6.19.gtf
```

ANd used tail to write the file from that line to the end to a new file.

```
tail -n +542124 notes/2022-03-08.md >> notes/2022-03-08-tail.md
```

the `-n` parameter allows you to say how manny lines, but if you put a + before the number it starts *from* that line to end instead of specifying how many lines.

Then I knew that the next long thing was when we used grep on mRNA, so I found that line:

```
grep -n "grep" notes/2022-03-08-tail.md
```

Since I didn't use mrna in the grep, I got three results:

```
70:[brownsarahm@seawulf ~]$ grep Act5c dmel-all-r6.19.gtf
82:[brownsarahm@seawulf ~]$ grep mRNA dmel-all-r6.19.gtf
34108:[brownsarahm@seawulf ~]$ grep mRNA dmel-all-r6.19.gtf | wc -l
```

And again I cna take the head of the file and write it to a new file.

```
head -n 83 notes/2022-03-08-tail.md > notes/2022-03-08-2.md
```

From that same last grep I also knew what the new end of file was:

```
tail -n +34107 notes/2022-03-08-tail.md > notes/2022-03-08-t2.md
```

I thought this might be done, but I checked:

```
wc -l notes/2022-03-08-t2.md
    3340 notes/2022-03-08-t2.md
```

that was still a lot of lines, and then I remembered we had a lot of output from `lshw`

```
grep -n lshw notes/2022-03-08-t2.md
```

which gave one result

```
152:[I have no name!@n005 ~]$ lshw
```

that told me the third section f the file I wanted:

```
head -n 153 notes/2022-03-08-t2.md > notes/2022-03-08-3.md
```

And I knew after `lshw` we exited the compute node:

```
grep -n exit notes/2022-03-08-t2.md
```

```
3323:[I have no name!@n005 ~]$ exit
3337:[brownsarahm@seawulf ~]$ exit
```

Which allowed me to create the fourth section that I needed:

```
tail -n +3321 notes/2022-03-08-t2.md > notes/2022-03-08-4.md
```

Next I removed the original file from git without deleting it locally

```
git reset --mixed HEAD~1
git status
```

git `rm` did not work because that left the file hashed into the git directory and I wanted to fully remove it. SO I reset the head back on setup, to undo the last commit. `--mixed` makes it delete from the repo for real, but not delete locally.

Now I see that it doesn't know anything about that file.

```
On branch nots38
Your branch is behind 'origin/nots38' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        notes/2022-03-08-1.md
        notes/2022-03-08-2.md
        notes/2022-03-08-3.md
        notes/2022-03-08-4.md
        notes/2022-03-08-t2.md
        notes/2022-03-08-tail.md
        notes/2022-03-08.md
        notes/2022-03-08cmdonly.md
```

Then I renamed the original file:

```
mv notes/2022-03-08.md notes/2022-03-08-all.md
```

I tried combinging them with echo:

```
echo notes/2022-03-08-1.md >> notes/2022-03-08.md
echo notes/2022-03-08-2.md >> notes/2022-03-08.md
echo notes/2022-03-08-3.md >> notes/2022-03-08.md
echo notes/2022-03-08-4.md >> notes/2022-03-08.md
```

And confirmed the number of lines in the combined file

```
wc -l notes/2022-03-08.md
```

Which is how I learned that did not works

```
       4 notes/2022-03-08.md
```

Then I remembered to use `cat`

```
cat notes/2022-03-08-1.md >> notes/2022-03-08.md
cat notes/2022-03-08-2.md >> notes/2022-03-08.md
cat notes/2022-03-08-3.md >> notes/2022-03-08.md
cat notes/2022-03-08-4.md >> notes/2022-03-08.md
```

Again checking the number of lines to show that it worked:

```
wc -l notes/2022-03-08.md
```

```
     333 notes/2022-03-08.md
```

and for rference comapred to the original

```
wc -l notes/2022-03-08-all.md
```

and we see the original length

```
  579569 notes/2022-03-08-all.md
```

Then I looked at the compiled file and deleted the others once I was sure I had what I needed.

14.6. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Do what you need to do to finish the semester well: eg rest, catchup on work, both (breaks are important)
2. Review your IDE thoughts right before class on Tuesday March 22 (so you're ready for the activity in class)

14.7. More Practice

1. **optional** Configure ssh keys to your GitHub account (this is actually GitHub's preferred terminal authentication method)

14.8. Questions after class

15. What is an IDE

read about IDEs

compare IDEs

most popular IDEs

15.1. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Explore the IDE you use most and add `ide.md` to your kwl with notes about which features it does/not have. (to review/reinforce)
2. **priority** Make a table comparing two programming languages you know in `languages.md`. The table should have 3 columns: one titled "Attributes" and one for each languages. Add one row for each attribute and fill in how that is for each language. Use the following list of attributes and add 1-3 more.

```
create a list
loop (with predetermined number of iterations)
loop (until a condition is met)
conditional
create a string variable
cast from one type to another
```

3. read about the parts of the developer survey about languages

15.2. More Practice

1. (priority) Compare at least 3 IDEs for working in a single language. Create `favoriteide.md` and include your most important criteria with their rankings, how each IDE meets/does not meet those criteria, and a conclusion of which IDE is the best based on your criteria.

16. How do we pick programming languages?

16.1. Admin

- make sure you have a PR open for any work you have not received feedback on

16.1.1. What *is* a pull request ?

like what say you were working on something you made some changes and then

you merge before class but then you made new changes on that Branch then you can make a Newport I sent the request from that bridge is closed but you have to it like

you can always do that right so you have to just keep things kind of separate so until it branches merged you can't make an additional per request from it because it's it's like

but it's not like a part of us isn't the thing that exists inside repository right branches exist inside their policies when you download with get clone like I'm working out so you can see all the branches you have but the poor West isn't there the poor of us is just

it's like a helper for a person to do it once a mergency branches or merge this one into the other like we talked about get work clothes right and there was like an iteration manager version on so get its actual Source forget itself

is actually mirrored on

right to have an organization the mirror that store still here but this is just a publish only depository that they don't actually use pull requests to hear people don't follow instructions and make them but they don't actually use the four of us to decide what it says yet to submit which of these approaches to the mailing list so if you want to submit a pull request to submit changes to the get source code that actually have an email

yes I do know there's that's their way of kind of limiting who's going to do it and who's changes are going to consider seriously because get invented like GitHub cannot exist until so get predates GitHub

16.2. IDE Review

so last week I had you read about IDEs is it going to see will come back in touch and reinforce some of those things in different classes of you talk about how like what actually happens to code cuz that's what's going to happen to the rest of the month or so the first part we talked a lot about the tools we use

but we we just been playing with text files because they're easy and they can do we can treat them just like weird code files what

happens with a code when you bought it starting from like how does comp compilation work all the way down to

like how our numbers are presented in computers machine representation. so

Wii U read about ideas but let's just a quick review to make sure they're paying attention and you're awake and here so first question or piss me off just what is it stand for

16.2.1. What does IDE stand for?

Integrated Development Environment

16.2.2. Why do we use them?

To keep all of the tools together, increase ease of using multiple together.

Also provide additional features when it's a development focused environment:

- easier path handling in terminal
- developer friendly features in text editor (mulitple cursors; copy+ paste at once; tab completion; syntax checking)
- having to file explorer panel while you're looking at your code helps you see what's going on and see for example a function definition at the same time as where you're calling it.

The More practice from last week is expected to be a longer one.

yeah right if it combines two things are copy and paste and do just one thing

well right. Really sophisticated phone replaced if I can see mine is here

this is what I was doing when I was editing notes after class cuz my nose when I export them have like the it has electronic don't like the brown there I'm at whatever for my criminals I don't take that other line but you can also

16.3. How can we compare Different Programming Languages

matter they very what are they share what sort of a structure a level not like any specific language in detail but like what is it what do you think about your choosing a program or you're trying to understand what happens cuz we need to think about this talk a little bit to set the stage for what happens in a in a specific case of how do we go from high-level code that's read of relative. What you don't see might be a little at the lower-level still have a really good but it's lower than that to the zeros and ones that are computers going to run out eventually

so let's start brainstorming and what are some different things you could use to compare

- Garbage collection

if you stop using something or you you don't need it anymore does it keep using resources until you manully, in your code tells it okay it's not that bad or does your the language know to okay this is we're done using this let's stop what's another thing

- static vs dynamic typing and memory allocation

number yeah so can you change the type of a variable on the fly or do you have to Define it like you to declare what type of variable is going to be so that it can allocate the memory in the right format for that variable and then it's stuck is that type you can't change it until you unless you re initiating are you can't it's not going to be there what else Christian

- syntax

white space use syntax specifically other like what type of bracket to use what types of whatever you use and I tried to add more things to my list and instead I sent it so now my list is there but we can keep going what are you doing

- programming Paradigm

whether you're focusing on like everything has to be declared as an object whether it's just like you have some functions and some scripting type things do some programming that support sort of really sophisticated data mapping structures or language is coming with you. I like python you can do however you want like it has a lot of functional features it has object-oriented it can find a support doing a lot of the different things in Python is added more functional feature support right there making it more supportive of those actions because they're they're powerful and they're really good for the type of thing to do would like to use Python for what else

16.4. how do you decided to pick one language vs another?

- API support to access necessary resources/hardware
- specialization of the language (some are built for specific purposes)

yeah so why don't you talk about so far have been really technical details like does it actually do the thing you wanted to do but there's also could be easier to learn the new language or there's also star in community aspects are the amount of users of language so there's a language called Julia that I think it came out like in like late 2010 2011 is somewhere around there maybe even a little bit later it was very very very powerful to do the types of research like the types of work that I was doing my butt is very very new so didn't have a lot of documentation it didn't have a lot of people using it

even though it was like quite appealing for a lot of things I was going to do it would have been more performance they were doing better before his life than python I will not let that up to that I ended up using I just not usually up because there are no libraries like I was going to implement basic things that I did not have to implement in Python because I could use libraries and python

1/2

what

actually do that with the Limitless

is that is having her list that is that is about choice so I'm going to

have you review this developer survey after I send I want you to read through for a couple minutes and pick the one thing you most you find most interesting you want to discuss and I will be right back

all right what is the most interesting thing you found on this Saturday

going to wait till someone else that has not spoken yet wants this week I'll get to the three of you but Tyler

looking for

that was really not that bad

so it's good that you're not loud is what it's supposed to be used for if you're doing linear algebra nothing else like all the rest of these are terrible but yes so there could be some explanations are what else

45% Windows users I don't know what text what section are they in

are there that's okay yeah a lot of surprising

you don't know

to the equal when it's Mason. OS a very small percentage of people using this what album is this what if we switch from all respondents to professional golfers it's all just a little bit to get more math less less than a Caesars

and this is still 57,000 people so it's not like a smaller number but not his number this is still what

most corporations is just what they give you if you work at a big Corporate Place like that's what they give you so you're not really this is really getting to choose so even though like and I could be me and you won't see as much people do people sell Windows 7 starter it's still realistic like that's what people get also if you want your stuff to run on Windows you have to make it work there

there has to be some amount of thought this Sabrina

yeah so it's been for a long time most loud but people actually using it as is smaller yeah some people that use it love it also if we will get people that want to use it it's still pretty high right through the do not use it but wish they could like would like to you so pretty high

Christian

send the next question

I'm just trying to find it so yeah this is this is a good thing obviously this was a good thing for us to be learning cuz don't have a version of the schedule and get is by far the most everyone using it of the respond to the survey this is so this is which is students as well

especially about the same does it mean when I smile doctors

sounds like a very hyper standardized it's kind of like a virtual machine almost so I'm guessing more of your views

so, basically

avocado pie what when you use it so

let's say you're doing with the reason the way I come into contact with people who are doing who produce the project maybe they designs project they use a whole bunch of libraries a whole bunch of installed that got everything working and then they want someone else to be able to use it to how many times have you set up a programming environment and there's a lot of insulation possibly as much insulation is trying to use the place actually build your thing at all

Rite insulation give me really hard and really annoying so doctor makes that easy to the doctor container specifies all of the requirements and stuff them all up it also allows you to have some multiple containers on your computer so you can have one environment for doing this project another event for doing a project that don't interact with each other and to doctors like a standard and a technology to help support that

badass more than 100

yeah so that the people could check more than one so it's just like

they allowed people to check as many as they want and they could even check both of they use it and they want to use Technologies

these are computers just like these are the most popular things at all communities is another like platform tool for like spitting things out separately I think he's going to take a package I think it's a package manager as well I have a feeling it might be related to

are or like publishing but clearly not crapping

yeah and install it's idiot Auntie I'm so it's a Java Java Java I forgot those 10 I don't either then when I choose to do so but it allows you to packaging and what else

you can get paid to go to school

oh yeah if you like probably like it so if you're like a managing service systems also we have a bunch of a do those kind of things different trying to hire someone who's going to manage their servers they'll be doing other things but only programming are probably doing it on his way back type things

I cannot this family was supposed to be there but the faster you don't know about it it's probably directly related to why it pays higher cuz things that people don't know get paid more

yeah

and then things on there so this is an artifact of a large percentage of people doing this or grad students

once you've done enough times with no longer that little crisis here at this is just what it is we truncate the list

eventually so we don't put every possible cuz out of 80 thousand people I'm sure unless it was other things arranged thing

why is it useful to read through these results why is it something that is worth spending time on a glass Tyler

yep he’s a home from other developers you get to know kind of what’s out there

yeah if you’re trying to decide what things you want to run outside of time that’s important also knowing kind of how different programming languages compared we saw we saw a list of Unforgiven images Nancy hadn’t heard of the four of us out there and if you don’t know it in great detail helps you kind of get the landscape of what’s out there and how different things are changing and how they interact with another also this sets the stage for the fact that we’re not talking about things that are just like cuz I feel like it

or like giving things are actually important that that are used in industry and that are that are useful so

it was another question so if you go back to that list of

difference things we can compare foreign languages on we’re trying to start thinking about using a project

how

what would your first step be if you were trying to pick what language to use for new project

figure out what your needs are for you what’s there

writing a what labor jobs

so when

but for example

if we talked about a linkage being compiled versus interpreted so a compiled language you write your code you hand it over to compiler and then it gets a machine code I’m about 100 over and over again and interpreted language you write your coat and head to The Interpreter The Interpreter runs and gives you the results back you can run one line at a time you can kind of you but each time you want to run it it’s doing any work that’s compilation like sort of on the flight as you run it right

when would you want one or the other

okay so

which one of those like if you so you’re saying whether the data that the program is going to use is going to be variable you might want to choose

okay so what’s on tap at typing with python although it is and then there’s static typing which see Elsa is but separate from that she has compiled do it like you get a bill to do that and then python is interpreted so you give him the code of The Interpreter and it runs right one at a time either of those could actually handle a large amount of data or variable size data right cuz you can

always

like Rita file and get the information then allocate besides that’s not going to change that but would be different

indeed.com

python

I don’t know what it is

oh oh cuz you can’t like use a class object inside of a class definition okay

yeah

okay so that would be a case for if you need to make kind of that type of nothing but a compiled language can do that but interpreted

I didn’t cannot that’s good example okay what else would you want one with the other compiled verse Jeopardy Sabrina

so if you mean something resource-intensive you want to be compiled so if you’re reading something that’s going to run like on a small device like a Fitbit or a smartwatch write those don’t have a lot of memory didn’t have a lot of things he’s going to find out where do I have a person I know who works at

did he propose in libraries part of NASA her job for awhile was too during the 17-minute window that happens twice a day send code to the Mars rover they can't send pythons are there right they can't let you know the river is is very far away that was 70 minute window to transmit things over a slow satellite link they have like KB a day they can send

price of a need cuz it's going to compress down to the thing also the how long ago to the Rover leave Earth I don't know if you know but as long as you know the actual time

great it's been gone for a long time so Hardware that we have now is very different than what was available and able to end amount of time before it actually left is what it was designed

so the hardware that has is very old for a computer hardware so they're very stress and strain but they're still want to like do scientific data analysis on another planet which is when

oh okay yeah there's everyone has been gone for a while

okay the last time I talk to this person was an in-person conference which was definitely in 2019 or early 2018 so

okay

I thought everything was old has a very resources and environment they still need to be able to do real things right it's not a toy problem is not a we're just playing with an Arduino because we feel like it it's a real thing right it's a real real where they had actual Hardware constraints they had to have very a pin code to decide how to collect the data and then how to get it back they know that 17 minute window to happen twice a day they can send its also receive other data back about her experiments in that time but if you really want to send and want to get data back

so we had to have really efficient code to get in the house so we've got some cases for when you would want them piled what made you want interpreted language

when does it become better for you

I mean you have to have you might have to have that but what would what would give you the advantage out of it

you're saying

you have to do is lovely all right cuz you don't to wait for it to compile and when would you why would you not want to wait for some pile you need to physically Tyler

program

yeah so with four examples for every type of Hardware that you want to run on right but interpreted language

someone has to make The Interpreter compiled for every single part where you want to run on but you the developer that language don't have to make your hair different right there's a few things for gambling python you can't use you can't hardcore pads with boards lashes because Windows use the other / but other than that you can run your code on everything you want your language help you out a lot

and you have made changes and I looked and test it and you're going to it's going to be a lot harder to take a little while tomorrow and you put her in it

yeah when you're experimenting or take ring or the reason I use Python is that you did an outfit I need to I don't know what the third step is going to be in till I see the results on the 2nd

so I need to run over s up get the results and then decide what to do next cuz I'm looking at the data processing and I'm calculating deciding what to do next so since I can't plan on my advance

compiled language doesn't make sense right

Oso

Sophia pass in San Antonio not Windows uses for it in Windows uses backsplashes and pads so you have to not talk with your path and python dissolution is there's a library called OS and you use OS taco joint

that join to put them together instead of typing the splashes because that it makes it Go your problems go away that is a problem that I learned through a lot of pain and suffering so isn't that I know well

what else am I to you

or any other reasons you might want to choose interpreted first compiled

all right so if you were complaining that you would want to use which kind interpret it right cuz you're trying to see if the thing can work you just want to get a really good result and you're not trying to package it up into everywhere

another case for when people would prefer compiled languages is if you want your source code to be private if you don't want to distribute your source code you have to use a compiled language I got to give it compiled down to the binaries and distribute the binaries right

that's my knowledge now

you can like

I'm just kidding I think but I don't think like is actually private

what you cannot see the code the source code to recover the source code from the banners I mean you can't recover it precisely because like you could write code lots of different ways that results in the same binary straight like you could write an arbitrary number of different thing that got to the same thing

there's a certain amount of you can like

pack together things but it's not true it's not trivial and not easy I believe that's a thing that like a certain type of researcher like attempted to do but it's not something I ever has drama divisions to do

so becomes really really hard to trace out the Kodak Center like recover at a high-level from the binary is because everything is compiled down in Pacific cases

right so yeah it is very good but you might be able to cover a certain amount of it

I know there's like Steps intermediate steps that that can help cuz like I had

a friend who was

Miss Universe that are working with a professor to help right auto traders because I thought my situation is a thousand students and their and their horses so I think they were doing what they want the cluster student Solutions in group all the teams that made the same mistake together so they could send the same advices is it doesn't seem the same as they were doing sort of like partial compilation to the who record you could tell when seems like even if they use different variable names they can if they have the same sort of bad structure in their Loop or whatever you can tell so there's things that you can do with different languages depending on what it is but that's not

generally easy or like a common thing to do

yeah so people that people do try to prove sort of like that things are the same or equivalent or like comparable to the stop now you're right up the front of the edge of like state-of-the-art kinds of things

Okay so

we talked about what's it like to build and run the environment

we've been to the garbage collection how hard do I tell if it is is a programming language the typing of the static or dynamic availability of like libraries in apis and excessive has access to whatever it is you're trying to enter access

and I touched down briefly but there could be this this factor of Life Community amount of user so other people like how do you actually get help for that language can you understand it like is it written in your human spoken language right some languages have some programming languages have documentation in a lot of different human spoken languages so I'm only in English so if you don't speak English well you can't use of languages

Okay so now that we've

talk about this a little bit let's go through something else so

you

we're going to do a project with

robotics what would you look for in an English

what kinds of things do you look for what would you want it to be like

the speed that needs will run fast on whatever the hardware the robot has cuz your robot may or may not have a powerful like computer or a lot of memory depending on the size and location is going to go

does he like does it have instructions that support the library or Library support like did the people that build that robot give you an API I'd like that to control it

it was a day when you'd like a compiled language you can give the results you give the robot the program you just let it go you don't want to be interacting with the robot sort of like one step at a time necessarily you might want to be able to do that sometimes but really most the time you want to let it go usually

probably able to make me to delete data so you want a language it's not going to like have a lot of bugs cuz you don't want someone accidentally like inserting code and like a feeling you're about to go do bad things that you didn't like to do you want to interact with any data that the robot is using

what about

what kind of language is you think people might use what types of what examples of languages.

Christian

and I think I thought would be good for that because I've heard that it's just like just the kind of thing and also it's easy to like them since August atticly type of language

okay so if your robot was going to do the day that I was just like in the field you might want to use Python what other cuz we talked about it you might want something that's going to be fast and packed I have good libraries for the robots itself what other languages you think you might use with a iRobot

Rikers Island so

its popularity has recent so the disadvantage of the reason I may not work is cuz the the hardware for the robot may not have support for rust

why

but it also

Boulder so compiled languages both good for that CD holders, in another sneaky language in common robotics because

not loud is not open source Wright was willing to do you talk about open source are developed by community of whoever else whatever Matt love is not / coming to go math works and MathWorks works very closely with Hardware manufacturers to make sure that Matlab can work with your Hardware because I want the Dave want people to pay to use their language so they work with people who sell expensive things to make their expensive things compatible with their language

rights to these other kinds of artifacts like a lot of Engineers use Matlab because all of their like different things you plug in that work really well with that right I speak cameras will have an API with Matlab all of these kind of hard because it's like we want to get people to use our language so they lock people into or what will make it easier for you cuz it's going to be easier than C

right it's maybe not as nice as like python but it's going to be way easier than see if you would want to buy that will go up that way to get these weird

MathWorks the company

I mean they can't they can't prevent that in that sounds like it's any damn sure is there and in fact you can actually it's relatively easy to integrate Matlab code whiskey cuz also most other thing a lot of other things are like written with the your compatibility so you'll start to see what a big project actually have multiple project multiple languages involved

Wright's like one of my club jobs I was doing signal processing I was like doing like real time radio stuff so to make the real time part work I had to use could I would you like Siri but for the early for gpus so to write the code to paralyze my functions cuz this was 2010 and no one has written libraries it did that for you and then the main part of the analysis and not lie but I was hoping to be in my life cuz I was working with you is awesome cleaner doing because if you have a really big project just probably going to be different parts of it that different languages are going to be better for and so sometimes it's not worth sacrificing one part for the other part you just right in both languages and select the data back and forth

like other examples are

python is way better at doing like date analysis calculations and like JavaScript write like you don't want to do a lot of math

browser than python you also don't want to like right to be 75 on like no one wants to do that so there's whole framework take data from the browser give it the python get the results back from Python and put them in the browser so they run little web servers just to do that right there makes passing back and forth so you can write in the most the best language for the specific tasks in your project and then he's like packages I'll help you tie it all together to make it work so

was he a machine-learning that's good

so

we're talking about doing

data analysis with python code on because I wanted to do another language really, information on and get as far as the language made by statisticians for statisticians right so

I don't know why someone would want to choose if you're doing something to get analysis when you might want to use Python A R

Library

just talked to your comment and it will calculate you even like my normal distribution table something else cuz it's too many things like specific first a decision that very very pretty close to fix the article will be like half as many lines even in Python with them because I was more powerful things are in Python but if you're trying to do like if you're an ecologist or a biologist try to do that goes those comments at 264 them to me way better does a port there then you get the snacks level where because most likely I'll just call just that her up use are when they develop new techniques novel data analysis techniques there specific strategies for analyzing genetic data for example

they're going to use our to do it they're going to release it as an AR Library do you start having used a virgin Obviously good the specific type of thing you're doing is going to be between which one you're going to use some kind of understanding what the community is and who likes that language often becomes useful and so beaver not that super think technical details we tend to talk about but they're really important things to know about languages and understand so

we will start wrapping up now

one thing you learn today

and 1 questions have you do something your cares about or something you wanted you want Clarity on

see a quick question about internal pandas is a library in Python and it's sort of an acronym

but it's

data analyst for data analysis so

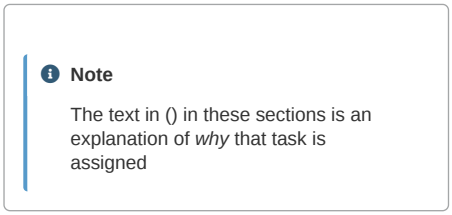
the main thing that pandas does is a python you have like other data types right you have lists and pandas gives you a dataframe or like a table

all the data structure is amazing provides and then all of the methods to do like a lot of basic analysis and plotting so they supposed to just fix I'm plotting and all of those kinds of things like on your I'm grabbing a little about a lot of things it's not so easy to read and write that people don't care that it's actually like very slow because like human time is way more valuable than her time computers have a lot of time people know

ready like undo a part of a carrot yeah actually time to do that actually I had to do it on my device like I do it locally and then push to go and back

I was working on something but the fire was like over 100 MB and I skip up because you can't put that yeah

16.5. Prepare for next class



1. For 2 languages from the loved vs dreaded list (one top 5; one bottom 5) read 2-3 posts about why people love/hate that language and summaries the key points on each side. Add this to your kwl repo in `language_love_dread.md`
2. Make sure that you have a C compiler installed on your computer. ideally gcc

16.6. More Practice

> **Try it Yourself**
>
> use scp to download the files generated to your local computer and add them to your kwl chart. (a possible extenstion idea to practice reading the official documentation to learn a new tool that is similar to ones you have seen before)

1. (priority) Describe a type of project where it would be worth it for you to learn a language you have never used before in `newlanguage.md` This should be based in what types of features for the language your project would require and/or what would contribute to the long term health of the project.
2. Try out/learn about one of the following languages that you have not used before, do something small that is typical of that language (eg a toy data analysis in R or): R, Julia, Clojure, Stan, Go, . Answer the following questions:

```
1. What is this language designed for?
1. What Progrmaming paradigm(s) does it support?
1. What about it makes it easy to learn for someone who already knows some
other language (name that language)?
1. What about it is hard to larn for (pick a language) prgorammers?
1. What is its most unique feature(s)?
```

16.7. Questions After class

16.7.1. What is pandas?

pandas is python data analysis library

16.7.2. How do you implement programs/projects which depend on multiple languages, like a web browser that uses javascript but does calculations in python.

You keep all of the files together and set up build tools to build as needed

16.7.3. Where might one person go to find out which programming languages is best for their project? Is it through personal research?

It is through learning about through networking and keeping up with trends. It's also through trying new things. As you know more, you build intuition about what to do and sometimes just talking to others.

16.7.4. Are the languages used for classes arbitrarily chosen or is there a process that the professor goes through when picking the language for a class?

We, as faculty choose together a lot actually, based on the concepts we want to teach and what language features that requires.

We also keep up to date with CS Education literature which studies how different languages impact learning.

16.7.5. What kind of non technical questions should I be asking when developing?

This is a *great* question, but beyond the scope of this course to cover it entirely. CSC 320 would help give you more ideas and context for this.

17. What happens when I build code in C?

We came across the term build before.

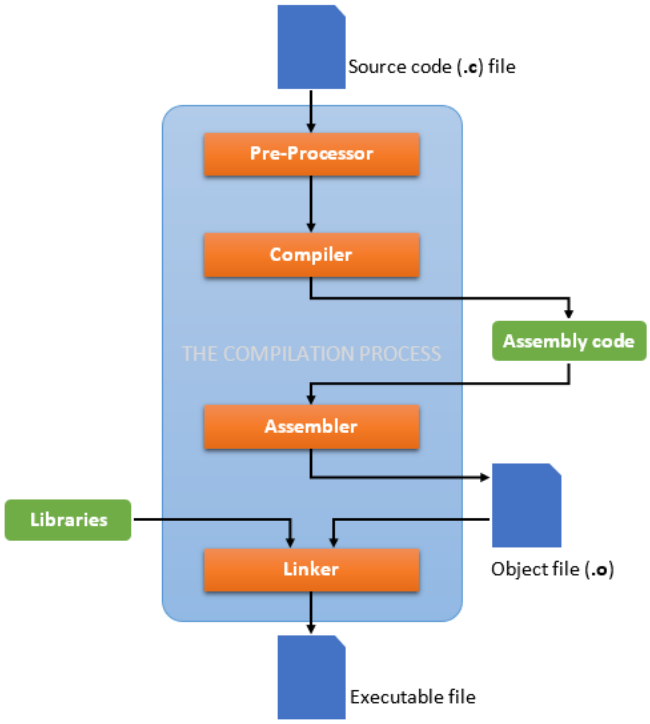
We sometimes say that compiling takes code from source to executable, but this process is actually multiple stages and compiling is *one* of those steps.

We will focus on *what* has to happen more than *how* it all happens.

CSC301, 402, 501, 502 go into greater detail on how languages work.

Our goal is to:

- (where applicable) give you a preview
- get enough understanding of what happens to know where to look when debugging



source

17.1. Setup

Let's setup by creating an empty folder

```
cd Documents/sysinclass/
mkdir compilec
cd compilec/
```

and then create a small program in C.

```
nano hello.c
```

and then we can look at the file

```
cat hello.c
```

```
#include <stdio.h>

void main () {
 printf("Hello world\n");
}
```

17.2. Preprocessing

First we handle the preprocessing which pulls in headers that are included. We will use the compiler gcc

```
gcc -E hello.c -o hello.i
```

If it succeeds, we see no output, but we can check the folder

```
ls
```

now we have a new file

```
hello.c hello.i
```

```
cat hello.i
```

This gives us a version with the header file's contents literally pasted in to replace the original `#include` statement

```
# 1 "hello.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 366 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "hello.c" 2
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 1 3 4
# 64 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 1 3 4
# 68 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 1 3 4
# 630 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_symbol_aliasing.h" 1 3 4
# 631 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 2 3 4
# 696 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_posix_availability.h" 1 3 4
# 697 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/cdefs.h" 2 3 4
# 69 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/Availability.h" 1 3 4
# 259 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/Availability.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/AvailabilityInternal.h" 1 3 4
# 260 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/Availability.h" 2 3 4
# 70 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4

# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_types.h" 1 3 4
# 27 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_types.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types.h" 1 3 4
# 33 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/_types.h" 1 3 4
# 32 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/_types.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/_types.h" 1 3 4
# 37 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/_types.h" 3 4
typedef signed char __int8_t;



typedef unsigned char __uint8_t;
typedef short __int16_t;
typedef unsigned short __uint16_t;
typedef int __int32_t;
typedef unsigned int __uint32_t;
typedef long long __int64_t;
typedef unsigned long long __uint64_t;

typedef long __darwin_intptr_t;
typedef unsigned int __darwin_natural_t;
# 70 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/_types.h" 3 4
typedef int __darwin_ct_rune_t;




typedef union {
 char __mbstate8[128];
 long long _mbstateL;
} __mbstate_t;

typedef __mbstate_t __darwin_mbstate_t;


typedef long int __darwin_ptrdiff_t;







typedef long unsigned int __darwin_size_t;




typedef __builtin_va_list __darwin_va_list;




typedef int __darwin_wchar_t;
```

```
typedef __darwin_wchar_t __darwin_rune_t;


typedef int __darwin_wint_t;




typedef unsigned long __darwin_clock_t;
typedef __uint32_t __darwin_socklen_t;
typedef long __darwin_ssize_t;
typedef long __darwin_time_t;
# 33 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/_types.h" 2 3 4
# 34 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types.h" 2 3 4
# 55 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types.h" 3 4
typedef __int64_t __darwin_blkcnt_t;
typedef __int32_t __darwin_blksize_t;
typedef __int32_t __darwin_dev_t;
typedef unsigned int __darwin_fsblkcnt_t;
typedef unsigned int __darwin_fsfilcnt_t;
typedef __uint32_t __darwin_gid_t;
typedef __uint32_t __darwin_id_t;
typedef __uint64_t __darwin_ino64_t;

typedef __darwin_ino64_t __darwin_ino_t;



typedef __darwin_natural_t __darwin_mach_port_name_t;
typedef __darwin_mach_port_name_t __darwin_mach_port_t;
typedef __uint16_t __darwin_mode_t;
typedef __int64_t __darwin_off_t;
typedef __int32_t __darwin_pid_t;
typedef __uint32_t __darwin_sigset_t;
typedef __int32_t __darwin_suseconds_t;
typedef __uint32_t __darwin_uid_t;
typedef __uint32_t __darwin_useconds_t;
typedef unsigned char __darwin_uuid_t[16];
typedef char __darwin_uuid_string_t[37];


# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_pthread/_pthread_types.h" 1 3 4
# 57 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_pthread/_pthread_types.h" 3 4
struct __darwin_pthread_handler_rec {
 void (*__routine)(void *);
 void *__arg;
 struct __darwin_pthread_handler_rec *__next;
};

struct _opaque_pthread_attr_t {
 long __sig;
 char __opaque[56];
};

struct _opaque_pthread_cond_t {
 long __sig;
 char __opaque[40];
};

struct _opaque_pthread_condattr_t {
 long __sig;
 char __opaque[8];
};

struct _opaque_pthread_mutex_t {
 long __sig;
 char __opaque[56];
};

struct _opaque_pthread_mutexattr_t {
 long __sig;
 char __opaque[8];
};

struct _opaque_pthread_once_t {
 long __sig;
 char __opaque[8];
};

struct _opaque_pthread_rwlock_t {
 long __sig;
 char __opaque[192];
};

struct _opaque_pthread_rwlockattr_t {
 long __sig;
 char __opaque[16];
```

```
};

struct _opaque_pthread_t {
 long __sig;
 struct __darwin_pthread_handler_rec *__cleanup_stack;
 char __opaque[8176];
};

typedef struct _opaque_pthread_attr_t __darwin_pthread_attr_t;
typedef struct _opaque_pthread_cond_t __darwin_pthread_cond_t;
typedef struct _opaque_pthread_condattr_t __darwin_pthread_condattr_t;
typedef unsigned long __darwin_pthread_key_t;
typedef struct _opaque_pthread_mutex_t __darwin_pthread_mutex_t;
typedef struct _opaque_pthread_mutexattr_t __darwin_pthread_mutexattr_t;
typedef struct _opaque_pthread_once_t __darwin_pthread_once_t;
typedef struct _opaque_pthread_rwlock_t __darwin_pthread_rwlock_t;
typedef struct _opaque_pthread_rwlockattr_t __darwin_pthread_rwlockattr_t;
typedef struct _opaque_pthread_t *__darwin_pthread_t;
# 81 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types.h" 2 3 4
# 28 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_types.h" 2 3 4
# 40 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_types.h" 3 4
typedef int __darwin_nl_item;
typedef int __darwin_wctrans_t;

typedef __uint32_t __darwin_wctype_t;
# 72 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4



# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_va_list.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_va_list.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/types.h" 1 3 4
# 35 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/types.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 1 3 4
# 76 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int8_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int8_t.h" 3 4
typedef signed char int8_t;
# 77 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int16_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int16_t.h" 3 4
typedef short int16_t;
# 78 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int32_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int32_t.h" 3 4
typedef int int32_t;
# 79 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int64_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_int64_t.h" 3 4
typedef long long int64_t;
# 80 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4

# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int8_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int8_t.h" 3 4
typedef unsigned char u_int8_t;
# 82 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int16_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int16_t.h" 3 4
typedef unsigned short u_int16_t;
# 83 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int32_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int32_t.h" 3 4
typedef unsigned int u_int32_t;
# 84 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int64_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_u_int64_t.h" 3 4
typedef unsigned long long u_int64_t;
# 85 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4


typedef int64_t register_t;




# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_intptr_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_intptr_t.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/types.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_intptr_t.h" 2 3 4

typedef __darwin_intptr_t intptr_t;
# 93 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_uintptr_t.h" 1 3 4
# 30 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_uintptr_t.h" 3 4
typedef unsigned long uintptr_t;
# 94 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/i386/types.h" 2 3 4
```

```
typedef u_int64_t user_addr_t;
typedef u_int64_t user_size_t;
typedef int64_t user_ssize_t;
typedef int64_t user_long_t;
typedef u_int64_t user_ulong_t;
typedef int64_t user_time_t;
typedef int64_t user_off_t;
```

```
typedef u_int64_t syscall_arg_t;
# 36 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/machine/types.h" 2 3 4
# 32 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_va_list.h" 2 3 4
typedef __darwin_va_list va_list;
# 76 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_size_t.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_size_t.h" 3 4
typedef __darwin_size_t size_t;
# 77 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_null.h" 1 3 4
# 78 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4

# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/stdio.h" 1 3 4
# 39 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/stdio.h" 3 4
int renameat(int, const char *, int, const char *) __attribute__((availability(macosx,introduced=10.10)));
```

```
int renamex_np(const char *, const char *, unsigned int) __attribute__((availability(macosx,introduced=10.12)))
__attribute__((availability(ios,introduced=10.0))) __attribute__((availability(tvos,introduced=10.0)))
__attribute__((availability(watchos,introduced=3.0)));
int renameatx_np(int, const char *, int, const char *, unsigned int) __attribute__((availability(macosx,introduced=10.12)))
__attribute__((availability(ios,introduced=10.0))) __attribute__((availability(tvos,introduced=10.0)))
__attribute__((availability(watchos,introduced=3.0)));
# 80 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 2 3 4

typedef __darwin_off_t fpos_t;
# 92 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 3 4
struct __sbuf {
 unsigned char *_base;
 int _size;
};


struct __sFILEX;
# 126 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_stdio.h" 3 4
typedef struct __sFILE {
 unsigned char *_p;
 int _r;
 int _w;
 short _flags;
 short _file;
 struct __sbuf _bf;
 int _lbfsize;


 void *_cookie;
 int (* _Nullable _close)(void *);
 int (* _Nullable _read) (void *, char *, int);
 fpos_t (* _Nullable _seek) (void *, fpos_t, int);
 int (* _Nullable _write)(void *, const char *, int);


 struct __sbuf _ub;
 struct __sFILEX *_extra;
 int _ur;


 unsigned char _ubuf[3];
 unsigned char _nbuf[1];


 struct __sbuf _lb;


 int _blksize;
 fpos_t _offset;
```

```
} FILE;
# 65 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4


extern FILE *__stdinp;
extern FILE *__stdoutp;
extern FILE *__stderrp;
# 142 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
void clearerr(FILE *);
int fclose(FILE *);
int feof(FILE *);
int ferror(FILE *);
int fflush(FILE *);
int fgetc(FILE *);
int fgetpos(FILE * restrict, fpos_t *);
char *fgets(char * restrict, int, FILE *);



FILE *fopen(const char * restrict __filename, const char * restrict __mode) __asm("_" "fopen" );

int fprintf(FILE * restrict, const char * restrict, ...) __attribute__((__format__ (__printf__, 2, 3)));
int fputc(int, FILE *);
int fputs(const char * restrict, FILE * restrict) __asm("_" "fputs" );
size_t fread(void * restrict __ptr, size_t __size, size_t __nitems, FILE * restrict __stream);
FILE *freopen(const char * restrict, const char * restrict,
              FILE * restrict) __asm("_" "freopen" );
int fscanf(FILE * restrict, const char * restrict, ...) __attribute__((__format__ (__scanf__, 2, 3)));
int fseek(FILE *, long, int);
int fsetpos(FILE *, const fpos_t *);
long ftell(FILE *);
size_t fwrite(const void * restrict __ptr, size_t __size, size_t __nitems, FILE * restrict __stream) __asm("_" "fwrite" );
int getc(FILE *);
int getchar(void);
char *gets(char *);
void perror(const char *) __attribute__((__cold__));
int printf(const char * restrict, ...) __attribute__((__format__ (__printf__, 1, 2)));
int putc(int, FILE *);
int putchar(int);
int puts(const char *);
int remove(const char *);
int rename (const char *__old, const char *__new);
void rewind(FILE *);
int scanf(const char * restrict, ...) __attribute__((__format__ (__scanf__, 1, 2)));
void setbuf(FILE * restrict, char * restrict);
int setvbuf(FILE * restrict, char * restrict, int, size_t);
int sprintf(char * restrict, const char * restrict, ...) __attribute__((__format__ (__printf__, 2, 3)))
__attribute__((__availability__(swift, unavailable, message="Use snprintf instead.")));
int sscanf(const char * restrict, const char * restrict, ...) __attribute__((__format__ (__scanf__, 2, 3)));
FILE *tmpfile(void);

__attribute__((__availability__(swift, unavailable, message="Use mkstemp(3) instead.")))

__attribute__((__deprecated__("This function is provided for compatibility reasons only.  Due to security concerns inherent in
the design of tmpnam(3), it is highly recommended that you use mkstemp(3) instead.")))

char *tmpnam(char *);
int ungetc(int, FILE *);
int vfprintf(FILE * restrict, const char * restrict, va_list) __attribute__((__format__ (__printf__, 2, 0)));
int vprintf(const char * restrict, va_list) __attribute__((__format__ (__printf__, 1, 0)));
int vsprintf(char * restrict, const char * restrict, va_list) __attribute__((__format__ (__printf__, 2, 0)))
__attribute__((__availability__(swift, unavailable, message="Use vsnprintf instead.")));
# 205 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_ctermid.h" 1 3 4
# 26 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/_ctermid.h" 3 4
char *ctermid(char *);
# 206 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4



FILE *fdopen(int, const char *) __asm("_" "fdopen" );

int fileno(FILE *);
# 228 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
int pclose(FILE *) __attribute__((__availability__(swift, unavailable, message="Use posix_spawn APIs or NSTask instead.")));



FILE *popen(const char *, const char *) __asm("_" "popen" ) __attribute__((__availability__(swift, unavailable, message="Use
posix_spawn APIs or NSTask instead.")));
# 249 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
int __srget(FILE *);
int __svfscanf(FILE *, const char *, va_list) __attribute__((__format__ (__scanf__, 2, 0)));
int __swbuf(int, FILE *);
# 260 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
inline __attribute__ ((__always_inline__)) int __sputc(int _c, FILE *_p) {
```

```
 if (--_p->_w >= 0 || (_p->_w >= _p->_lbfsize && (char)_c != '\n'))
  return (*_p->_p++ = _c);
 else
  return (__swbuf(_c, _p));
}
# 286 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
void flockfile(FILE *);
int ftrylockfile(FILE *);
void funlockfile(FILE *);
int getc_unlocked(FILE *);
int getchar_unlocked(void);
int putc_unlocked(int, FILE *);
int putchar_unlocked(int);



int getw(FILE *);
int putw(int, FILE *);


__attribute__((__availability__(swift, unavailable, message="Use mkstemp(3) instead.")))

__attribute__((__deprecated__("This function is provided for compatibility reasons only.  Due to security concerns inherent in
the design of tempnam(3), it is highly recommended that you use mkstemp(3) instead.")))

char *tempnam(const char *__dir, const char *__prefix) __asm("_" "tempnam" );
# 324 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_off_t.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_off_t.h" 3 4
typedef __darwin_off_t off_t;
# 325 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4


int fseeko(FILE * __stream, off_t __offset, int __whence);
off_t ftello(FILE * __stream);




int snprintf(char * restrict __str, size_t __size, const char * restrict __format, ...) __attribute__((__format__ (__printf__,
3, 4)));
int vfscanf(FILE * restrict __stream, const char * restrict __format, va_list) __attribute__((__format__ (__scanf__, 2, 0)));
int vscanf(const char * restrict __format, va_list) __attribute__((__format__ (__scanf__, 1, 0)));
int vsnprintf(char * restrict __str, size_t __size, const char * restrict __format, va_list) __attribute__((__format__
(__printf__, 3, 0)));
int vsscanf(const char * restrict __str, const char * restrict __format, va_list) __attribute__((__format__ (__scanf__, 2,
0)));
# 349 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_ssize_t.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/sys/_types/_ssize_t.h" 3 4
typedef __darwin_ssize_t ssize_t;
# 350 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4


int dprintf(int, const char * restrict, ...) __attribute__((__format__ (__printf__, 2, 3)))
__attribute__((availability(macosx,introduced=10.7)));
int vdprintf(int, const char * restrict, va_list) __attribute__((__format__ (__printf__, 2, 0)))
__attribute__((availability(macosx,introduced=10.7)));
ssize_t getdelim(char ** restrict __linep, size_t * restrict __linecapp, int __delimiter, FILE * restrict __stream)
__attribute__((availability(macosx,introduced=10.7)));
ssize_t getline(char ** restrict __linep, size_t * restrict __linecapp, FILE * restrict __stream)
__attribute__((availability(macosx,introduced=10.7)));
FILE *fmemopen(void * restrict __buf, size_t __size, const char * restrict __mode)
__attribute__((availability(macos,introduced=10.13))) __attribute__((availability(ios,introduced=11.0)))
__attribute__((availability(tvos,introduced=11.0))) __attribute__((availability(watchos,introduced=4.0)));
FILE *open_memstream(char **__bufp, size_t *__sizep) __attribute__((availability(macos,introduced=10.13)))
__attribute__((availability(ios,introduced=11.0))) __attribute__((availability(tvos,introduced=11.0)))
__attribute__((availability(watchos,introduced=4.0)));
# 367 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
extern const int sys_nerr;
extern const char *const sys_errlist[];

int asprintf(char ** restrict, const char * restrict, ...) __attribute__((__format__ (__printf__, 2, 3)));
char *ctermid_r(char *);
char *fgetln(FILE *, size_t *);
const char *fmtcheck(const char *, const char *);
int fpurge(FILE *);
void setbuffer(FILE *, char *, int);
int setlinebuf(FILE *);
int vasprintf(char ** restrict, const char * restrict, va_list) __attribute__((__format__ (__printf__, 2, 0)));
FILE *zopen(const char *, const char *, int);
```

```
FILE *funopen(const void *,
                int (* _Nullable)(void *, char *, int),
                int (* _Nullable)(void *, const char *, int),
                fpos_t (* _Nullable)(void *, fpos_t, int),
                int (* _Nullable)(void *));
# 407 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 1 3 4
# 31 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
# 1 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_common.h" 1 3 4
# 32 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 2 3 4
# 42 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
extern int __sprintf_chk (char * restrict, int, size_t,
     const char * restrict, ...);
# 52 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/secure/_stdio.h" 3 4
extern int __snprintf_chk (char * restrict, size_t, int, size_t,
      const char * restrict, ...);









extern int __vsprintf_chk (char * restrict, int, size_t,
      const char * restrict, va_list);



extern int __vsnprintf_chk (char * restrict, size_t, int, size_t,
       const char * restrict, va_list);
# 408 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/stdio.h" 2 3 4
# 2 "hello.c" 2

void main () {
 printf("Hello world\n");
}
```

At the bottom of the file, we see the orignal code with an extra bit of informaiton that helps the compiler write better error messages, by saying where contents came from.

17.3. Compiling

Next we take our preprocessed file and compile it to get assembly code.

```
gcc -S -c hello.i
hello.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main () {
^
hello.c:3:1: note: change return type to 'int'
void main () {
^~~~
int
1 warning generated.
```

```
ls
hello.c hello.i hello.s
```

```
cat hello.s
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 10, 15, 6 sdk_version 10, 15, 6
        .globl  _main                   ## -- Begin function main
        .p2align        4, 0x90
_main:                                  ## @main
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        leaq    L_.str(%rip), %rdi
        movb    $0, %al
        callq   _printf
        popq    %rbp
        retq
        .cfi_endproc
                                        ## -- End function
        .section        __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
        .asciz  "Hello world\n"

.subsections_via_symbols
```

17.4. Assembling

```
gcc -c hello.s -o hello.o
```

```
ls
```

now we see a new file, the `.o`

```
hello.c hello.i hello.o hello.s
```

let's look at it

```
cat hello.o
```

This is not machine readable, though

```
<__compact_unwind__LD( P?__eh_frame__TEXTH@p
                                          h2

??
  PUH??H?=      ??]?Hello world
zRx
-_main_printf```
```

MacOS tried to help a little but it's still not very readable.

17.5. Linking

Now we can link it all together; in this program there are not a lot of other depdencies, but this fills in anything from libraries and outputs an executble

```
gcc -o hello hello.o -lm
```

the `-o` flag specifie the name for output and -lm tells it t link from the.o file.

again we can look at the directory

```
ls
```

we have a new executable file

```
hello   hello.c hello.i hello.o hello.s
```

Finally we can run our program

```
./hello
```

and see that it works!

```
Hello world
```

17.6. Putting it all together

We can also do all of it at once, to see how it's different let's clean up the directory:

```
rm hello.i hello.s hello.o
```

and now we can tell it to compile and link

```
gcc -Wall -g -o hello hello.c -lm
```

```
hello.c:3:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main () {
^
hello.c:3:1: note: change return type to 'int'
void main () {
^~~~
int
1 warning generated.
```

```
ls
hello           hello.c         hello.dSYM
```

17.7. Working with multiple files

```
nano main.c
```

```
nano help.c
```

```
cat main.c
```

```
/* Used to illustrate separate compilation.

Created: Joe Zachary, October 22, 1992
Modified:

*/

#include <stdio.h>

void main () {
 int n;
 printf("Please enter a small positive integer: ");
 scanf("%d", &n);
 printf("The sum of the first n integers is %d\n", sum(n));
 printf("The product of the first n integers is %d\n", product(n));
}
```

```
cat help.c
```

```
/* Used to illustrate separate compilation

Created: Joe Zachary, October 22, 1992
Modified:

*/


/* Requires that "n" be positive. Returns the sum of the
  first "n" integers. */

int sum (int n) {
 int i;
 int total = 0;
 for (i = 1; i <= n; i++)
  total += i;
 return(total);
}


/* Requires that "n" be positive. Returns the product of the
  first "n" integers. */

int product (int n) {
 int i;
 int total = 1;
 for (i = 1; i <= n; i++)
  total *= i;
 return(total);
}
```

We can first compile to object the helper functions:

```
gcc -Wall -g -c help.c
```

and then main

```
gcc -Wall -g -c main.c
```

but here we get an error:

```
main.c:10:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main () {
^
main.c:10:1: note: change return type to 'int'
void main () {
^~~~
int
main.c:14:52: error: implicit declaration of function 'sum' is invalid in C99
      [-Werror,-Wimplicit-function-declaration]
 printf("The sum of the first n integers is %d\n", sum(n));
                                                   ^
main.c:15:56: error: implicit declaration of function 'product' is invalid in C99
      [-Werror,-Wimplicit-function-declaration]
 printf("The product of the first n integers is %d\n", product(n));
                                                       ^
1 warning and 2 errors generated.
```

We can get around this, by telling main about the functions by adding

```
int sum(int n);
int product (int n);
```

to the `main.c`

```
nano main.c
cat main.c
```

to Review what we added, we can look at it

```
/* Used to illustrate separate compilation.

Created: Joe Zachary, October 22, 1992
Modified:

*/

#include <stdio.h>

int sum(int n);
int product (int n);

void main () {
 int n;
 printf("Please enter a small positive integer: ");
 scanf("%d", &n);
 printf("The sum of the first n integers is %d\n", sum(n));
 printf("The product of the first n integers is %d\n", product(n));
}
```

Now that we have added those lines, we can build again :

```
gcc -Wall -g  main.c
```

As before we get the warning, but we also get an error.

```
main.c:13:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main () {
^
main.c:13:1: note: change return type to 'int'
void main () {
^~~~
int
1 warning generated.
Undefined symbols for architecture x86_64:
  "_product", referenced from:
      _main in main-aaba11.o
  "_sum", referenced from:
      _main in main-aaba11.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Since the we got a linker error, but we didn't intend to link, we know that we ran the command incorrectly by adding the `-c` option we can get it to compile and produce the object file.

```
gcc -Wall -g -c  main.c
```

we now get only the warning as before

```
main.c:13:1: warning: return type of 'main' is not 'int' [-Wmain-return-type]
void main () {
^
main.c:13:1: note: change return type to 'int'
void main () {
^~~~
int
1 warning generated.
```

We can again see what happened

```
ls
```

Now we have the `.o` file for main

```
hello           hello.dSYM      help.o          main.o
hello.c         help.c          main.c
```

We can link everythng togther, finally by passing it both object files.

```
gcc -o demo main.o help.o -lm
```

Now we see what is there

```
ls
```

and we see tht there is a new file, the executable named demo

```
demo            hello.c         help.c          main.c
hello           hello.dSYM      help.o          main.o
```

Now we can run our program

```
./demo
```

and we see:

```
Please enter a small positive integer: 4
The sum of the first n integers is 10
The product of the first n integers is 24
```

it works!

17.8. Prepare for next class

> **Tip**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Practice using gcc. Repeat steps we did in class, change the order of parameters. Then in `gcctips.md` summarize what you learned as a list of tips and reminders on what the parameters do/why/when you would need them (or not). (to reinforce what we learned)
2. Create `operators.md` and make some notes about what you know about operators. What kinds of operators are you familiar with? Which have you seen in programming? math?
3. Add the following to your kwl chart

```
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
```

17.9. More Practice

1. (priority) Write two short programs that do the same thing in different ways and compile them both to assembly (eg using a for vs while loop to sum numbers up to a number). Check the assembly to see if they produce the same thing or if it's different.
2. Add (or link) a glossary term, cheatsheet tip, or historical context to note on the course website.

17.10. Questions After Class

17.10.1. In the .o files, how does the computer interpret the gibberish? What does it actually do?

This is machine code, it is binary that is designed for your hardware to interpret as instructions.

When we display it to the terminal, your computer tries to interpret that binary as text, so it ends up as weird characters mostly.

17.10.2. what flags are order sensitive when compiling in C?

This is a practice task to try out.

17.10.3. how does the -g flag work for all debuggers

the `-g` option leaves additional information in the object file as informtion for the debugger to use.

17.10.4. When would there be a time where you would have to look in a lower-level file (for instance, maybe something can be fixed only by going into the assembly or object code?)

The object file is specifically not human readable, you would never really look at it. The advantage of *having* it is so that you can only re-compile some parts of the code when you make small changes.

You might need to look at the assembly if you are working on very low level code, like a driver that helps a peripheral talk to the computer.

17.10.5. Why does Apple try to make the object file make sense?

Have not yet found this answer

18. Why is the object file not human readable?

The object file displays as random weird characters because it is written to disk in a different format than our terminal reads it in. It is the specific instructions (from the assembly) and memory addresses written to a file in binary. Our terminal reads it one byte (8bits) at a time and interprets those as characters.

Today, we will see how characters are represented as integers, then binary, and read and write files in binary of strings we know to see how it what happens and mimic something like how that object file happened by writing as binary and then reading it as characters.

18.1. Prelude: REPL

One way we can use many interpreted languages, including Python is to treat the interpreter like an application and then interact with it in the terminal.

This is called the Read, Execute, Print Loop or REPL.

We can execute simple expressions like mathemeatical expressions:

```
4+5
```

and it will print the result

```
9
```

We can create variables

```
name = 'sarah'
```

but assignment does not return anything, so nothing is printed.

We can see the value of a variable, by typing its name

```
name
```

and then it prints the value

```
'sarah'
```

18.2. Characters to Integers

Python provides a builtin function **ord** to find the Unicode key for that character.

> **Tip**
>
> This structure, the `[]` is called a list comprehension. It executes a `for` loop in order to build a `list`. In the REPL, it will then print out the list.

```
[ord(char) for char in name]
```

We get the integer representtion for each one.

```
[115, 97, 114, 97, 104]
```

Note that it's important to try multiple things if you aren't sure the format you are getting back or to check the official documents, as Denno found by testing this with his name in lowercase.

```
[ord(char) for char in 'denno']
```

Coincidentally, the letters in his name all fall into the hundreds and only require 0 and 1 in decimal, so it looks like they could be 3 bit binary numbers.

```
[100, 101, 110, 110, 111]
```

Python also allows us to change content into a byte array:

```
bytearray(34)
```

It does not display in the 0 and 1 like a string that we migth expect, but it does change the type.

```
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
```

We can then write this to a file, by getting the actual byes of each item. First, we will save our list in a variable.

We will use the Python `with` keyword to open the file so that we do not to need to separately open and close the file. in the `open` function, the first parameter is the file name and the second is the mode. In this case, we want to write ('w') in binary ('b'). `f` is the filestream.

```
name_ints = [ord(char) for char in name]
with open("name.txt",'wb') as f:
...     f.write(bytes(bytearray(name_ints)))
...
```

it returns 5 in this case, because it wrote out 5 bytes, one per character.

```
5
```

If we exit python we can look at the file and see how it is interpreted.

```
exit()
```

Remember we wrote, as binary, the numbers, `115, 97, 114, 97, 104` to the file

```
cat name.txt
```

but it is interpretted as the string by the terminal.

```
sarah
```

```
python
```

```
Python 3.8.6 (v3.8.6:db455296be, Sep 23 2020, 13:31:39)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

Back in Python, we can read the file as binary

```
with open("name.txt",'rb') as f:
    print(f.read())
```

This is binary, but the python interpreter also *prints* it as a string, the `b` indicates the object is actually binary type.

```
b'sarah'
```

If we read it not as binary, it returns as the string

```
with open("name.txt",'r') as f:
    print(f.read())
```

```
sarah
```

We can also confirm the type

```
with open("name.txt",'rb') as f:
    content = f.read()
```

```
type(content)
<class 'bytes'>
```

The with keyword is alternative to:

```
f = open("name.txt",'rb')
f.read()
f.close()
```

Practice exercise

```
msg = [68, 114, 46, 32, 66, 114, 111, 119, 110,10]
```

```
bytes(bytearray(msg))
b'Dr. Brown\n'
```

18.3. Bit level Operations

We can operate on integers bit by bit.

If we shift 3 one to the right we transform from `11` to `1`, we can use the `>>` operator to do this.

```
3 >> 1
```

```
1
```

We can shift it the other way too: `11` to `110` so we get from 3 to 6

```
3 << 1
```

```
6
```

We can also use logical operators

```
1 & 0
```

```
0
```

They operate bit by bit so `11` and `01` ends at `01`

```
3 & 1
1
```

The inputs to these operators must be integers.

```
type(3)
```

```
<class 'int'>
```

```
3.4 & 4.5
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for &: 'float' and 'float'
```

And Xor

```
3 ^ 3
```

```
0
```

and or

```
3 | 1
```

```
3
```

18.4. Not and Negative numbers in binary

We can also invert the bits, which flips all of bits.

```
~3
```

the output is

```
-4
```

We started with 00000011 When we flip all of the bits, we get 11111100. To see how this translates to decimal, we then have to think about how negative numbers are calulated.

There are two common ways, but the one in practice here is called two's complement. The two's complement is calculated by adding one to the one's complement, so we will calculate that first.

One's complement is to invert the bits, so in one's complement `11111100` is -3.

If we add one to that, we get the two's compliment, `11111101` so in two's compliment, that is -3.

If we calculate the two's compliment of 4:

in binary 4 is `00000100`, then we do the on'es complement, `11111011` and then add one, `11111100`, which is the what we got above by flipping the bits of 3.

note remember the first bit is the sign. 1 is for negative numbers and 0 is for positive. We used 8bit above to save visual space, but if represented as 16, 32, or 64 bit integer it would be the same.

We can also check the minimum number of bits for a number

```
(54).bit_length()
```

```
6
```

We can also get the binary representation for an integer with the builtin `bin`

```
bin(3)
```

```
'0b11'
```

This case it write the sign before the letter b, instead of using two's complement or ones complement.

```
bin(-5)
```

```
'-0b101'
```

we can also write numbers in that format and get the integer back.

```
0b11
```

```
3
```

and we can do the bitwise operations on binary numbers directly as well.

```
0b111 & 0b0111
```

```
7
```

```
0b111 & 0b0100
```

```
4
```

```
bin(0b111 & 0b0100)
```

```
'0b100'
```

```
bin(0b1111 & 0b0100)
```

```
'0b100'
```

```
bin(0b1111 ^ 0b0100)
'0b1011'
```

```
bin(0b1111 - 0b0100)
'0b1011'
```

```
exit()
```

18.5. Prepare for next class

> **Tip**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Add `bitwise.md` to your kwl and write the bitwise operations required for the following transformations:

```
4 -> 128
12493 - > -12494
127 -> 15
7 -> 56
4 -> -5
```

2. For the following figure out the bitwise operator:

```
45 (_) 37 = 37
45 (_) 37 = 45
3 (_) 5 = 7
6 (_) 8 = 0
10 (_) 5 = 15
```

3. Create `readingbytes.md` and answer the following:

```
1. if a file had the following binary contents, what would it display in the
terminal? Describe how you can figure this out manually and check it using C
or Python. '01110011011110010111001101110100011001010110110101110011'
1. What is the contents of the `sample.bn` if `cat sample.bn` is: ` ¢¶"*`
```

4. Read about integer overflow and in `overflow.md` describe what it is, use an example assuming an 8 bit system.

18.6. More Practice

1. (priority) Add to `overflow.md` how integer overflow is handled in Python, C, Javascript, and one other language of your choice.
2. Add to `readingbytes.md` and exmaple of how machine code that was a 3 bit instruction followed by an 8 bit address might render.
3. Add a box to the notes for any class that includes historical context of something covered in class or a related topic. Use the tempalte below:

```
```{admonition} Historical Context
```
```

18.7. Questions After Class

18.7.1. When is the cutoff for getting work approved for the class?

Our assigned final exam time is: May 10

So, until then you can get feedback and revise. At that time, my assessment of your work will be final, and I'll evaluate your contract. You can also change you contract up until that point.

18.7.2. Will the fact that we didn't have class the other week effect the amount of work we have to complete for our contracts?

I will announce in class how many you can reduce the required number of more practice/deeper exploration you can do.

what are some programing implementations of this arithmetic

18.7.3. Is there a low-level interface to actually make using bits as flags useful in python?

You could possibly

18.7.4. Whats the likelihood of me using bit operations in high level languages?

18.7.5. When would we use those binary operation (like in a job / research)?

We may not use them directly very often, but they are important building blocks to understanding hardware and understanding these representations, how the hardware works, and these operators, help you understand edge cases. Understanding these operations, provides the component knowledge to understand overflow and approximation issues, for example, that we will see more in the next class.

18.7.6. When using the bitwise operators (the &, |, etc.) what are the inputs?

In Python, bit level operators are defined only for integers, whether you pass them as integers or represented as binary.

18.7.7. how often (if ever) do you find yourself working in a python shell like we did today?

I use the shell directly very limited, to do quick calculations or to plan things. Mostly if I need to try to remember something or do a quick one time operation, for example, i have a file that I want to transform.

18.7.8. Questions to be answered later

- I have seen XOR gate used to solve certain programming problems efficiently, but I am curious about how to actually implement it

19. How do represent non integer numbers?

- mini coverage of notebooks
- floating point
- IEEE double format

19.1. When would you use the REPL with Python?

> ⚠ **Additional context relative to class**
>
> I added here some additional context for using the notebook that I did not mention explicitly in class.

One of the questions from Tuesday was why would we ever use python in the terminal instead of by making .py files. A Juptyer notebook aims to balance the advantages of a file and working in the interpreter.

We work in our browser, and there are cells, some are Markdown.

```
name = 'sarah'
```

```
name
```

```
cat checker.sh
```

19.2. What about a fixed point?

Let's experiment with an 8 bit representation with 1 bit for sign and then the next 4 used for the part of the number greater than 0 and the last 3 for the fractional part.

so then the number:

```
01000001
```

would be

```
0-1000-001
```

positive, 8.1

in this then we can represent the numbers 0-8 for the right hand side and 0-15 on the left hand side so we get

```
import itertools
num_list = [str(n+f) for n,f in itertools.product(range(16),[i/10 for i in range(10)])]
', '.join(num_list) + ', '.join(['-'+ n for n in num_list])
```

```
len(num_list)*2
```

This is far fewer different values than we could represent with 1 bit for sign and 7 bits to represent intergers

```
(2**7)*2
```

Another way we could represent numbers with a fixed point, is to use base two fractions: \(\frac{1}{2}\), \(\frac{1}{4}\), \(\frac{1}{8}\) etc.

In this fixed point we would have, for example: `0101.1010` would be $\(0*2^3 + 1*2^2 + 0*2^2 + 1*2^0 + 1*\frac{1}{2^{-1}} + 0*\frac{1}{2^{-2}} + 1*\frac{1}{2^{-}} + 0*\frac{1}{2^{-4}} = 4 + 1 + \frac{1}{2} + \frac{1}{8} = 5 + \frac{5}{8}= 5.625\)$

In this case we still have a small range of numbers, but it's a different set of possible numbers.

19.3. Floating Point Notation

We can write numbers in many different forms. We have written integers through many different bases so far.

For example in scientific contexts, we often write numbers (in base 10) like:
\[3.45 \times 10^2 = 345 \]
We can use a similiar form to represent numbers in binary. Using base 2 instead of base 10.

19.4. Floating point numbers are not all exact

Let's look at an example, we add `.1` together 3 times, and we get the expected result.

```
.1 + .1 + .1
```

However, if we check what it's equal to, it does not equal .3

```
.1 + .1 + .1 == .3
```

This is because the floating point representation is an *approximation* and there are multiple full numbers that are close to any given number that cannot be expressed exactly. However, the display truncates since usually we want only a few significant digits.
Even rounding does not make it work.

```
round(.1,1) + round(.1,1) + round(.1,1) == round(.3,1)
```

```
round(.1 + .1+ .1 ,1)
```

```
.1
```

```
num = .1
```

```
num
```

19.5. Floating point IEEE standard

Now, lets see how these numbers are actually represented.

IEEE Floating Point is what basically everyone uses, but it is technically a choice hardware manufacturers can techically make.

- Initially in 1985
- Reviesd in 2008 after 7 year process that expanded
- revised in 2019 after 4 year process that mostly clarified

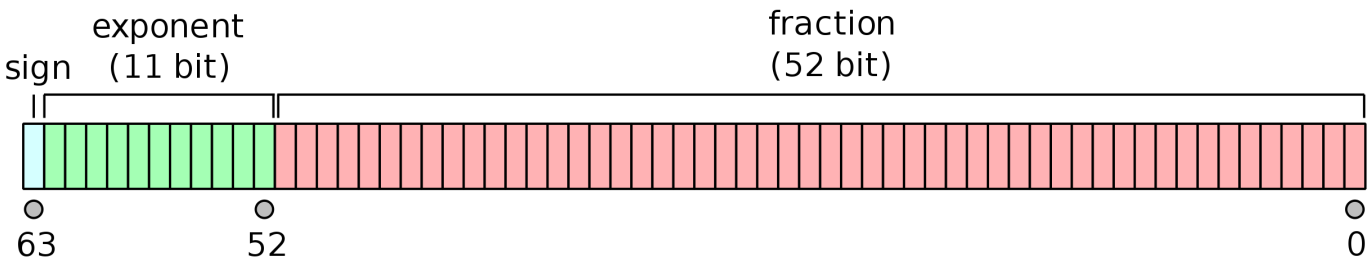
IBM mainframes use their own representiton based on Hex

Next revision is projected to 2028.

this is a double precision float or binary64 inthe current standard.

it was called double in the original, offiically, so it is commonly called that.

In this case we will 1 bit for sign, 11 for exponent and 52 for the fraction part



19.5.1. How do we read a number like this?

if the sign bit is \(s\) and the number represented by the exponent bits is \(e\) and the 52 bits are number from right most is 0 and the last one is 52.
\[(-1)s + \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023} \]
Note that this is \(2^{-1}\) so we are working with *fractions* instead of integers in the sum.

So if we had:

```
0 01111111111 0000000000000000000000000000000000000000000000000000
```

it would represent:
$$ (-1)*0 + \left(1 + 0 \cdot 2^{-0} + 0 \cdot 2^{-1} + \ldots + 0 \cdot 2^{-51) + 0 \cdot 2^{-52) \times 2^{1023-1023} = 0 + (1 + 0) \times 2^0 = 1 \times 1 = 1.0 $$
or

```
0 01111111111 0100000000000000000000000000000000000000000000000000
```

it would represent: $\((-1)*0 + \left(1 + 0\cdot 2^{-0} + 1\cdot 2^{-1} + \ldots + 0\cdot 2^{-51) + 0\cdot 2^{-52) \times 2^{1023-1023} = 0 + (1 + \frac{1}{2}) \times 2^0 = 1.5 \times 1 = 1.5 \)$

```
0b01111111111
```

```
float.hex(1.5)
```

```
0b01111111000-1023
```

19.5.2. How do we get numbers into this form?

Now, let's return to our example of .1.

First we take the sign of the number for the sign bit, but then to get the exponent and fraction, we have more work to do.

Let's take the equation we had from the standard:
$$ \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}\]
If we think about the fraction and how we add fractions, by reaching a common denominator. Since they're all powers of two, we can use the last one.
$$ \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}\]
$$ \left(1 + \frac{b_{52}}{2^{1}} + \frac{b_{51}}{2^{2}} + \cdots + \frac{b_{1}}{2^{51}} + \frac{b_{0}}{2^{52}} \right) \times 2^{e-1023} \]
Now with a common denominator:
$$ \left(\frac{2^{52}}{2^{52}} + \frac{2^{51} b_{52}}{2^{52}} + \frac{2^{50} b_{51}}{2^{52}} + \cdots + \frac{2^{1} b_{1}}{2^{52}} + \frac{2^0 b_{0}}{2^{52}} \right) \times 2^{e-{1023}} \]
So then this becomes a binary number with 53 bits (the 52 + 1) and a denominator of 2^{52}, let's call that number J.
$$ \frac{J}{2^{52}} \times 2^{e-{1023}} \]
we can then combine the powers of 2.
$$ \frac{J}{2^{52-e+1023}} \]
So i order to return to our `.1` that we want to represent, we can represent it as a fraction and then estimate it in the form above.
$$\frac{1}{10} ~= \frac{ J }{2^N}\]
$$J ~= 2^N / 10\]
Since we want to use exactly 53 bits to represent J, we want $\frac{2^N }{10}$ to be greater than or equalt to 2^{52} and less than 2^{53}.
$$ 2^{52} <= \frac{2^N }{10} < 2^{53} \]
Since $10 =8+2 = 2^3 +2^1$ then $2^3<10<2^4$ We can say that $\(\frac{2^N }{2^4} < \frac{2^N }{10} < \frac{2^N }{2^3} \)$
$$ 2^{N-4} < \frac{2^N }{10} < 2^{N-3} \]
so if we want:
$$ 2^{52} <= \frac{2^N }{10} < 2^{53} \]
then best N is 56, but we can check it.

```
2**52 <= 2**56 //10 < 2**53
```

Now we can get the number we will use for J. We want to get as close to .1 with our fraction as possible, so

```
q,r = divmod(2**56,10)
```

Then we check the remainder to decide if we should round up by 1 or not.

```
r
```

$\(6 > 5 = \frac{10}{2}\)$ so we round up

```
q+1
```

then we chan check the length in bits of that number

```
len(bin(q+1))-2
```

```
bin(q+1)
```

We need \(52-e+1023 = 56\) so

```
52-56+1023
```

Python doesn't provide a binary reprsentation of floats, but it does provide a hex representation.

```
float.hex(.1)
```

If we tak the binary above, it matched this for the part before the p. `0b1 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1010`

after the p is the exponent of \(-4 = 1019-1023\). Which matches the approximation we found.

```
(q+1)/2**56 == .1
```

this confirms that the approximaiton we found is the same as the float representation of .1.

```
format(.1, '.17f')
```

We can also use built in classes to get at the needed quantitites

```
from decimal import Decimal
from fractions import Fraction
```

```
Fraction.from_float(.1)
```

```
bin(3602879701896397)
```

```
bin(55)
```

```
float.hex(.1)
```

```
q/2
```

```
2**55
```

```
Decimal.from_float(.1)
```

```
Decimal.from_float(.1 + .1 + .1)
```

```
Decimal.from_float(.3)
```

```
Decimal.from_float(.4)
```

```
Decimal.from_float(.1 + .1 + .1 + .1)
```

19.6. Prepare for next class

> **Tip**
>
> The text in () in these sections is an explanation of *why* that task is assigned

> **Tip**
>
> If you have jupytext installed, you can make the file, `floatexpt.md` a runnable notebook when you open it through the `jupyter notebook` interface and a plain text file that is easily readable on github, by pairing or by manually creating it as a jupytext markdown notebook in any text editor (including the IDE of your choice) and then the code will execute to show the outputs when you build your KWL repo as a jupyterbook. Jupyter book will be able to run it even if you don't separately install jupytext, but you need the jupytext to generate it automatically.

1. Write a C program to compare values as doubles and as float (single precision/32bit) to see that this comparison issue is related to the IEEE standard and is not language specific. Make notes and comparison around its behavior and include it in a code cell in `cdouble.md`(to practice)
2. (priority) confirm that you can run the hardware simulator `HardwareSimulator` not `CPUEmulator` that we used before (we will use this in class)

19.7. More Practice

1. In `floatexpt.md` design an experiment using the `fractions.Fraction` class in python that shows helps illustrate how `.1*3 == .3` evaluates to `False` but `.1*4 ==.4` evaluates to `True`. (practice/review)

20. How can we use logical operations?

20.1. Admin

20.1.1. Grading Contract Updates

- You can reduce your more practice days completed by 2.
- You can reduce the number of deeper exploration by 1.

> **Important**
>
> Review your grading contract and either submit that PR, or an issue noting if you need another update.

20.1.2. Deadlines

- Things submitted by May 5 you can revise again.
- I will hold office hours for your last chance to do final touches during our exam time May 10, 11:30-1:30.
- I will do final grading starting May 11 in the afternoon; all work must be submitted by noon May 11 to guarantee grading.
- I will post additional office hours and feedback hours during finals week soon.

20.2. Review

- git revert applies a new commit that does the opposite of the reverted commit. review
- A file with permissions 644 is rw-r–r–, so only the owner, not other members of the group, can write to the file. review

21. Why do we need to think about bitwise operations?

Understanding them is prereq to what we will see today and that will help you understand hardware overall.

You of course will not *need* every single thing we teach you in every single class.

- Seeing topics once at least is the only way you can make an informed decision to study a topic deeper or not.
- Seeing a topic in more detail than you will use all the time actually helps you build intuition, or deep understanding, of the topic overall, and help you remember what you need to remember

21.1. Bitwise operators

- & : and
- | : or
- ^ : xor
- ~ : not
-
  > : shift right
- <<: shift left

Let's review truth tables for and, or, and xor.

a	b	output
0	0	0
0	1	0
1	0	0
1	1	1

Table 21.1 AND

a	b	output
0	0	0
0	1	1
1	0	1
1	1	1

Table 21.2 OR

a	b	output
0	0	0
0	1	1
1	0	1
1	1	0

Table 21.3 XOR

In order to implement more complex calculations, using gates, we can use these tables as building blocks compared to the required output.

There are more gate operations; you can see a simulation for 16 gates

21.2. Adding with gates

Let's review adding binary numbers

- add the two bits
- carry the next place like in adding multi-digit numbers otherwise

\[101 + 100 = 1001 \]
We first add the ones place and get a 1, then the two's place and get a zero then the 4's place and get 0 with a carried one.
\[010 + 011 = 101 \]
In this case in the ones place we add 0 + 1 to get one, the two ones add to 0 with carry then 1 + 0 + 0 gives another 1.

let's make a truth table for adding two bits.

a	b	out 2's	out 1's
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 21.4 Add

Now, what gate can we use to get the output 1's place bit and what gate can we use to get the output 2's place bit by comparing to the truth tables above.

It turns out the one's place is an xor gate, and the two's place is an and gate.

This makes up the half adder, try one out at this simulator.

So this lets us as two bits, but what about adding a number with more bits?

We can put multiple together, but there's one more wrinkle: the carry.

That's what makes a full adder different. It adds three single bits, or a carry and two bits and outputs the result as a sum bit and a carry bit.

Then we can link many of those together to get an 8 bit ripple adder.

Alternatively, we can "lookahead" with the carry bit, passing it forward multiple places all at once, as shown in this 4 bit carry lookahead adder.

21.3. How can we choose between things?

A mux (multiplexer) is a component (like a gate is a component) that allows us to select one operation out of a circuit.

21.4. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Compare the 2 bit multiplier to the full adder in `multiplication.md`. Use that to explain how it works, relative to the fact that multiplication can be thought of like repeated addition.
2. Study the 8 bit ALU. Try it out and be prepared to answer questions about how it works on Thursday.
3. Read gates out of anything

21.5. More Practice

1. In `addertypes.md` compare ripple adders and lookahead adders.

```
1. Give a synopsis of each adder type
1. Compare them in terms of time (assume that each gate requires one unite of
time)
1. Compare them in terms of space/cost by counting the total number of gates
required.
```

2. (priority) While we saw many types of gates today, but we actually could get all of the operations needed using only NAND gates. Work out how to use NAND gates to implement a half adder.

21.6. Questions After Class

21.6.1. How do we know to use the selector in a mux?

This comes from the instruction itself.

21.6.2. What other mediums besides water could these be implemented with?

We will talk more about different ways to physically create gates in the next few classes

22. What *is* a computer?

- how, physically, do we get the components we have seen?
- what other components do we need?
- how are those implemented

How have computers changed over time?

- at the physical level
- what was the context and motivation for these advances?
- how does that context influence how we use computers today?
- how does that influence computing as a discipline?

22.1. Let's start with a dictionary

we can start with a dictionary

- note that this starts with reference to a person.

22.1.1. Computers as people

and this text this particular dictionary is useful because it also includes where does the term originated like where does in that particular usage start appearing and so these are these are pretty all right this is starting 1646 1707

- Use of the word computer to refer to the person still common how recently 1950s
- the movie hidden figures is available on Disney+ with subscription or other streaming platforms for ~$4 streaming rental.
- the book hidden figures is available at URI's library

22.1.2. Computers as Machines

computer starts to be termed talking about a machine must later so that 1897 is a long time after 1646

you like ideas

most of them are physical really things like you can see there's a couple things early but most of it starts around 1944 46 around then what was going on

they weren't ya so early computer people were mathematicians by like training and their own identity but then they kind of blurred into things too the world was very dark place and it is today they're worried about

other like it was a lot of applications it was also like the Manhattan Project was one of the things that motivated them like this is why we need to build a lot of calculations and then later at the end of the 56 the Moon that was also a lot of math like we need to make sure these Rockets can like go up and come back down the way we expect them to

22.1.3. How much do we talk about computers?

how much do we talk about them changes over time

how much do we have to talk about them so Google does this thing with Google Books

it's me what's surprising about that surprising

if you talk to you not the

is there talk about things when they're new

- note that we talk about things a lot using their basic name when they're new
- we don't talk about normal things

hey I think things are trained on news Corporal and do not like otech bodies of news text from the internet

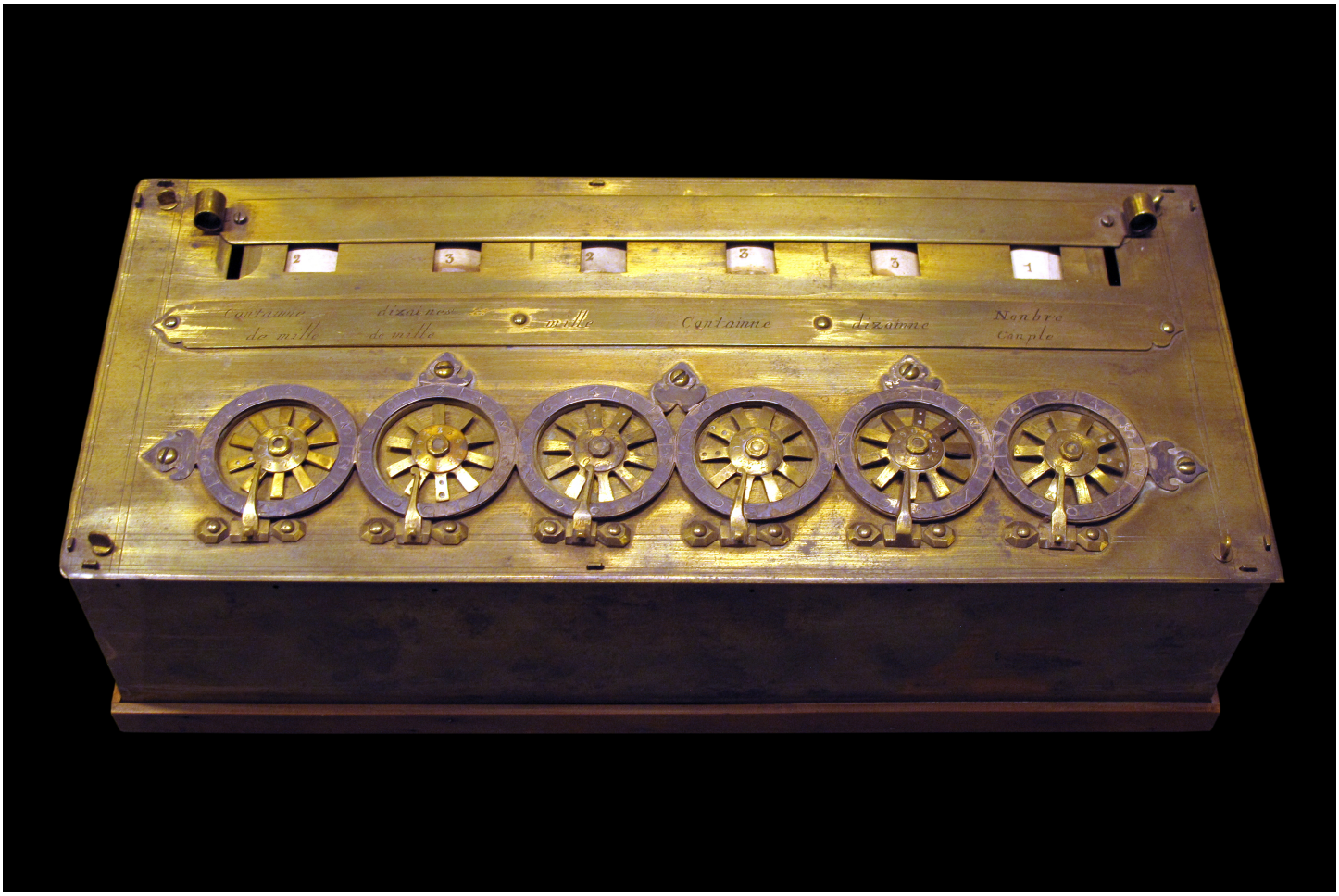
they would consider murder a very high frequency and blinking very low frequency of that

don't talk about blinking we do it all the time like I'm sure since I started talking with my king every one of us has blank more than once

22.2. Mechanical Calculators

how do we actually like physically make it be machine

well we've we've talked we handed out that like it's going to reduce down to or isn't my check and logical operations write those de back much further than

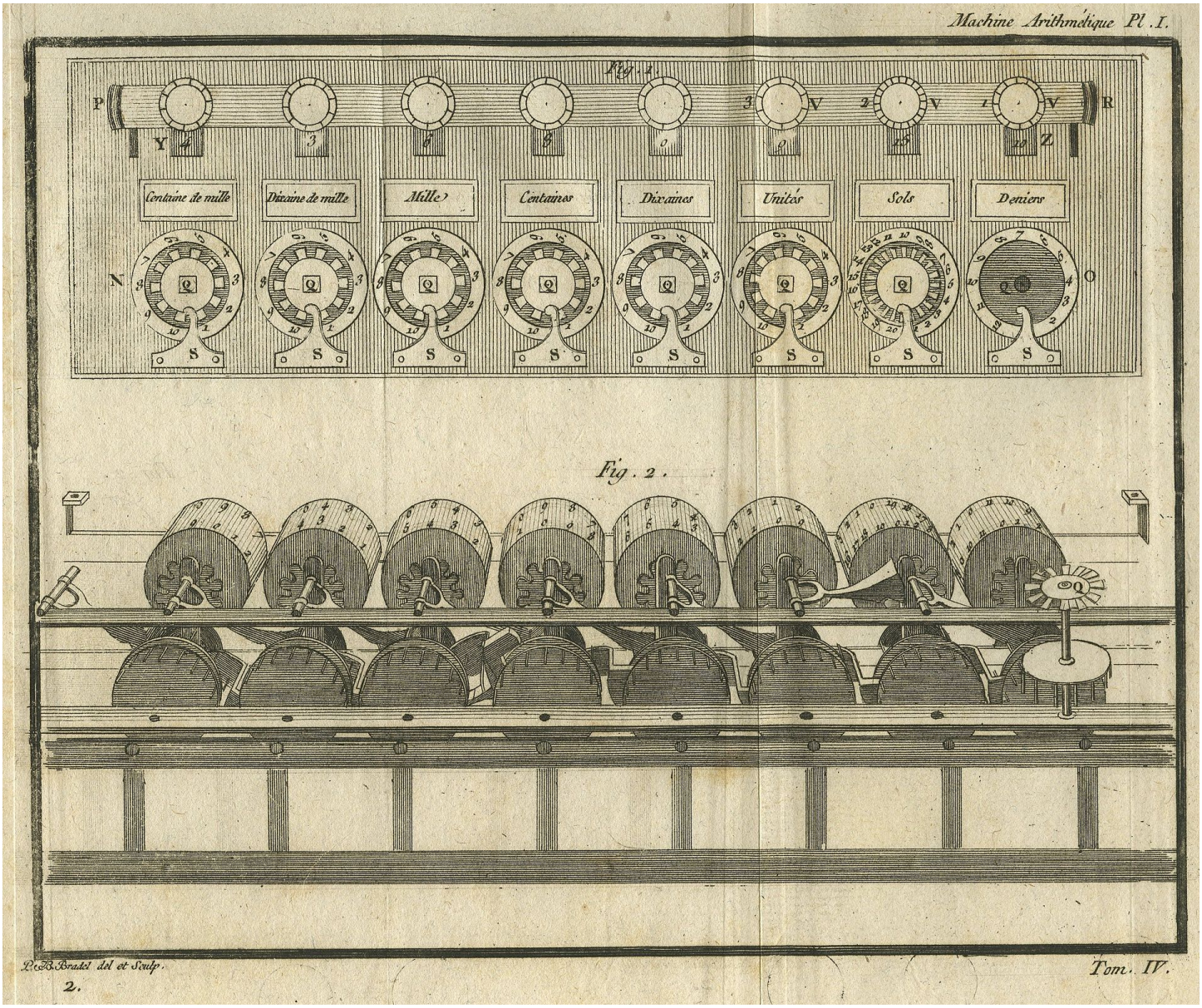


what is the mechanical calculator it works by like incrementing it doesn't do the logic like they took not a binary representation it's not adding via input two numbers and then it goes forward its where's my income as you put in one number then you can add the next number and it kind of works the wheels have like the numbers on them and it uses like 9 supplement to carry over to the next bit so there's a gear that when it hits a certain point that turns the next year and then it goes for it any Castles in the Sky why the semi did

Why did he make this?

He was 18 and his father was a tax commissioner in France and he wanted to like reduce his father's workload

so he invented this calulator machine and received roayl privilege from France in 1649 to be only person allowed to build a manufactured Computing a calculator is like mechanical calculators



Note

Contrast this royal privelege with the 1970s where in resolving some other conflict, a court that declared the content of the computer that they cannot be found it at all and it's free for anyone to use the concept of a computer

22.3. How did this change over time?

calculators that are still analog so they're still working with mechanical systems using Waze electricity becomes easier

we can start doing like electrical signal processing so we can move analog signals and start doing our agents and calculations with those are not what was really used through for transmitting information like telegram telephone all of that is analog thing

between Springs and dampers, the mechanical things are used to build these mechanical calculators and the electrical components so resistors transistors.

timeline of computing

22.3.1. Model K Adder (1937)

- George Stibitz took some pieces at home from work at Bell Laboratory and got it working on his kitchen table.
- it can do four bit addition

- uses a telephone relay switch or an electromechanical switch so they're using like a

> **Note**
>
> Relays are designed to repeat, or relay, a signal to the next circuit to compelte long distance telegraphs and early telphone. As the signal transpits down the cable (one circuit) it gets weaker due to loss, so it has to be repeated at some point to amplify the magnitude of it for the next length of the journey.

precursor to the complex number calculator

he demonstrated how this works in combination with Telegraph

for the first remote access

- complex number calulator was at Bell Labs in New York City at the time in 1940 at a Confrence at Dartmouth in Hanover New Hampshire and he use a telegraph to do remote calculations on this calculator and bring them the numbers back

22.3.2. So how do these thigns actually work

- We can build an adder mechanically with marbles and wood

As electricity became more available, they realized that in terms of mathematical operations, electrical components resemble springs and switches

- first vaccuum tubes: or diodes
- diodes prevent flow of electricity in one direction and allow in the other. they can be used to create circuits that behave like the logical operations, and create the gates.
- diodes only are easy for `&` and `|` gates,
- transistors can operate as a switch or amplifier and can also be used to build logic gates
- they're faster, smaller, and better at more types of gates than diodes

22.4. Early Computers

During this time small, lots of small computers were made, but each was made completely by its creators, there was no standards, per say.

These computers were also stored program computers. Meaning their operation at a program level was fixed by the circuitry.

switch the circuit to do different things between the numbers so this would get us bitwise operations of an indoor and outdoor on our numbers

we had the first program later this eniac computer

which athlete came out certainty 43 wasn't finished for a little while this this all this is one computer and see there's several people in this room doing this one computer this was programmed manually like changing the circuits because it helped me had and that point in time they only had read-only memory you could put something into memory we don't have memory like what kind of memory that you could keep when you said the power off for a long time what's the powers off then what was that whatever was in Army was on and all we had was just reading the numbers you could program the computer stuff is working out to do the thing we're going to do and then you could go do that calculation many times you needed an Ordy bug a program that it took weeks to value of all experience the pain of debugging rarely it takes two weeks to debug one error correct

right yes maybe it hurts sometimes it's like a long night not enough sleep but like this week's of like moving cables did anyone get his memory we need some way about you storing things and writing things down and we need a way that actually can store memory that first they do more vacuum tubes do the same sort of switching on and off with store stuff but the problem with that is the power off

so but this

stored program computers they only had read only memory and eventually we got random access memory which we could write and read and change whenever we wanted

which we can have including storage without power

so and not but we have three types of places we store information we got registers which store information that is currently being processed so we look back at our circuits these inputs here we're turning onto just as B were treating them as switches but these are actually implemented as register if you have a register at the input and a register register the input for each of the inputs to b l u r i a registered but it's only live it's got some number of bits in it so see if you're paying attention and putting things together we talked about a computer system having like operating doing some type of some type of a b s what device stores that many bits

Dell computers we have

so what what is Ashley's what is the thing that has that number of bits is a limitation of the device equipment stores at me I guess I should have changed the options but we'll go look what it is

22.5. von Neumann Architecture

what else do we need besides ALU for to have like a whole computer

we eventually need clocks to time things and synchronize things to like know when to clear and when to change so that you if you want to do multiple things in sequence you can do that what else you need or want to do multiple things in sequence but don't know what else we need a general now

store more insturctions and data

my diagram didn't work so the basic architecture we used to talk about all the main component we need is called the Von Neumann architecture this is a slightly expanded version of it could be slightly simpler than this but

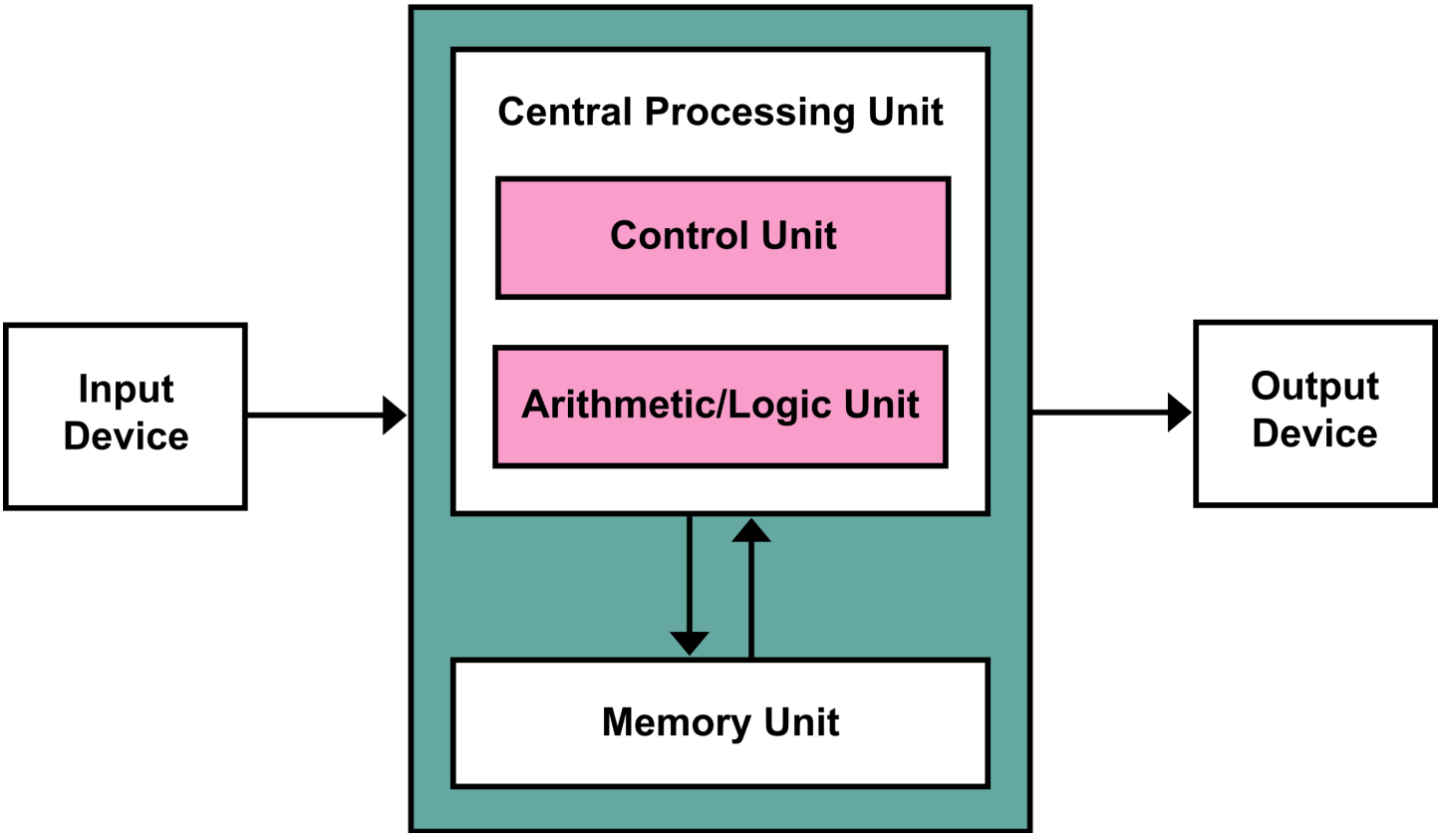
theater image put the volume in architecture

was from 1945 was written by people at 10 and this is basically how we still design computers for the most part we have you control unit

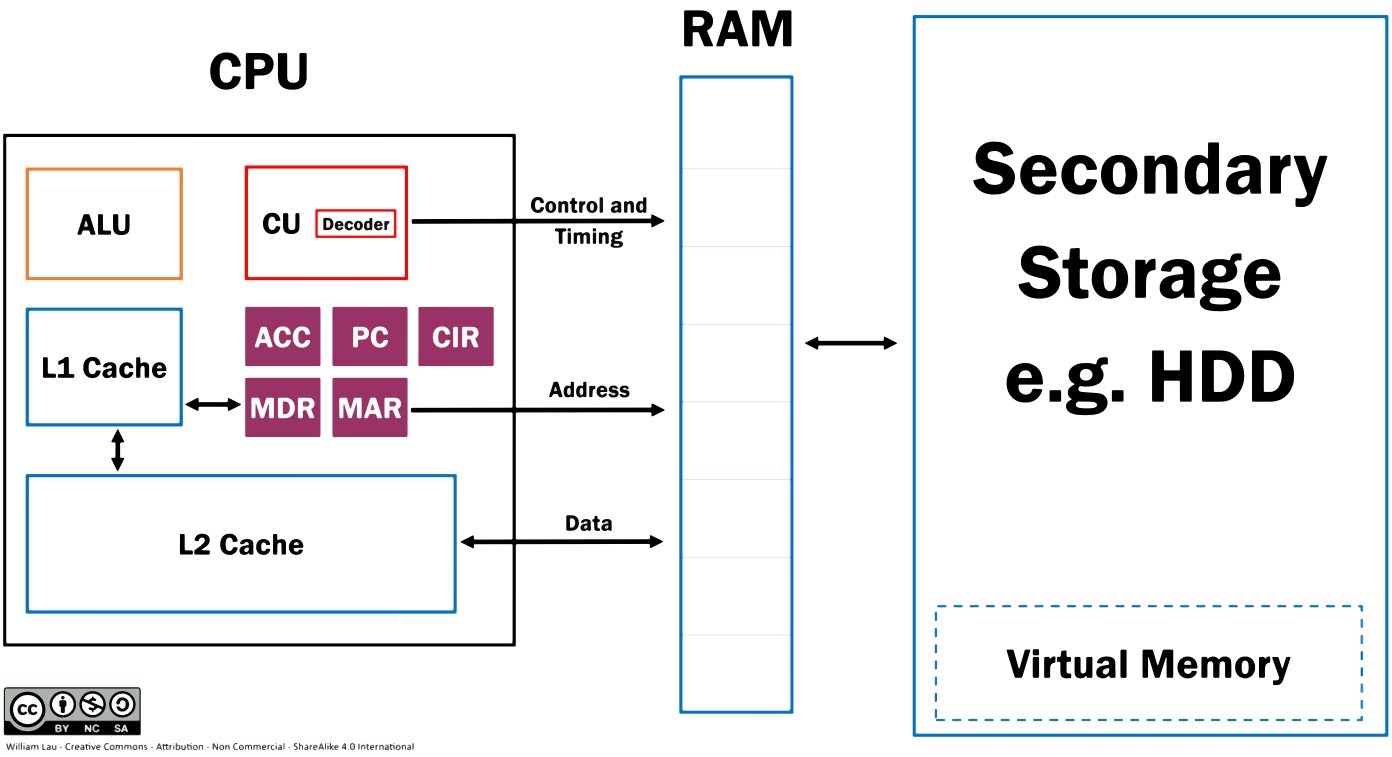
some memory and some registers together all of these things or the CPU

these can talk to Ram which is memory random-access memory so this is a memory that can be changed any time we want it Amanda storage or hard drive which is now not as physically different from these two but that's there

In 1945 we get a draft of a general template. This is basically what we use today.



Computer Systems - Von Neumann Architecture



22.6. Storage

- there are some thigns we have not yet seen in detail in that diagram
- Ram Is Random Access we can access it whenever we want we can read and write to
- ROM: is read only memory; the instructions are Permanently

In the stored program computers, the programs were in ROM.

The very earliest computers could not store any values without power.

- Register: data currently processing
- Memory: will be required for processing
- Disk: long term storage

22.7. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Read about systems abstractions in CACM. Answer reflection questions in `systemsabstractions.md`:

```
1. How many of these are familiar/not?
1. How has you undertanding of these changed during this course.
1. Do you think you understand this article more now than you would have at
the beginning of the semester
1. Write 3 questions and their answers that could be a quiz to see if someone
understood or had misconceptions about the abstractions in this article.
```

2. Review the notes and add one more historical event's information to the notes using either the computer history museum and/or additonal sources. Check the open PRs to ensure that no one else has submitted the same as you.
3. Update your KWL chart based on the minimum rows list

22.8. More Practice

1. Pick one historical event about computers, memory and storage or networking and the web (not limited to what is on those pages) and in, 'history-' describe the event, how it impacted what has come since.
2. Map out what you know about computer hardware in some form of visual or outline in `hardwaremap.md` and bring three questions to class.

22.9. Questions After Class

22.9.1. Do we eventually switch our computers to use more than 64 bits?

We really get just about all of the precision we need out of 64 bits and creating registers, ALUs, buses, etc with more than that is more expensive and slower, so mostly there seems to be no need, for general purpose computing.

Switching from 32 to 64 was important not only for precise calculations because we can use memory tricks to circumvent that anyway, but for the ALU to participate in addressing memory. With 32 bit registers we can only express 2^32 addresses for the RAM, so we can only access 4Gibibytes or 3.8GB RAM (we'll cover this translations later), with 64 bit registers we can address 2^64= 2^32*2^32 so we squared the amount of memory we can address this is 16 Exibbytes or a little bit fewer Exabytes, which is far more than we need for regular computing, while more than 4GB of RAM is quite desirable for everyday use.

So, it's really that the extra cost provides no realizable benefits, more than the absolute cost.

22.9.2. Are certain gates more commonly used in CPU design?

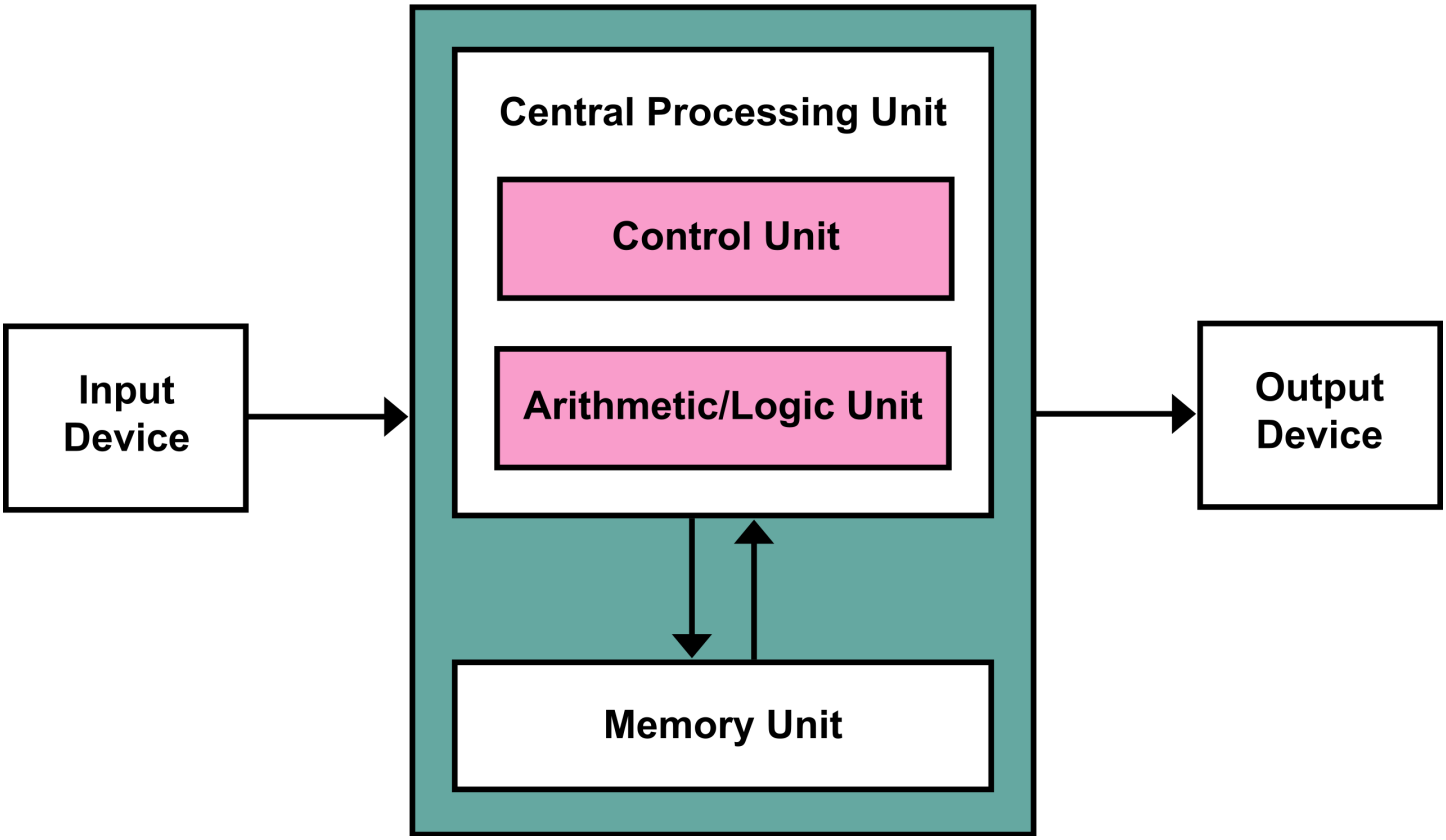
See the exercise from the last class, but yes!

22.9.3. How like literally how is even remotely possible to assemble modern day cpus and the size and scale they are

Computers got smaller by better physics that allows for smaller implementations of the gates. This is a good topic for a deeper exploration.

23. How do clocks impact computing?

23.1. Recall our model Computer



We have talked about the ALU at length and we have touched on memory, but next we will start to focus on the Control unit.

We discussed that the operations we need to carry out is mostly

23.2. Control Unit

The control unit converts binary instructions to signals and timing to direct the other components.

23.2.1. What signals?

We will go to the ALU again since the control unit serves it to figure out what it needs.
Remember in the ALU, has input signals that determine which calculation it will execute based on the input.

23.2.2. Why Timing signals?

Again, the ALU itself tells us why we need this, we saw that different calculations the ALU does take different amount of times to propagate through the circuit.

Even adding numbers of different numbers that require different number of carries can take different amount of times.

So the control unit waits an amount of time, that's longer than the slowest calculation before sending the next instruction. It also has to wait that long before reading the output of the ALU and sending that result to memory as needed.

23.3. What is a clock?

In a computer the clock refers to a clock signal, historically this was called a logic beat. This is represented by a sinusoidal (sine wave) or square (on, off, on, off) signal that operates with a constant frequency.

This has changed a lot over time.

The first mechanical analog computer, the Z1 operated at 1 Hz, or one cycle per second; its most direct successor moved up to 5-10Hz; later there were computers at 100kHz or 100,000Hz, but where one instruction took 20 cycles, so it had an effective rate at 5kHz.

> **Try it Yourself**
>
> Look up the CPU speed of your computer and your phone. How do they compare.

23.4. Execution Times

```
time kwlfilecheck
```

We get three times:

- real: wall clock time, the total time that you wait for the the process to execute
- user: CPU time in user mode within the the process, the time the CPU spends executing the process itself
- sys: CPU time spent in the kernel within the process, the time CPU spends doing operating system interactions associated with the process

The real time includes the user time, the system time, and any scheduling or waiting time that that occurs.

23.5. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Check that your KWL is up to date with all required rows and add a section (`##`) below your chart in the same file (either `README` or `chart.md`) that summarizes what you think about computer systems.
2. (priorty) Read about systems vs applications programming. Create `systemsprogramming.md` to summarize the difference and what skills would be diffrent. Which do you have more experience with? What experience do you have with the other? (we will address this in class Thursday)

23.6. More Practice

1. (priority) Compare the time of the following operations, investigate what the different operations do/not do, how do the three times compare? and hypothesize explanations in `timing.md`. Which times are constant vs variable? Which vary the most? least? Tip: use a script with loops and see the optionsand write output to a file (to test how these impact your work. *extension idea:* expand the level of detail to include more analysis on this, Try to find operations that increase the user vs system time. You can use basic bash operations or programs you have from other courses)

```
1. wget https://github.com/introcompsys/spring2022/blob/main/img/2022-01-27-
mental_model_advanced.svg
2. wget
https://raw.githubusercontent.com/introcompsys/spring2022/main/img/2022-01-27-
mental_model_advanced.svg
3. kwlcheck
4. kwlfilecheck
5. hello world compiled from C
```

2. Practice exploring the list of processes running on your computer and investigate how much resources different things you do on a regular basis use. Make some general observations In `kernelproc.md`.

23.7. Questions After Class

23.7.1. where are the registers?

The registers are a part of the CPU.

23.7.2. Is there a model of CPU without an ALU?

The term CPU specifically refers to combining an ALU with a control unit and registers. Modern CPUs use integrated circuits, so multiple components are etched into silicone instead of having separate components that are, for example, soldered onto a board.

23.7.3. What is a core

A core is an execution unit: a control unit, an ALU, and the necessary registers. Traditionally a CPU had one core, so the model CPU we saw is the same as a core. A modern, multi-core CPU has multiple execution units and additional circuitry to coordinate.

23.7.4. Will we get time in class this week / next week to use as like a workshop to finish things we need to with help from you?

This is what office hours are for. I added more office hours, the link to the times are in slack and will be shared in class.

Please, please! come to office hours or ask for an additional appointment.

23.7.5. Why do the different times matter?

the different times demonstrate different parts of what has to happen, there is scheduling and waiting; there is the actual executing and there are kernel (OS) steps.

23.7.6. What are the side effects of overclocking the computer?

It gets hotter, so that it causes more stress and strain on the

23.7.7. Is there a good intro to this stuff online? Or will we go over it in 411?

More detail on this will be in 411 and the OS/ process side will be in 412.

23.7.8. What are the pieces of a kernel and what does it actually do for the computer?

The kernel is software, not a physical component.

24. Systems Programming and threading

24.1. Systems vs Application Programming

- most of you will do a lot more application programming than systems programming
- knowing how the systems stuff works is important because you will need to interact with it.

24.2. System interaction in Python

We'll use the python interpreter a bit again today.

```
python
```

```
Python 3.8.6 (v3.8.6:db455296be, Sep 23 2020, 13:31:39)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

In order to interact with the system in a Python program, we use the python os library.

```
import os
```

It has methods that look a lot like a lot of the bash commands we have used.

For example `listdir` is like `ls`

```
os.listdir()
```

```
['test', 'tiny-book', 'checker.sh', 'compilec', 'README', '2022-04-07.md', 'nand2tetris', '.ipynb_checkpoints', 'courseutils',
'test2', '2022-04-07.ipynb', 'github-in-class-brownsarahm']
```

We could then use this as a python list object.

```
file_list = os.listdir()
file_list
```

```
['test', 'tiny-book', 'checker.sh', 'compilec', 'README', '2022-04-07.md', 'nand2tetris', '.ipynb_checkpoints', 'courseutils',
'test2', '2022-04-07.ipynb', 'github-in-class-brownsarahm']
```

This library can also help us interact with the system in a more abstract way, for example it can hide what varies per operating system.

For example the following works on Linux and MacOS, but would not work on Windows.

```
with open('tiny-book/intro.md','r') as f:
    f.read()
```

The `path` module within the library will join parts of a path together with the right delimiter. The following will output differently depending on what operating system you run it on

```
os.path.join('tiny-book','intro.md')
```

On Unix-based systems

```
'tiny-book/intro.md'
```

on Windows it would be:

```
'tiny-book\\intro.md'
```

Knowing that tools like this exist allows you to interface with the system and write different types of programs.

This is an important type of library to know exists so that you can look up the specific functions and modules you would need in any given scenario.

and we can exit Python now.

```
exit()
```

24.3. Threading

We are going to pretend that summing squares of numbers is expensive computation and we want to spread it across different multiple cores. To do that, we spread it into separate threads.

```
nano sq_sum_threaded.c
```

The program will be:

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

/* single global variable */
/* shared, accessible, modifiable by all threads */
int accum = 0;

void* square(void* x) {
    int xi = (int)x;
    accum += xi * xi;
    return NULL; /* nothing to return, prevent warning */
}

int main(int argc, char** argv) {
    int i;
    pthread_t ths[20];
    for (i = 0; i < 20; i++) {
        pthread_create(&ths[i], NULL, square, (void*)(i + 1));
    }

    for (i = 0; i < 20; i++) {
        void* res;
        pthread_join(ths[i], &res);
    }

    printf("accum = %d\n", accum);
    return 0;
}
```

> **Note**
>
> this activity is from Matthew Wachs lecture notes

Then we can build the program.

```
gcc -pthread -Wall -g -o sqsum sq_sum_threaded.c -lm
```

We can both compile and link it at once and we get just a warning

```
sq_sum_threaded.c:19:43: warning: cast to 'void *' from smaller integer type
      'int' [-Wint-to-void-pointer-cast]
    pthread_create(&ths[i], NULL, square, (void*)(i + 1));
                                          ^
1 warning generated.
```

and we can run the program

```
./sqsum
```

I got the wrong answer

```
accum = 2833
```

We saw that different students got different answers.

We can use a for loop in bash to explore this further and figure out why.

```
for i in {1..1000}; do ./sqsum; done | sort | uniq -c
```

this also uses some new bash commands:

- `sort` orders all of the outputs of the 1000 runs of our program

- and `uniq` with the `-c` option counts how many times any given result appeart multiple times

```
   1 accum = 2701
   1 accum = 2726
   1 accum = 2749
   1 accum = 2770
   1 accum = 2814
   1 accum = 2820
   1 accum = 2821
   1 accum = 2833
   4 accum = 2834
   1 accum = 2841
   8 accum = 2845
   8 accum = 2854
  10 accum = 2861
   4 accum = 2866
   2 accum = 2869
 955 accum = 2870
```

So, this time I got the right answer most of the times 955 out of 1000, but lots of other answers at least once.

To understand what happens, lets look at the following program, which should be an equivalent way to implement the body of the `square` function.

```
int temp = accum;
temp += x * x;
accum = temp;
```

In this one, we first copy the `accum` value to a temporary variable, then square the value and add that to `temp`, and then finally add that value back to `accum`. This should be equivalent to the program above, result wise.

Even though this is not how we wrote our program, this is actually what it has to do, as we spin out each process.

This table traces through what occurs in two threads.

// Thread 1	// Thread 2	Status
`int temp1 = accum;`	`int temp2 = accum;`	now `temp1 = temp2 = 0`
	`temp2 += 2 * 2;`	now temp2 = 4
temp1 += 1 * 1;		// temp1 = 1
accum = temp1;		// accum = 1
accum = temp2;		// accum = 4

So, what happens is each thread looks at the current value of `accum` and stores it to a thread-specific temporary variable. Each thread has its own memory, but they do all share the global variables.

Then thread 2 completes its calculation and updates `temp2` and thread 1 updates `temp1`. So far everything is okay, but next thread 1 writes to `accum` and sets it to 1, and finally thread to writes to `accum` and makes it 4. The two values from the threads did not get added together, because thread 2 started before thread 1 finished.

So, we end up losing some of the values.

24.4. Locking

We can instead change our `square` function as follows:

```
int accum = 0;
pthread_mutex_t accum_mutex = PTHREAD_MUTEX_INITIALIZER;

void* square(void* x) {
    int xi = (int)x;
    int temp = xi * xi;

    pthread_mutex_lock(&accum_mutex);
    accum += temp;
    pthread_mutex_unlock(&accum_mutex);

    return NULL; /* nothing to return, prevent warning */
}
```

and save this to a new file `sq_sum_threaded_m.c`

This version uses something from the pthread library, to create a lock.

Now when it executes each thread will do the calculation part on it's own time, possibly simultaneously. Then the lock part means that they will each take turns to add their value to the global variable `accum`.

We can the build it

```
gcc -pthread -Wall -g -o sqsum_m sq_sum_threaded_m.c -lm
```

```
sq_sum_threaded_m.c:26:43: warning: cast to 'void *' from smaller integer type 'int'
      [-Wint-to-void-pointer-cast]
    pthread_create(&ths[i], NULL, square, (void*)(i + 1));
                                          ^
1 warning generated.
```

we get the same warning

Now we can run it again, using our loop

```
for i in {1..1000}; do ./sqsum_m; done | sort | uniq -c
```

and now we get the same result all 1000 times.

```
1000 accum = 2870
```

24.5. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. (**Priority**) Work with a partner to add "advice to future students" testimonials to website for this offering and "why take this class" testimonials to the main course site using an integration manager like workflow. Coordinate to determine who's fork will be used for integration of each . Each of you should clone that fork and add your contribution to **`resources/testimonials.md`** and **`fromstudents.md`** on a branch, `<initials>_testimonial` eg Professor Brown's would be `smb_testimnial`. Then merge the two contributions to a single `testimonial` branch. (both to help future students *and* to practice with branches and collaboration, we will work with these further in class next week)
2. Review the notes.
3. Update your KWL chart. Check that all rows are included.
4. Simulate a more computationally intensive program using the `sleep` function in C and compare the time of a threaded vs single threaded version of the program. Include the two programs, output of your test, explanation text, and conclusions in `singlevmultithread.md` (to better illustrate the impact of the threads)

24.6. More Practice

1. Learn about the system libraries in two languages (one can be C or Python, one must be something else). Find the name(s) of the library or libraries. In `systeminteraction.md` make notes for each language about what categories of functions they provide.
2. (**priority**) Research examples of programs using multi-threading besides splitting up a single calculation for time reasons, include three examples in `whymultithread.md`.

24.7. Questions After Class

24.7.1. Are there certain languages or systems that lock for you?

Different languages handle threading different ways. I do not know all of them. I would expect that there are some libraries and frameworks that help handle it for you.

The important reason to use a lower level language like C to explore this, where things are not handled for you, to learn the concepts and see all of the steps that have to occur. This helps you when it is supposed to be handled, but something goes wrong.

24.7.2. Could you recommend any extra readings about this?

You will learn more about threading in CSC412 and you can optionally take CSC415 parallel programming.

Threading and concurrency is a generally hard to grasp topic, so there are lots of tutorials and examples online in lots of different languages.

High Performance Computing Carpentry has a workshop. It begins with why you would use parallel computing and then shows how to do some operations on a cluster. You could work through this on the cluster we used in class.

24.7.3. Can this cause a deadlock?

Yes, deadlock is a possible outcome of having multiple threads. The particular program we wrote today, would not, but using threads for more co-dependent components of a program could.

24.7.4. What kinda of signs would be visible that we need to make our program multithreaded?

Researching more examples and then brainstorming a few based on the ones you find is actually a part of the more practice for today's class.

The general case is just when there are two or more things that need to happen "at the same time" not in sequence, even if they can sort of take turns.

24.7.5. One question I have is why would you want to split up your problem among different computers? Is it just so that more parts of your problem can get done at the same time?

High Performance Computing Carpentry has a good description of why you would want to split a program across multiple computers.

25. How does this all come together?

25.1. Logistics and Grading

- I'm a little behind on feedback, will catch up
- Extra Office Hours

- Zoom link is on slack to keep it private
- To be able to revise, work must be submitted by **May 5 at 4pm**. Feedback will be by afternoon May 6.
- All final work must be submitted by **May 10** I will do final grading on May 11.

25.2. Contract Checkin

Create an issue on your grading contract repo using the following as a template.

Edit so that it applies to you, including deleting any sections that do not apply to you and assign or tag @brownsarahm on the issue so that I can get to it easily.

```
I am (not) on track to complete my contract as agreed.


I plan to (catch up/ revise my contract / take an incomplete).

To revise my contract I will:
<!-- consultaiton is optional -->
- [ ] attend office hours to discuss options
- [ ] submit the revised contract as PR by the contract revision deadeline of May 2.


To catch up, I will:
- [ ] tag @brownsarahm in all of my open PRs by 5pm today
- [ ] attend office hour on (dates)
- [ ] complete (items) by Thursday at 5pm
- [ ] complete (items) by May 3rd at 5pm
- [ ] complete (items) by May 5 at 4pm
- [ ] complete (items) by May 10

To complete work after accepting an Incomplete grade I will:
- [ ] complete all activities in my contract by (date)

I have the following questions about my grade/grading in this course:

- [ ] q1
- [ ] q2
```

> **Important**
>
> The last date to change your contract is May 2

> **Important**
>
> If you do not complete your contract, you will get an incomplete which will be changed when you complete work or you will get a low grade (eg no work/D/F) at the deadline for me to change the incomplete

25.3. Testing Integration manager workflow

See the prepare for next class number 1.

Why are merge conflicts expected here?

1. Resolve the merge conflicts
2. combine work with your partner on a fork
3. make PR to main sites
4. I will be merging and then tag you when you should resovle the merge conflicton your PR

> **Important**
>
> Merge conflicts **will** happen. Knowing how to fix them is **very** important. I want you to get practice with it.

25.4. End of Semester Wrap up

Instead of introducing new big ideas this week, we will reivew and work on tying ideas together.

> **Note**
>
> Answers for all of these questions exist in this site

What does threading do?

- [ ] divide a program into multiple processes
- [ ] divides a task across multiple computes
- [ ] divide a process into mutlple parts that are completed separately

Why do we need to lock writing to a global variable when threading a program?

- [ ] otherwise all threads will overwrite incorrectly and only the slowest one will be saved
- [ ] otherwise their start/finish order can impact the calculation
- [ ] it is the only way for a variable to be shared across threads

What is git? *multiple may apply**

A. a file system B. backup software
C. a version control system D. a cloud protocol for collaborating on code

Why do we use version control?

- [ ] to have a cloud backup of code
- [ ] it's a legacy tool for undoing changes before IDEs could do it
- [ ] to be able to separate different versions of code

Why floating point instead of fixed point for fractions ?

- [ ] floating point allows for faster calculations
- [ ] floating point allows more precise representation
- [ ] floating point allows for representation of a broader range of values

Label and put in order the following steps of building C code: (replace `x` with a number and `step` with the name of that step)

```
x. **step** binaries for compiled libraries is combined with the compiled program code
x. **step** code is translated from assembly to machine code
x. **step** code from includes is copied into the file
x. **step** code is translated from a high level language to assembly
```

25.5. Prepare for next class

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Review the questions from class today and in `review.md` write a solution and explanation for your confusion for any that you got wrong. If you missed class, write a solution guide for all of the questions in today's notes. (to reinforce concepts and make sure you understand them correctly)
2. Update your KWL chart by adding 2-3 rows about topics you learned about that are not listed. (to review all concepts in class)
3. Add a new, up to date overview of how you think about computing overallto your KWL repo. Mark the one you made at the beginning as "overview-pre" and this one as "overview-post" either with heading or the file name (to synthesize ideas)
4. Confirm that your jupyterbook works. (we will work with your KWL in class on Thursday)
5. Create PRs to the main sites from the last class's prepare for next class task. (more GitHub practice)

25.6. More Practice

> **Important**
>
> - If you're on track, you can skip the following and Thursday's More Practice
> - if you are not on track: **today's and Thursdays will count for two each, but will be more complex and integrate concepts across multiple class sessions**

1. Make a table that compares and contrasts the unix file system to git *as* a file system on at least 5 aspects (eg header + 5 rows) in `filesystem.md`. Based on this, write a definition of a file system generally and how knowing that these are both file systems helps reinforce concepts and improve understanding.
2. In `whypointers.md` write a blog-post style argument for why understanding pointers as a concept is important for a computer science student even if they will not work directly in a language that uses pointers. Consider your audience to be a student who is in CSC212 and struggling. Use two examples from this class where we relied on the concept of pointers to explain how something worked.
3. Reflect on how thise course impacts programming/debugging skills in `skillup.md`. You can write this as how you think your own skill has improved **or** as if you are convincing another student to take this class. Touch on at least three topics.

25.7. Questions After Class

25.7.1. What is the industry standard between github/gitlab/bitbucket? is there one that is better than the other?

Feature-wise they're relatively comparable, they all do core git and provide hosting. The details of features are somewhat different. The deciding factor though, usually comes down to price and/or a moral position on GitHub being owned by Microsoft and having US Immigration and Customs Enforcement (ICE) as a customer.

26. What do we do next?

26.1. KWL Maintainance

- Jupyter book
- building and publishing it
- taking ownership

26.1.1. Making sure it is a jupyterbook

Check that you have:

- `_config.yml` best practice is to copy the defaults then change the “Book settings” and “# Launch button settings” settings as needed.
- `_toc.yml` follow the instructions and example inthe docs.
- make sure that all of your markdown files are in the `_toc.yml`

26.1.2. Rendering and publishing it with GitHub

Github Actions allow you to run scripts, both specialized and anythign that you can run in bash on GitHub servers and use those results online.

What we are goign to do is add a file in `.github/workflows/publish.yml` To do run jupyterbook on GIthub’s server and then put the contents of the build directory on the gh-pages branch.

```yaml
name: deploy-book

on:
  push:
    branches:
    - main


jobs:
  deploy-book:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    # Install dependencies
    - name: Set up Python 3.8
      uses: actions/setup-python@v1
      with:
        python-version: 3.8

    - name: Install dependencies
      run: |
        pip install -r requirements.txt

    # Build book
    - name: Build the book
      run: |
        jupyter-book build . --builder pdfhtml
        mv _build/pdf/book.pdf _build/html/kwlbook.pdf
        jupyter-book build .

    # Push the book's html to github-pages
    - name: GitHub Pages action
      uses: peaceiris/actions-gh-pages@v3.6.1
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./_build/html
```

And then a `requirements.txt` file to the top level directory so that it knows what to install (see above we told it to install from requiremetns.txt)

```
jupyter-book
matplotlib
numpy
pyppeteer
pandas
lxml
sphinx_fontawesome
sphinx_panels
```

> **Note**
>
> Later, I’ll enable you to to turn on pages if you would like

26.1.3. Create an issue for its future

Use one of the following as the **title** of the issue.

```
I want to take ownership of my KWL repo and make it public
```

```
I want to take ownership of my KWL repo for my own reference
```

```
I want my kwl repo archived
```

26.2. Review

What happens in linking?

- [ ] code from includes is copied into the file
- [ ] binaries for compiled libraries is combined with the compiled program code
- [ ] code is translated from assembly to machine code
- [ ] code is translated from a high level language to assembly

What are three factors that you could use to choose the right langague for a project?

assuming you know or have time and motivation to learn many languages

What is the relationship between a half adder and a full adder?

- [ ] a half adder adds half of the bits in the register and the full adds all
- [ ] a half adder adds two one bit numbers into a two bit number; a full adder adds two one bit numbers and a carry bit into a one bit output and a carry bit
- [ ] a half adder adds two one bit numbers into 1 bit; a full adder adds two one bit nubmers into two bits

26.3. Final Comments

- this course site will remain available to you as a reference
- Future sections will be separate sites, all will be listed on this main site
- All students are responsible for the prepare activities and reflections
- all re responsible for the minimum set of rows in your KWL chart
- more practice grouped by class session
- **May 2** is the last day to change your grading contract
- **May 5** is the last day to submit work that you can revise
- **May 10** AOE[1] is the last day to submit work
- extra office hours go until May 10.

26.4. Feedback

> **Important**
>
> If 70% of the class completes the IDEA survey you will **all** get 1(large)-3(small) free mistakes. What this will mean is that, instead of having you having to find the answer and revise, I will give you the answer as a suggestion and you only have to commit my suggestion to get full credit on it.

26.5. Prepare for future classes

> **Note**
>
> The text in () in these sections is an explanation of *why* that task is assigned

1. Review the questions from today's class and write a solution guide for any that you got wrong with notes about what confused you in `review.md` (to make sure you do not leave with misconceptions)
2. Add `conclusion.md` to your KWL repo. In the file include 2 sections: **summary** with an overall description of what you learned and how this course was/not helpful to you (paragraph or bullet form is fine) **key points** with the top 10 things you want to make sure you remember from this class (to reinforce these concepts *and* serve as a quick reference for yourself in the future)

26.6. More Practice

1. In `surprisingfacts.md` explain *why* each of the following and one additional fact of your own is true. For each fact, include a demo showing that it happens.

```
1. rename and move use the same command in bash
1. `git revert` adds a commit
1. adding to a large number can result in a negative number
1. the following would most likelycause an endless loop `while not(i==.3)`
1. A program without any inputs or randomization in it can give different
results each time it is run
```

2. For two imaginary projects, compare compare and contrast different programming languages and git workflows. Choose the two projects so that the best language and best workflow will be different choices. Include a description of each project, your comparisons, and conclusions in `projectplanning.md`

---

[1] Anywhere on Earth time.

KWL Chart

Working with this repo

> **Important**
>
> - Treat your main branch as what will "published" if you choose to use this as a portoflio of sorts.
> - The feedback branch should only contain material that has been reviewed and approved by the instructors.

> **Tip**
>
> You could apply branch protections on your feedback branch if you like

You may mix these workflows as long as feedback only contains work that has been reviewed and approved.

Workflow 1

1. When you are working on things that are not ready for feedback make a new branch to work on them.
2. If you are working on things that you want feedback on right away you can work directly on main.
3. When work is ready for feedback merge it into main
4. Create a pull request from main into feedback.
5. When PRs are approved, merge them into feedback

Workflow 2

1. work on a specific branch
2. when it is ready for review, create a PR from the item-specifc branch to feedback.
3. when it is approved, merge into feedback, then merge feedback into main.

> **Tip**
>
> After your KWL repo is a jupyterbook, you may want to move your chart into a chart.md from the README. You could then put content from this section in the README to have an in-place reminder.

Minimum Rows

```
# KWL Chart


<!-- replace the  _ in the table or add new rows as needed -->

| Topic | Know | Want to Know | Learned |
| ------| ------- | ------ | ------- |
| Git | _ | _ | _ |
| GitHub | _ | _ | _ |
| Terminal | _ | _ | _ |
| IDE | _ | _ | _ |
| text editors | _ | _ | _ |
|file system | _ | _ |_ |
|bash | _ | _ | _ |
|abstraction | _ | _ | _ |
|programming languages | _ | _ | _ |
|git workflows | _ | _ | _ |
| git branches | _ | _ | _ |
| bash redirects | _ | _ | _ |
|number systems | _ | _ | _ |
| merge conflicts | _ | _ | _ |
| documentation | _ | _ | _ |
| templating | _ | _ | _ |
|bash scripting | _ | _ | _ |
| developer tools | _ | _ | _ |
| networking | _ | _ | _ |
|ssh | _ | _ | _ |
| ssh keys | _ | _ | _ |
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
|logic gates | _ | _ | _ |
| ALU | _ | _ | _ |
| binary operations | _ | _ | _ |
| memory | _ | _ | _ |
| cache | _ | _ | _ |
| register | _ | _ | _ |
| clock | _ | _ | _ |
| Concurrency | _ | _ | _ |
```

Added Files

file	content (link to instructions)	type (prepare/practice)	zone
README	the chart, (or usage)	default	all
gitoffline.md	reflection (extra exercises, add & commit out of order)	prepare (and practice)	grade free
terminal.md	reflection	prepare	grade free
software.md	examine a software project	prepare	grade free
abstraction.md	how you think about abstraction and updates (and reconcilation)	prepare (practice)	grade free
chart.md	your chart (from README)	practice	grade free
reorg.md	ntoes/troubleshooting	practice	grade free
stdinouterr.md	echo and redirect practice	practice	grade free
hardwaresurvey.md	challenges & reminders (more exercises) reflection	prepare and (practice)	grade free
gitlog.txt	git log output	prepare	graded
workflows.md	compare git workflows	practice	graded
gitplumbing.md	map plumbing to porcelain (and memory device)	prepare (and practice)	graded
gitunderstanding.md	how your undersanding has changed	prepare	graded
idethoughts.md	notes on ide usage	prepare	graded
numbers.md	usage of nondecimal number systems (and number conversions)	prepare	graded
hexspeak.md	summarize & generate	prepare	graded
gitstory.md	tutorial of using git for a challenge	prepare	graded
gitstory2.md	tutorial of using git for a challenge	practice	graded
donotcommit.md	tutorial of using git for a challenge	practice	graded
jupyterbooktroubleshooting.md	problems with jupyterbook	prepare	graded
templating.md	how template based engines work	prepare	graded
docs.md	documentation ecosystem of a language other than python	prepare	graded
octal.md	pracitce with octal numbers in file permissions	practice	graded
hpc.md	reflect on HPC systems	practice	graded
ide.md	featrures in most used ide	prepare	graded
languages.md	compare two languages	prepare	graded
favoriteide.md	compare three ides	practice	graded
`gcctips.md`	c compilation notes	prepare	graded
`operators.md`	operators in programming and math	prepare	graded
`bitwise.md`	practice with bitwise operations	prepare	graded
`readingbytes.md`	review how reading bytes works	prepare	graded
`overflow.md`	compare integer overflow	prepare	graded
`cdouble.md`	c program with double comparison	prepare	graded
`overflow.md`	compare integer overflow	practice	graded
`readingbytes.md`	c program with double comparison	practice	graded
`floatexp.md`	c program with double comparison	practice	graded
`multiplication.md`	interpret mulitplication as addition	prepare	graded
`addertypes.md`	compare adders	practice	graded
`systemsabstractions.md`	reflect on an article	prepare	graded
`history-<event>.md`	summarize and evaluate an aspect of computer history	practice	graded
`hardwaremap.md`	visualize current hardware knowledge	practice	graded

file	content (link to instructions)	type (prepare/practice)	zone
systemsprogramming.md	learn about systemsvs application programming	prepare	graded
singlevmultithread.md	compare single vs multi threaded program	prepare	graded
timing.md	experiment with timing of operations	practice	graded
kernelproc.md	check your processes	practice	graded
systeminteraction.md	compare systems libraries in multiple languages	practice	graded
whymultithread.md	learn more applications of threads	practice	graded
review.md	review questions from class April 26 AND April 28	prepare	graded
filesystem.md	compare and contrast the two file systems we saw in class	practice	graded
whypointers.md	explain why the concept of pointers is important with examples	practice	graded
skillup.md	Reflect on how thise course impacts programming/debugging skills	practice	graded
surprisingfacts.md	explain contradictions we saw	practice	graded
projectplanning.md	compare languages and git workflows	practice	graded

Prepare for the next class

Warning

these are listed by the date they were *posted* (eg the content here under Feb 1, was posted Feb 1, and should be done before the Feb 3 class)

below refers to following in the notes

```
import os
from IPython.display import Markdown, display

prep_file_list = sorted(os.listdir('../_prepare/'))
```

```
for prep_file in prep_file_list:
    date_str = prep_file[:-3]
    date_link = '[' + date_str + '](../notes/' + date_str + ')'
    display(Markdown(date_link))
    display(Markdown('../_prepare/' + prep_file))
```

2022-01-25

- More practice with GitHub terminology. Accept this assignment, read through it, and follow the instructions at the end.
- Review these notes, bring any questions you have to class
- Read the syllabus, explore this whole website. Bring questions about the course. Be prepared for a scavenger hunt that asks you not to recall every fact about the course, but to know where to find informatio.
- Think about one thing you've learned really well (computing or not) and how do you know that you know it? (bring your example)

2022-01-27

1. Review these notes, both rendered as html and the raw markdown in the repository.
2. find 2-3 examples of things in programming you have got working, but did not really understand. this could be errors you fixed, or something you just know you're supposed to do, but not why
3. map out your computing knowledge and add it to your kwl chart repo. this can be an image that you upload or a text-based outline.
4. Make sure you have a working environment for next week. Use slack to ask for help.
    - check that you have Python installed with Jupyter, ideally with Anaconda
    - install jupyter book
    - install GitBash on windows (optional for others)
    - make sure you have Xcode on MacOS
    - install the GitHub CLI on all OSs

2022-02-01

1. Complete the classmate issue in your in-class repository.
2. read the notes PR, add or comment on a tip, resource, a bit of history in a sidebar or additional end of class question
3. try using git in your IDE of choice, log any challenges you have on the practice repo (`github-in-class-username`), and tag @sp22instructors on GitHub. You can use either repo we have made in class, or one for an assignment in another course.
4. using your terminal, download your KWL repo and update your 'learned' column on a new branch
5. answer the questions below in a new markdown file, `gitoffline.md` in your KWL on your new branch and push the changes to GitHub
6. Create a PR from your new branch to main **do not merge this until instructed**
7. add your programming challenge(s) you have had as issues to our private repo or to the course website repo if you like. Put one 'challenge/question' per issue so that we can close them as addressed. See last class notes for prompt.
8. Create or comment on a discussion thread in the private repo about the part of CS/ type of programming you like best/what you want to do post graduation.

2022-02-03

1. Review the notes
2. Reorganize a folder on your computer (good candidate may be desktop or downloads folder), using only a terminal to make new directories, move files, check what's inside them, etc. Answer reflection questions (will be in notes) in a new file, `terminal.md` in your kwl repo.
3. Add a glossary to the site to define a term or cheatsheet entry to describe a command that we have used so far.
4. Examine a large project you have done or by finding an open source project on GitHub. Answer the reflection questions in `software.md` in your kwl repo. (will be in notes)

2022-02-08

1. install h/w simulator
2. Add a glossary, cheatsheet entry, or historical context/facts about the things we have learned to the site.

3. Review past classes prep/more practice and catchup if appropriate
4. Map out how you think about data moving through a small program using the levels of abstraction. Add this to a markdown table in your KWL chart repo called `abstraction.md`. If you prefer a different format than a table, that is okay, but put it in your KWL repo. It is okay if you are not sure, the goal is to think through this.

2022-02-10

1. Read these notes and practice with the hardware simulator, try to understand its assembly and walk through what other steps happen. Make notes on what you want to remember most or had the most trouble with in `hardwaresurvey.md`
2. Review and update your listing of how data moves through a program in your `abstraction.md`. Answer reflection questions below.
3. Review the commit history and git blame of a repo in browser- what must be in the .git directory for GitHub to render all of that information? (be prepared to discuss this in class)
4. Fill in the Know and Want to know columns for the new KWL chart rows below.
5. Begin your grading contract, bring questions to class Tuesday.

2022-02-15

1. review the notes and ensure that you have a new, empty repository named test with its branch renamed to main from master.
2. Add the following to your kwl:

```
|git workflows | _ | _ | _ |
| git branches | _ | _ | _ |
| bash redirects | _ | _ | _ |
```

3. Practice with git log and redirects to write the commit history for your kwl chart to a file gitlog.txt and commit that file to your repo.

2022-02-17

1. Review the notes
2. For the core "Porcelain" git commands we have used (add, commit), make a table of which git plumbing commands they use in `gitplumbing.md` in your KWL repo
3. In a `gitunderstanding.md` list 3-5 items from the following categories (1) things you have had trouble with in git in the past and how they relate to your new understandign (b) things that your understanding has changed based on today's class (c) things about git you still have questions about
4. Make notes on *how* you use IDEs for the next week or so using the template `idethoughts.md` file in the course notes.

2022-02-22

1. review the past two classes of notes
2. find 3 more examples of using other number systems, list them in `numbers.md`
3. Read about hexpeak from Wikipedia for an overview and one additional source. In your kwl repo in `hexspeak.md` summarize it in your own words, one interesting fact from your additional source and link to the source you found. Come up with a word or two on your own.
4. (priority) Bring to class a scenario where you think git could help, but we haven't covered yet how to do it. (be prepared to post it to Prismia at the start of class)

2022-02-24

1. Check that jupyter book is installed on your computer and the windows advice
2. Fix any open PRs you have that need to be rebased.
3. In your github in class repo, create a series of commits that tell as story of how you might have made a mistake and fixed it. Use git log and redirects to write that log to a file, `gitstory.md` in your KWL repo and then annotate your story to add in any narrative that didn't fit in the commit

messages. This could be that you made a mistake in your code and used git to recover or that you got your git database to an undesirable state and got it back on track.

2022-03-01

1. review the notes
2. (priority) Convert your KWL repo to a jupyter book. Ignore the builds from your reposiotry. Use the documenation to choose your settings: link it to your repo, do not serve it publicly on github. Be sure that it can build and if you encounter problems, create a `jupyterbooktroubleshooting.md` file and log them with solutions if you find them.
3. complete Feedback survey if you have not alredy.

2022-03-03

1. Review the notes (to reinforce ideas and improve your memory of them)
2. Update your KWL to include all rows listed on the KWL page of the course site. (because tracking your learning deepens your learning)
3. **priority** Make an issue on your grading contract repo that includes: a self-assessment of your progress on your contract so far and a plan going forward. Tag @sp22instructors so that we can help you meet your goals. (so that you get the grade you want)
4. **priority** On Windows, install Putty (we will use this Monday)
5. Write a bash script that checks only if the number of files in your kwl meets or exceeds the minimum (number from list + 2 for toc and config) OR that does something else of your choosing. Add it to your KWL repo in a utils directory. Include Comments. (to practice so that you remember)

2022-03-08

1. Review the notes (to reinforce ideas and improve your memory of them)
2. **priority** add a `## summary` section to your IDE notes where you summarize what you observed in your IDE usage and what questions or what features (we will work with this after break and I want you to prepare before break)
3. Modify the sbatch job we made in class so that you get files back for any errors and output from the script. see the options (to review/practice what we did)

2022-03-10

1. Do what you need to do to finish the semester well: eg rest, catchup on work, both (breaks are important)
2. Review your IDE thoughts right before class on Tuesday March 22 (so you're ready for the activity in class)

2022-03-24

1. Explore the IDE you use most and add `ide.md` to your kwl with notes about which features it does/not have. (to review/reinforce)
2. **priority** Make a table comparing two programming languages you know in `languages.md`. The table should have 3 columns: one titled "Attributes" and one for each languages. Add one row for each attribute and fill in how that is for each language. Use the following list of attributes and add 1-3 more.

```
create a list
loop (with predetermined number of iterations)
loop (until a condition is met)
conditional
create a string variable
cast from one type to another
```

3. read about the parts of the developer survey about languages

2022-03-28

1. For 2 languages from the loved vs dreaded list (one top 5; one bottom 5) read 2-3 posts about why people love/hate that language and summaries the key points on each side. Add this to your kwl repo in `language_love_dread.md`
2. Make sure that you have a C compiler installed on your computer. ideally gcc

2022-03-31

1. Practice using gcc. Repeat steps we did in class, change the order of parameters. Then in `gcctips.md` summarize what you learned as a list of tips and reminders on what the parameters do/why/when you would need them (or not). (to reinforce what we learned)
2. Create `operators.md` and make some notes about what you know about operators. What kinds of operators are you familiar with? Which have you seen in programming? math?
3. Add the following to your kwl chart

```
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
```

2022-04-05

1. Add `bitwise.md` to your kwl and write the bitwise operations required for the following transformations:

```
4 -> 128
12493 - > -12494
127 -> 15
7 -> 56
4 -> -5
```

2. For the following figure out the bitwise operator:

```
45 (_) 37 = 37
45 (_) 37 = 45
3 (_) 5 = 7
6 (_) 8 = 0
10 (_) 5 = 15
```

3. Create `readingbytes.md` and answer the following:

```
1. if a file had the following binary contents, what would it display in the
terminal? Describe how you can figure this out manually and check it using C
or Python. '01110011011110010111001101110100011001010110110101110011'
1. What is the contents of the `sample.bn` if `cat sample.bn` is: ` ¢¶"*`
```

4. Read about integer overflow and in `overflow.md` describe what it is, use an example assuming an 8 bit system.

2022-04-07

1. Write a C program to compare values as doubles and as float (single precision/32bit) to see that this comparison issue is related to the IEEE standard and is not language specific. Make notes and comparison around its behavior and include it in a code cell in `cdouble.md`(to practice)
2. (priority) confirm that you can run the hardware simulator `HardwareSimulator` not `CPUEmulator` that we used before (we will use this in class)

2022-04-12

1. Compare the 2 bit multiplier to the full adder in `multiplication.md`. Use that to explain how it works, relative to the fact that multiplication can be thought of like repeated addition.
2. Study the 8 bit ALU. Try it out and be prepared to answer questions about how it works on Thursday.
3. Read gates out of anything

2022-04-14

1. Read about systems abstractions in CACM. Answer reflection questions in `systemsabstractions.md`:

```
1. How many of these are familiar/not?
1. How has you undertanding of these changed during this course.
1. Do you think you understand this article more now than you would have at
the beginning of the semester
1. Write 3 questions and their answers that could be a quiz to see if someone
understood or had misconceptions about the abstractions in this article.
```

2. Review the notes and add one more historical event's information to the notes using either the computer history museum and/or additonal sources. Check the open PRs to ensure that no one else has submitted the same as you.
3. Update your KWL chart based on the minimum rows list

2022-04-19

1. Check that your KWL is up to date with all required rows and add a section (`##`) below your chart in the same file (either `README` or `chart.md`) that summarizes what you think about computer systems.
2. (priorty) Read about systems vs applications programming. Create `systemsprogramming.md` to summarize the difference and what skills would be diffrent. Which do you have more experience with? What experience do you have with the other? (we will address this in class Thursday)

2022-04-21

1. (**Priority**) Work with a partner to add "advice to future students" testimonials to website for this offering and "why take this class" testimonials to the main course site using an integration manager like workflow. Coordinate to determine who's fork will be used for integration of each . Each of you should clone that fork and add your contribution to **`resources/testimonials.md`** and **`fromstudents.md`** on a branch, `<initials>_testimonial` eg Professor Brown's would be `smb_testimnial`. Then merge the two contributions to a single `testimonial` branch. (both to help future students *and* to practice with branches and collaboration, we will work with these further in class next week)
2. Review the notes.
3. Update your KWL chart. Check that all rows are included.
4. Simulate a more computationally intensive program using the `sleep` function in C and compare the time of a threaded vs single threaded version of the program. Include the two programs, output of your test, explanation text, and conclusions in `singlevmultithread.md` (to better illustrate the impact of the threads)

2022-04-26

1. Review the questions from class today and in `review.md` write a solution and explanation for your confusion for any that you got wrong. If you missed class, write a solution guide for all of the questions in today's notes. (to reinforce concepts and make sure you understand them correctly)
2. Update your KWL chart by adding 2-3 rows about topics you learned about that are not listed. (to review all concepts in class)
3. Add a new, up to date overview of how you think about computing overallto your KWL repo. Mark the one you made at the beginning as "overview-pre" and this one as "overview-post" either with heading or the file name (to synthesize ideas)
4. Confirm that your jupyterbook works. (we will work with your KWL in class on Thursday)
5. Create PRs to the main sites from the last class's prepare for next class task. (more GitHub practice)

2022-04-28

1. Review the questions from today's class and write a solution guide for any that you got wrong with notes about what confused you in `review.md` (to make sure you do not leave with misconceptions)
2. Add `conclusion.md` to your KWL repo. In the file include 2 sections: **summary** with an overall description of what you learned and how this course was/not helpful to you (paragraph or bullet form is fine) **key points** with the top 10 things you want to make sure you remember from this class (to reinforce these concepts *and* serve as a quick reference for yourself in the future)

More Practice

> **Note**
>
> these are listed by the date they were *posted*

```
import os
from IPython.display import Markdown, display

prep_file_list = sorted(os.listdir('../_practice/'))
```

```
for prep_file in prep_file_list:
    date_str = prep_file[:-3]
    date_link = '[' + date_str + '](../notes/' + date_str + ')'
    display(Markdown(date_link))
    display(Markdown('../_practice/' + prep_file))
```

2022-01-27

- (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later, or try getting it to render on your own.
- (optional) read chapter 1 the programmmer's brain. Some of the ideas we talked about today are mentioned there, and it relates to where you're suppoed to be looking for things that you have done, but didn't really understand.
- try adding something to this page or the glossary of the course site or link a glossary term to an occurence of it on the site.

2022-02-01

1. Find the "Try it yourself" boxes in these notes, try them and add notes/ responses under a `## More Practice` heading in your `gitoffline.md` file of your KWL repo.
2. Download the course site repo via terminal.
3. Explore the difference between `git add` and `git commit` try committing and pushing without adding, then add and push without committing. Describe what happens in each case in your `gitoffline.md`

2022-02-03

1. Try to do as many things as possible on the terminal for a whole week.
2. Make yourself a bash cheatsheet (and/or contribute to one on the course site)
3. Read through part 1 of the programmer's brain and try the exercises, especially in chapter 4.

2022-02-08

1. Once your PRs in your KWL are merged so that main and feedback match, pull to updates your local copy. In a new terminal window, navigate there and then move the your KWL chart to a file called `chart.md`. Create a new README files with a list of all the files in your repo. Use `history N` (N is the number of past commands that history will return) and redirects to write the steps you took to `reorg.md`. Review that file to make sure it doesn't have extra steps in it and remove any if needed using nano then commit that file to your repo.
2. find a place where there is a comment in the course notes indicating content to add and submit a PR adding that content. This could be today's notes or a past day's.
3. Add a new file to your KWL repo called `stdinouterr.md` Try the following one at a time in your terminal and describe what happens and explain why or list questions for each in the file. What tips/reminders would you give a new user (or yourself) about using redirects and `echo`?
    - `echo "hello world" > fa > fb`
    - `echo "a test" > fc fd`
    - `> fe echo "hi there"`
    - `echo "hello " > ff world`
    - `<ff echo hello`
    - `fa < echo hello there`
    - `cat`

2022-02-10

1. Complete the `Try it Yourself` blocks above in your `hardwaresurvey.md`.
2. Expand on your update to `abstraction.md`: Can you reconcile different ways you have seen memory before?

2022-02-15

1. ¸ Read about different workflows in git and add responses to the below in a `workflows.md` in your kwl repo. Git Book atlassian Docs
2. Contribute either a glossary term, cheatsheet item, additional resource/reference, or history sidebar to the course website.

```
## Workflow Reflection

1. What advantages might it provide that git can be used with different workflows?
1. Which workflow do you think you would like to work with best and why?
1. Describe a scenario that might make it better for the whole team to use a
workflow other than the one you prefer.
```

2022-02-17

1. Add to your `gitplumbing.md` file explanations of the main git operations we have seen (add, commit, push) in your own words in a way that will either help you remember or behow you would explain it to someone else at a high level. This might be analogies or explanations using other progrmaming concepts or concepts from a hobby. Add this under a subheading `##` with a descriptive title (for example "Git In terms of ")
2. For one thing your understanding changed or an open question you, look up or experiment to find the answer

2022-02-22

1. Read more about git's change from SHA-1 to SHA-256 and reflect on what you learned (questions provided)
2. (priority) In a language of your choice or pseudocode, write a short program to convert, without using libraries, between all pairs of (binary, decimal, hexidecimal) in `numbers.md`. Test your code, but include in the markdown file enclosed in three backticks so that it is a "code block" write the name of the language after the ticks like:

```
```python
# python code
```
```

2022-02-24

1. Find an open source repository and look at the .gitignore file. In `donotcommit.md` reflect on what types of content typically get ignored and why you think they are ignored. If you can't figure out why, try to look it up.
2. Create a second git story for the other type form your first (code mistake or git mistake) in `gitstory2.md`
3. add something to the glossary, cheatsheet or a history sidebar to the notes.

2022-03-01

1. build your kwl repo to a pdf or one other format locally.
2. Add `templating.md` to your KWL repo and explain templating in your own words using other programming concepts you have learned so far. Include in a markdown (same as HTML `<!-- comment -->`) comment the list of CSC courses you have taken for context while we give you feedback.
3. Learn about the documentation ecosystem in another language that you know. In `docs.md` include a summary of your findings and compare and contrast it to jupyter book/sphinx.

2022-03-03

1. **priority** add or link a glossary or cheatsheet, OR add a box with some historical context or extra reading links to the notes from any past class (in order to practice git/github and using jupyter book features, which are similar to other documentation tools, so even if you do not work in a python centric environment the concepts will translate)

2022-03-08

1. **priority** write a script that outputs the first 3 lines of each `.fastq` line and sbatch script that runs it and saves the output to files and e-mails you when it is done. Submit the job. (to reinforce /practice what we saw in class)

2. File permissions are represented numerically often in octal, by transforming the permissions for each level (owner, group, all) as binary into a number. Add `octal.md` to your KWL repo and answer the following. Try to think through it on your own, but you may look up the answers, as long as you link to (or ideally cite using jupyterbook citations) a high quality source.

```
1. Transform the permissions [`r--`, `rw-`, `rwx`] to octal, by treating it as
three bits.
1. Transform the permission we changed our script to `rwxr-xr-x` to octal.
1. Which of the following would prevent a file from being edited by other
people 777 or 755?
```

3. Answer the following in `hpc.md` of your KWL repo: (to think about how the design of the system we used in class impacts programming and connect it to other ideas taught in CS)

```
1. What kinds of things would your code need to do if you were going to run it
on an HPC system?
1. What Sbatch options seem the most helpful?
1. How might you go about setting the time limits for a script? How could you
estimate how long it will take?
```

2022-03-10

1. **optional** Configure ssh keys to your GitHub account (this is actually GitHub's preferred terminal authentication method)

2022-03-24

1. (priority) Compare at least 3 IDEs for working in a single language. Create `favoriteide.md` and include your most important criteria with their rankings, how each IDE meets/does not meet those criteria, and a conclusion of which IDE is the best based on your criteria.

2022-03-28

1. (priority) Describe a type of project where it would be worth it for you to learn a language you have never used before in `newlanguage.md` This should be based in what types of features for the language your project would require and/or what would contribute to the long term health of the project.
2. Try out/learn about one of the following languages that you have not used before, do something small that is typical of that language (eg a toy data analysis in R or): R, Julia, Clojure, Stan, Go, . Answer the following questions:

```
1. What is this language designed for?
1. What Progrmaming paradigm(s) does it support?
1. What about it makes it easy to learn for someone who already knows some
other language (name that language)?
1. What about it is hard to larn for (pick a language) prgorammers?
1. What is its most unique feature(s)?
```

2022-03-31

1. (priority) Write two short programs that do the same thing in different ways and compile them both to assembly (eg using a for vs while loop to sum numbers up to a number). Check the assembly to see if they produce the same thing or if it's different.
2. Add (or link) a glossary term, cheatsheet tip, or historical context to note on the course website.

2022-04-05

1. (priority) Add to `overflow.md` how integer overflow is handled in Python, C, Javascript, and one other language of your choice.
2. Add to `readingbytes.md` and exmaple of how machine code that was a 3 bit instruction followed by an 8 bit address might render.
3. Add a box to the notes for any class that includes historical context of something covered in class or a related topic. Use the tempalte below:

```{admonition} Historical Context
```

2022-04-07

1. In `floatexpt.md` design an experiment using the `fractions.Fraction` class in python that shows helps illustrate how `.1*3 == .3` evaluates to `False` but `.1*4 ==.4` evaluates to `True`. (practice/review)

2022-04-12

1. In `addertypes.md` compare ripple adders and lookahead adders.

```
1. Give a synopsis of each adder type
1. Compare them in terms of time (assume that each gate requires one unite of
time)
1. Compare them in terms of space/cost by counting the total number of gates
required.
```

2. (priority) While we saw many types of gates today, but we actually could get all of the operations needed using only NAND gates. Work out how to use NAND gates to implement a half adder.

2022-04-14

1. Pick one historical event about computers, memory and storage or networking and the web (not limited to what is on those pages) and in, 'history-' describe the event, how it impacted what has come since.
2. Map out what you know about computer hardware in some form of visual or outline in `hardwaremap.md` and bring three questions to class.

2022-04-19

1. (priority) Compare the time of the following operations, investigate what the different operations do/not do, how do the three times compare? and hypothesize explanations in `timing.md`. Which times are constant vs variable? Which vary the most? least? Tip: use a script with loops and see the optionsand write output to a file (to test how these impact your work. *extension idea:* expand the level of detail to include more analysis on this, Try to find operations that increase the user vs system time. You can use basic bash operations or programs you have from other courses)

```
1. wget https://github.com/introcompsys/spring2022/blob/main/img/2022-01-27-
mental_model_advanced.svg
2. wget
https://raw.githubusercontent.com/introcompsys/spring2022/main/img/2022-01-27-
mental_model_advanced.svg
3. kwlcheck
4. kwlfilecheck
5. hello world compiled from C
```

2. Practice exploring the list of processes running on your computer and investigate how much resources different things you do on a regular basis use. Make some general observations In `kernelproc.md`.

2022-04-21

1. Learn about the system libraries in two languages (one can be C or Python, one must be something else). Find the name(s) of the library or libraries. In `systeminteraction.md` make notes for each language about what categories of functions they provide.
2. (**priority**) Research examples of programs using multi-threading besides splitting up a single calculation for time reasons, include three examples in `whymultithread.md`.

2022-04-26

1. Make a table that compares and contrasts the unix file system to git *as* a file system on at least 5 aspects (eg header + 5 rows) in `filesystem.md`. Based on this, write a definition of a file system generally and how knowing that these are both file systems helps reinforce concepts

and improve understanding.

2. In `whypointers.md` write a blog-post style argument for why understanding pointers as a concept is important for a computer science student even if they will not work directly in a language that uses pointers. Consider your audience to be a student who is in CSC212 and struggling. Use two examples from this class where we relied on the concept of pointers to explain how something worked.
3. Reflect on how thise course impacts programming/debugging skills in `skillup.md`. You can write this as how you think your own skill has improved **or** as if you are convincing another student to take this class. Touch on at least three topics.

2022-04-28

1. In `surprisingfacts.md` explain *why* each of the following and one additional fact of your own is true. For each fact, include a demo showing that it happens.

```
1. rename and move use the same command in bash
1. `git revert` adds a commit
1. adding to a large number can result in a negative number
1. the following would most likelycause an endless loop `while not(i==.3)`
1. A program without any inputs or randomization in it can give different
results each time it is run
```

2. For two imaginary projects, compare compare and contrast different programming languages and git workflows. Choose the two projects so that the best language and best workflow will be different choices. Include a description of each project, your comparisons, and conclusions in `projectplanning.md`

Deeper Explorations

If your contract includes that you will complete deeper explorations, this page includes guidance for what is expected.

Deeper explorations can take different forms so the sections below outline some options, it is not a cumulative list of requirements.

Where to put the work?

- If you extend a more practice exercise, you can add to the markdown file that the exercise instructs you to create.
- If its a question of your own, add a new file to your KWL repo.

How to get it reviewed?

Follow the workflows for your kwl repo and tag the instructors for a review.

What should the work look like?

It should look like a blog post or written tutorial. It will likely contain some code excerpts the way the notes do. Style-wise it can be casual, like how you may talk through a concept with a friend or a more formal, academic tone. What is important is that it clearly demonstrates that you understand the material.

For special formatting, use jupyter book's documentation.

Project Information

Proposal Template

If you have selected to do a project, please use the following template to add a section to the end of your `contract.md`

```
## < Project Tite >

<!-- insert a I sentence summary  -->

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solve. this should be 2-5 sentences, it can be bullet points/numbered or a paragraph  -->

### method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? will you do experiments?-->

### deliverables

<!-- list what your project will produce with target deadlines for each-->
```

The deliverables will depend on what your method is, which depend on your goals. It must be approvd and the final submitted will have to meet what is approved. Some guidance:

- any code or text should be managed with git (can be GitHub or elsewhere)
- if you write any code it should have documentation
- if you do experiments the results should be summrized
- if you are researching something, a report should be 2-4 pages in the 2 column ACM format.

This guidance is generative, not limiting, it is to give ideas, but not restrict what you *can* do.

Updates and work in Progress

These can be whatever form is appropriate to your specific project. Your proposal should indicate what form those will take.

Summary Report

This summary report will be added to the grading contract repo as a new file `project_report_title.md` where title is the title from the project proposal.

This summary report have the following sections.

1. **Abstract** a one paragraph "abstract" type overview of what your project consists of. This should be written for a general audience, something that anyone who has taken up to 211 could understand. It should follow guidance of a scientific abstract.
2. **Reflection** a one paragraph reflection that summarizes challenges faced and what you learned doing your project
3. **Artifacts** links to other materials required for assessing the project. This can be a public facing web resource, a private repository, or a shared file on URI google Drive.

Syllabus and Grading FAQ

How much does activity x weigh in my grade?

> There is no specific weight for any activities, because your grade is based on fulfilling your contract. If all items are completed to a satisfactory level, then you earn that grade, if not, then you will be prompted to revise the contract to signal that you are aware you have complted fewer items.

Can I submit this assignment late if …?

> In this class, there are feedback hours instead of deadlines for specific assignments. If your work is not in by the feedback hours, you will get feedback at the next hours. If your grading contract has dated milestones, (eg for a project) post an issue in your contract repo with title `late milestone: <name of milestone>` and in the body proposes when it will be in instead and how this delay will not affect your future work. If you need more dramatic shift, propose a revision to your contract.

I don’t understand the feedback on this assignment

If you have questions about your grade, the best place to get feedback is to reply on the Feedback PR. Either reply directly to one of the inline comments, or the summary.

Be specific about what you think is wrong so that we can expand more.

What should a Deeper exploration look like and where do I put it?

It should be a tutorial or blog style piece of writing, likely with code excerpts or screenshots embedded in it.

an example that uses mostly screenshots

an example of heavily annotated code

They should be markdown files in your KWL repo. I recommend myst markdown.

Git and GitHub

I can’t push to my repository, I get an error that updates were rejected

If your error looks like this…

```
! [rejected] main -> main (fetch first)
error: failed to push some refs to <repository name>
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Your local version and github version are out of sync, you need to pull the changes from github to your local computer before you can push new changes there.

After you run

```
git pull
```

You’ll probably have to resolve a merge conflict

My command line says I cannot use a password

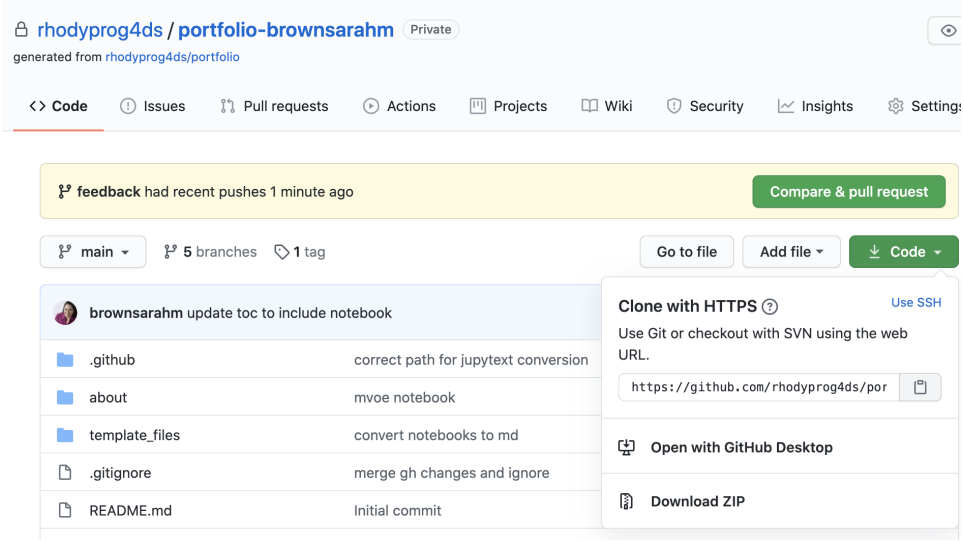
GitHub has strong rules about authentication You need to use SSH with a public/private key; HTTPS with a Personal Access Token or use the GitHub CLI auth

Help! I accidentally merged the Feedback Pull Request before my assignment was graded

That’s ok. You can fix it.

You’ll have to work offline and use GitHub in your browser together for this fix. The following instuctions will work in terminal on Mac or Linux or in GitBash for Windows. (see Programming Environment section on the tools page).

First get the url to clone your repository (unless you already have it cloned then skip ahead): on the main page for your repository, click the green “Code” button, then copy the url that’s show



Next open a terminal or GitBash and type the following.

```
git clone
```

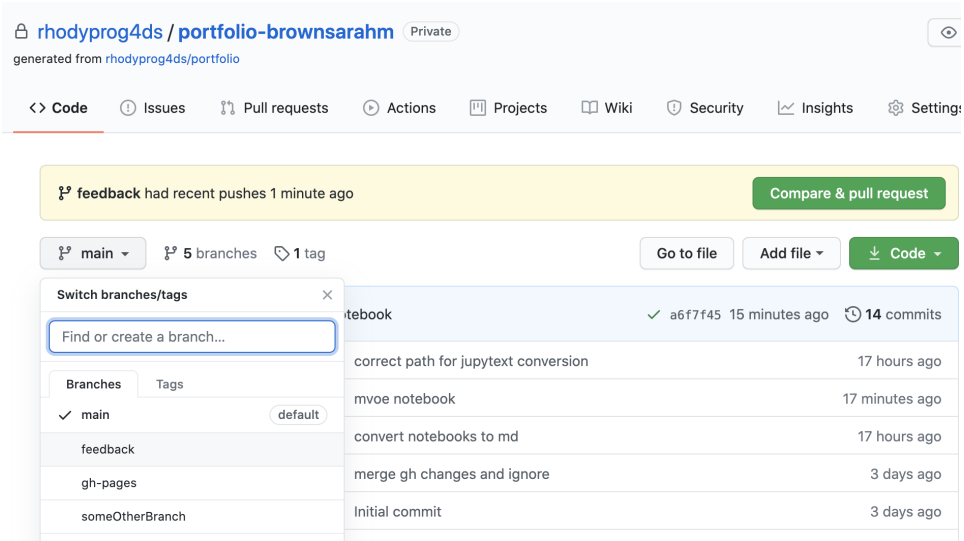
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
git clone https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
```

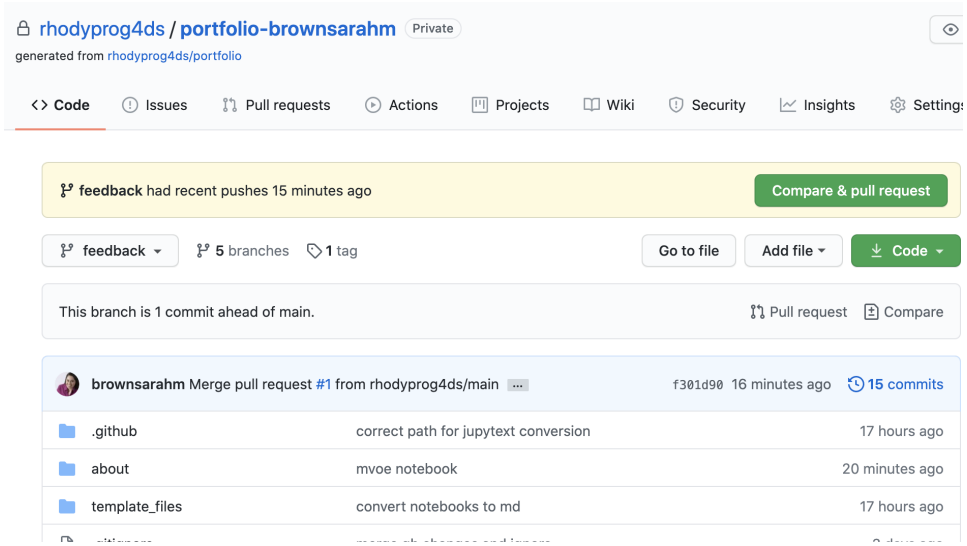
When you merged the Feedback pull request you advanced the `feedback` branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it’s up to date with the `remote` (the copy on GitHub)

```
cd portfolio-brownsarahm
git checkout feedback
git pull
```

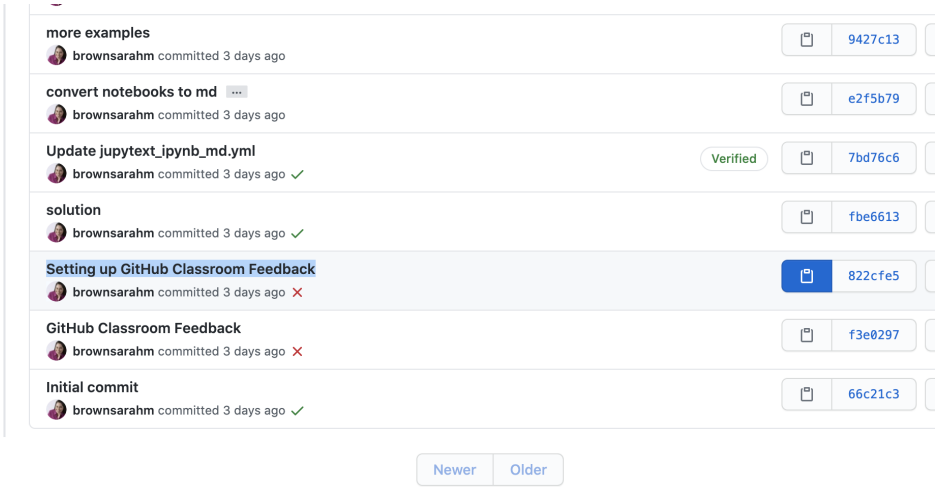
Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and swithc to the feedback branch there. Click on where it says `main` on the top right next to the branch icon and choose feedback from the list.



Now view the list of all of the commits to this branch, by clicking on the clock icon with a number of commits



On the commits page scroll down and find the commit titled “Setting up GitHub Classroom Feedback” and copy its hash, by clicking on the clipboard icon next to the short version.



Now, back on your terminal, type the following

```
git reset --hard
```

then paste the commit hash you copied, it will look something like the following, but your hash will be different.

```
git reset --hard 822cfe51a70d356d448bcaede5b15282838a5028
```

If it works, your terminal will say something like

```
HEAD is now at 822cfe5 Setting up GitHub Classroom Feedback
```

but the number on yours will be different.

Now your local copy of the `feedback` branch is reverted back as if you had not merged the pull request and what’s left to do is to push those changes to GitHub. By default, GitHub won’t let you push changes unless you have all of the changes that have been made on their side, so we have to tell Git to force GitHub to do this.

Since we’re about to do something with forcing, we should first check that we’re doing the right thing.

```
git status
```

and it should show something like

```
On branch feedback
Your branch is behind 'origin/feedback' by 12 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

Your number of commits will probably be different but the important things to see here is that it says `On branch feedback` so that you know you're not deleting the `main` copy of your work and `Your branch is behind origin/feedback` to know that reverting worked.

Now to make GitHub match your reverted local copy.

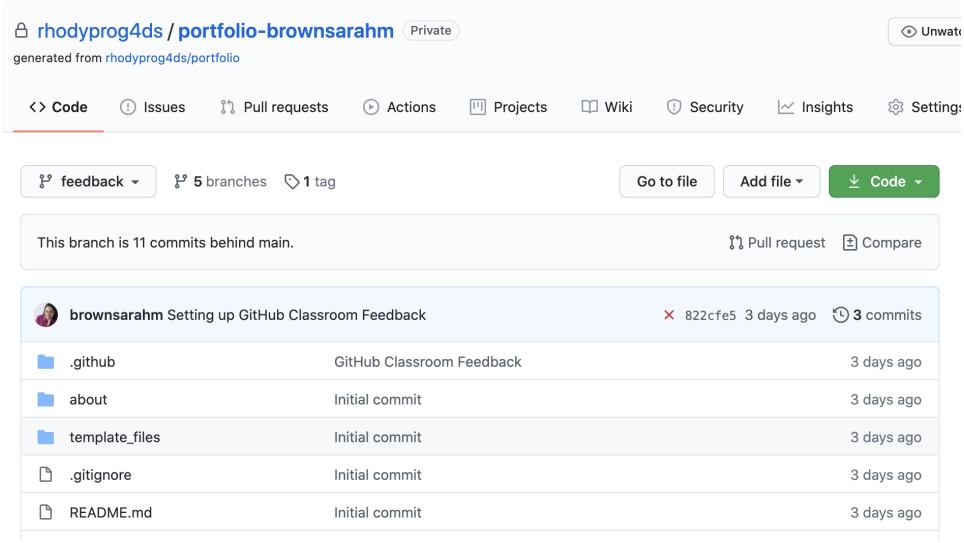
```
git push origin -f
```

and you'll get something like this to know that it worked

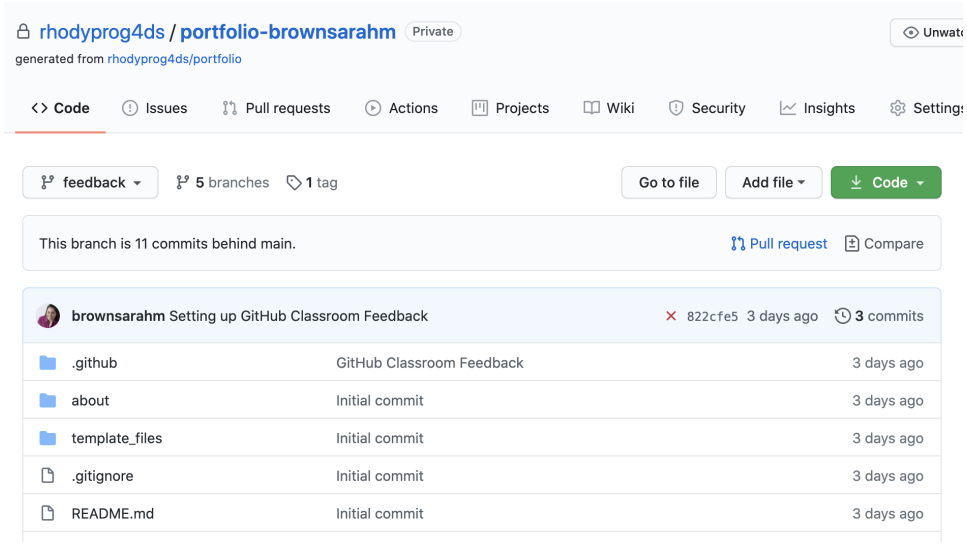
```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

Again, the numbers will be different and it will be your url, not mine.

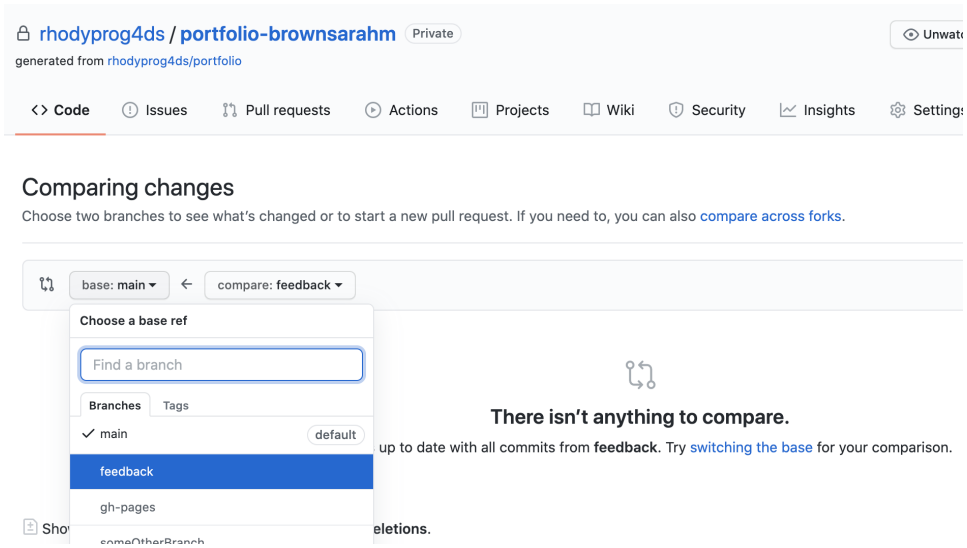
Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, "This branch is 11 commits behind main" your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the changes you made to main (11 for me) and the 1 commit for merging main into feedback. Also the last commit (at the top, should say "Setting up GitHub Classroom Feedback").



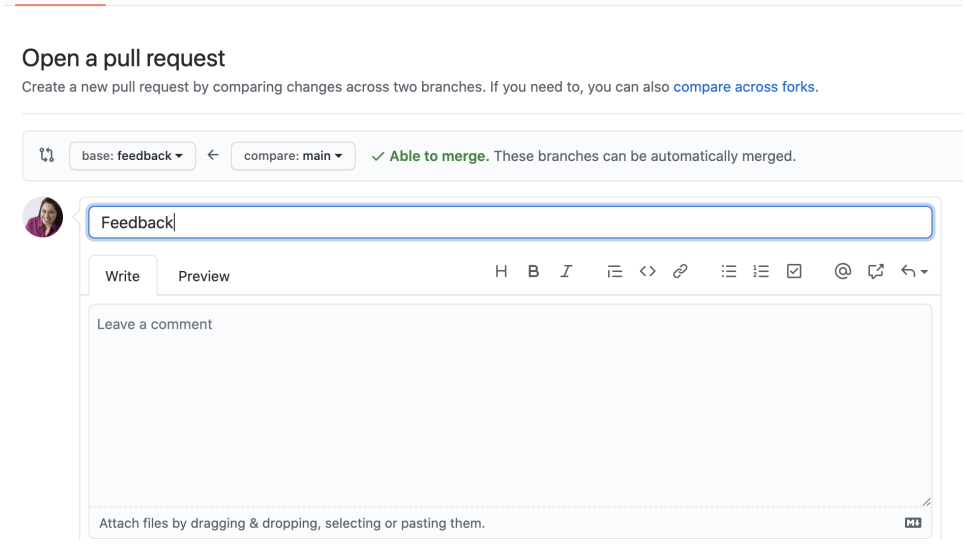
Now, you need to recreate your Pull Request, click where it says pull request.



It will say there isn't anything to compare, but this is because it's trying to use `feedback` to update `main`. We want to use `main` to update `feedback` for this PR. So we have to swap them. Change base from `main` to `feedback` by clicking on it and choosing `feedback` from the list.



Then the change the compare `feedback` on the right to `main`. Once you do that the page will change to the "Open a Pull Request" interface.



Make the title "Feedback" put a note in the body and then click the green "Create Pull Request" button.

Now you're done!

If you have trouble, create an issue and tag `@@rhodyprog4ds/fall20instructors` for help.

For an Assignment, should we make a new branch for every assignment or do everything in one branch?

```
Doing each new assignment in its own branch is best practice. In a typical software development flow once the codebase is
stable a new branch would be created for each new feature or patch. This analogy should help you build intuition for this
GitHub flow and using branches.  Also, pull requests are the best way for us to give you feedback. Also, if you create a
branch when you do not need it, you can easily merge them after you are done, but it is hard to isolate things onto a branch
if it's on main already.
```

Glossary

> 💡 **Tip**
>
> We will build a glossary as the semester goes on. When you encounter a term you do not know, create an issue to ask for help, or contribute a PR after you find the answer.

add (new files in a repository)

the step that stages/prepares files to be committed to a repository from a local branch

bitwise operator

an operation that happens on a bit string (sequence of 1s and 0s). They are typically faster than operations on whole integers.

git

a version control tool; it's a fully open source and always free tool, that can be hosted by anyone or used without a host, locally only.

GitHub

a hosting service for git repositories

Git Plumbing commands

low level git commands that allow the user to access the inner workings of git.

Git Workflow

a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner

git init name_of_repo

used to create a git repo in a directory (which can then be uploaded to github or another online directory

push (changes to a repository)

to put whatever you were working on from your local machine onto a remote copy of the repository in a version control system.

pull (changes from a repository)

download changes from a remote repository and update the local repository with these changes.

repository

a project folder with tracking information in it in the form of a .git file

shell

a command line interface; allows for access to an operating system

terminal

a program that makes shell visible for us and allows for interactions with it

ROM (Read-Only Memory)

Memory that only gets read by the CPU and is used for instructions

directory

a collection of files typically created for organizational purposes

.gitignore

a file in a git repo that will not add the files that are included in this .gitignore file. Used to prevent files from being unnecessarily committed.

templating

templating is the idea of changing the input or output of a system. For instance, the Jupyter book, instead of outputting the markdown files as markdown files, displays them as HTML pages (with the contents of the markdown file).

ssh keygen

allows computers to safely connect to networks (such as when we used an ssh key to clone our github repos)

hashing

putting an input through a function and getting a different output for every input (the output is called a hash; used in hash tables and when git hashes commits).

hash function

the actual function that does the hashing of the input (a key, an object, etc.)

SHA 1

the hashing function that git uses to hash its functions (found to have very serious collisions (two different inputs have same hashes), so a lot of software is switching to SHA 256)

git objects

something (a file, directory) that is used in git; has a hash associated with it

tree objects

type of git object in git that helps store multiple files with their hashes (similar to directories in a file system)

HEAD

the branch that is currently being checked out (think of the current branch)

merge

putting two branches together so that you can access files in another branch that are not available in yours

Compiled Code

code that is put through a compiler to turn it into lower level assemlby language before it is executed. must be compiled and re-executed everytime you make a change.

interpreted code

code that is directly executed from a high level language. more expensive computationally because it cannot be optimized and therefore can be slower.

Linker

a program that links together the object files and libraries to output an executable file.

integreated development environment

also known as an IDE, puts together all of the tools a developer would need to produce code (source code editor, debugger, ability to run code) into one application so that everything can be done in one place. can also have extra features such as showing your file tree and connecting to git and/or github.

bitwise operator

an operation that happens on a bit string (sequence of 1s and 0s). They are typically faster than operations on whole integers.

floating point number

the concept that the decimal can move within the number (ex. scientific notation; you move the decimal based on the exponent on the 10). can represent more numbers than a fixed point number.

fixed point number

the concept that the decimal point does not move in the number (the example in the notes where if we split up a bit in the middle and one half was for the decimal and the other half was for the whole number. Cannot represent as many numbers as a floating point number.

Language Specific References

Python

- Python

Cheatsheet

Patterns and examples of how to accomplish frequent tasks. We will build up this section together over the course of the semester.

Basic Bash file operations

Move one folder up:

```
cd ..
```

Move at the top of your directory:

```
cd
```

Move to the specified directory:

```
cd directory
```

Create a file:

```
touch file_name.ext
```

Text editor:

```
nano file_name.ext
```

Display content:

```
cat file_name.ext
```

Create a new folder:

```
mkdir new_folder_name
```

Move a file:

```
mv file_name.ext folder_name_file_is_going_moved_to
```

Move multiple files:

```
mv file_name1.ext file_name2.ext file_name3.ext folder_name_files_are_going_moved_to
```

List files in the current directory:

```
ls
ls -hl  //displays rw-r-r
ls -G - //folders are colorized
ls -hlG //displays rw-r-r and folders are colorized
```

Show your currect directory:

```
pwd
```

Remove a file:

```
rm file_name.ext
```

Copy a file:

```
cp file_name.ext copied_file_name.ext
```

> **Note**
>
> 1. In every command you can add your directory/ location (ex. docs/file_name.ext).
> 2. "." dot symbolizes our current location and "…" two dots, one level up in the directory tree.
> 3. "*" represents any number of unknown characters; creates a pattern (ex. rm pyt*.py removes all files that start on 'pyt' and end with extension py).

Delete an empty directory:

```
rmdir directory
```

Since you can't delete a directory with files in it you need to recursively delete the folder and its contents. The `-R` is a recursive declaration which tells the terminal to delete the folder, the files within the folder, subfolders, files in the subfolder etc. Source

Delete everything in a directory without confirmation:

```
rm -R directory
```

The `-i` is a flag that prompts you if you want to remove each separate file in the directory.

Delete everything in a directory with confirmation:

```
rm -iR directory
```

List the contents of the directory:

```
ls
```

Git status displays the current state of the repository relative to the working directory. Shows the differences between the index file and the current directory.

```
git status
```

Redirect Output: to overwrite an existing file or a create a new file with the given file name: `echo "send output to file.txt" > file.txt` to append the output to already existing file or create a new file with the given file name:

```
echo "append output to file.txt >> file.txt
```

Git log allows you to see all of the hashes of your previous commits in your branch:

```
git log
```

Wildcard / Kleene star

The wildcard character in bash `*` works by expanding in place to separate arguments that match whatever pattern you're writing.

Example, given a directory `stuff/`:

```
stuff/
  a.py
  b.py
  c.py
  other.txt
  another.md
  nested_folder/
```

If we were to run `ls *.py` while in the `stuff/` directory, the command actually run by the computer is `ls a.py b.py c.py`. This also works for commands like `mv`. I.e. `mv *.py nested_folder/` runs `mv a.py b.py c.py nested_folder/`

This is video explains what a half-adder is

General Tips and Resources

This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

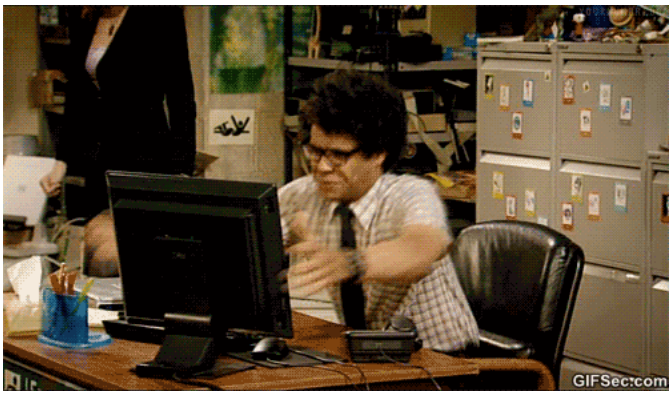
on email

- how to e-mail professors

How to Study in this class

In this page, I break down how I expect learning to work for this class.

I hope that with this advice, you never feel like this while working on assignments for this class.



Why this way?

A new book that might be of interest if you find programming classes hard is the Programmers Brain As of 2021-09-07, it is available for free by clicking on chapters at that linked table of contents section.

Learning requires iterative practice. It does not require memorizing all of the specific commands, but instead learning the basic patterns.
Using reference materials frequently is a built in part of programming, most languages have built in help as a part of the language for this reason. This course is designed to have you not only learn the material, but also to build skill in learning to program. Following these guidelines will help you build habits to not only be successful in this class, but also in future programming.

Learning in class

> **Important**
>
> My goal is to use class time so that you can be successful with *minimal frustration* while working outside of class time.

Programming requires both practical skills and abstract concepts. During class time, we will cover the practical aspects and introduce the basic concepts. You will get to see the basic practical details and real examples of debugging during class sessions. Learning to debug something you've never encountered before and setting up your programming environment, for example, are *high frustration* activities, when you're learning, because you don't know what you don't know. On the other hand, diving deeper into options and more complex applications of what you have already seen in class, while challenging, is something I'm confident that you can all be successful at with minimal frustration once you've seen basic ideas in class. My goal is that you can repeat the patterns and processes we use in class outside of class to complete assignments, while acknowledging that you will definitely have to look things up and read documentation outside of class.

Each class will open with some time to review what was covered in the last session before adding new material.

To get the most out of class sessions, you should have a laptop with you. During class you should be following along with Dr. Brown. You'll answer questions on Prismia chat, and when appropriate you should try running necessary code to answer those questions. If you encounter errors, share them via Prismia chat so that we can see and help you.

After class

After class, you should practice with the concepts introduced.
This means reviewing the notes: both yours from class and the annotated notes posted to the course website.
When you review the notes, you should be adding comments on tricky aspects of the code and narrative text between code blocks in markdown cells.
While you review your notes and the annotated course notes, you should also read the documentation for new modules, libraries, or functions introduced in that class. We will collaboratively annotate notes for this course. Dr. Brown will post a basic outline of what was covered in class and we will all fill in explanations, tips, and challenge questions. Responsibility for the main annotation will rotate.

If you find anything hard to understand or unclear, write it down to bring to class the next day or post an issue on the course website.

Getting Help with Programming

This class will help you get better at reading errors and understanding what they might be trying to tell you. In addition here are some more general resources.

Asking Questions

One of my favorite resources that describes how to ask good questions is this blog post by Julia Evans, a developer who writes comics about the things she learns in the course of her work and publisher of wizard zines.

Describing what you have so far

> **Note**
>
> A fun version of this is rubber duck debugging

Stackoverflow is a common place for programmers to post and answer questions.
As such, they have written a good guide on creating a minimal, reproducible example.

Creating a minimal reproducible example may even help you debug your own code, but if it does not, it will definitely make it easier for another person to understand what you have, what your goal is, and what's working.

Getting Organized for class

The only **required** things are in the Tools section of the syllabus, but this organizational structure will help keep you on top of what is going on.

Your username will be appended to the end of of the repository name for each of your assignments in class.

File structure

I recommend the following organization structure for the course:

```
CSC392
  |- notes
  |- kwl-char-username
  |- spring2022
  |- ...
```

This is one top level folder will all materials in it. A folder inside that for in class notes, and one folder per repository that you work on.

Finding repositories on github

Each assignment repository will be created on GitHub with the `introcompsys` organization as the owner, not your personal acount. Since your account is not the owner, they do not show on your profile.

Your assignment repositories are all private during the semester. At the end, you may take ownership of your portfolio[^pttrans] if you would like.

If you go to the main page of the organization you can search by your username (or the first few characters of it) and see only your repositories.

> **Warning**
>
> Don't try to work on a repository that does not end in your username; those are the template repositories for the course and you don't have edit permission on them.

By Professor Sarah M Brown