About this Site

Contents

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

This class meets MW 4:30-5:45 in Engineering Building Room 045.

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

> **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.

> **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.

> **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

> **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.

> **Ram Token Opportunity**
>
> Chances to earn ram tokens are highlighted this way.

Basic Facts

About this course

About this syllabus

This syllabus is a *living* document. You can get notification of changes from GitHub by “watching” the 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.

About your instructor

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

Dr. Sarah M Brown is a third 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

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

> **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).

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

> **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.

- 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, Fall 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.

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, whether it takes one try or multiple tries.

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 can 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.

The course is designed for you to *succeed* at a level of your choice. No matter what level of work you choose to engage in, you will be expected to revise work until it is correct. As you accumulate knowledge, the grading in this course is designed to be cumulative instead of based on deducting points.

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 assignment 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 contracts for you to use.

Most work will be small, frequent activities, but for an A you will also do larger, more in depth activities.

All contracts will include maintaining a KWL Chart for the duration of the semester, coming to class prepared, participating in class activities, and collaborating with peers to maintain reference materials.

Grading Contract Reference

Sample Contracts

Creative Sample A Grading Contract

To earn this grade I will:

- attend class prepared or makeup in class and preparation activities asynchronously
- review class notes regularly
- keep my KWL chart up to date
- complete all review and prepare activities in my KWL repo as directed
- complete priority more practice tasks for at least 10 class sessions after the grade free zone
- complete two projects:
    - one on tools of the trade
    - one on software infrastructure or hardware

For each project I will:

- (if needed) consult during office hours to develop the idea
- submit a proposal to this repository for approval
- submit regular updates and work in progress for review and revision
- complete the project as agreed
- submit a summary report with links as appropriate to this repository

Guided Sample A Grading Contract

To earn this grade I will:

- attend class prepared or makeup in class and preparation activities asynchronously
- review class notes regularly
- keep my KWL chart up to date
- complete all review and prepare activities in my KWL repo as directed
- complete all of the more practice tasks for at least 16 of the class sessions after the grade free zone
- complete a deeper exploration on one `more practice` task or related question of my own for at least 10 classes after the grade free zone (approximately once per week).

Creative Sample B Grading Contract

To earn this grade I will:

- attend class prepared or makeup in class and preparation activities asynchronously
- review class notes regularly
- keep my KWL chart up to date
- complete all review and prepare activities in my KWL repo as directed
- complete a deeper exploration on one `more practice` task or related question of my own for at least 16 classes after the grade free zone (approximately once per week).

Guided Sample B Grading Contract

To earn this grade I will:

- attend class prepared or makeup in class and preparation activities asynchronously
- review class notes regularly
- keep my KWL chart up to date
- complete all review and prepare activities in my KWL repo as directed
- complete all of the more practice tasks for at least 16 of the class sessions after the grade free zone.

For each deeper exploration I will write up a report with references and/or a tutorial style post with code excerpts or detailed steps and images as appropriate.

Sample C Grading Contract

To earn this grade I will:

- attend class prepared or makeup in class and preparation activities asynchronously
- review class notes regularly
- keep my KWL chart up to date
- complete all review and prepare activities in my KWL repo as directed

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 questions than we can tackle in one class, but will give the general idea of how the class will go.

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

We'll spend a few classes doing an overview where we go through each topic in a little 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

> **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 multiple times with a slightly different framing each time. This will both connect ideas, give you chance to practice recalling (more recall practice improves long term retention of things you learn), and give you a chance to learn things in different ways.

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

Tenative Schedule

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 overview. More Details will be in other parts of the site, linked to here.

Date	Key Question	Prepation	Activities
2022-09-07	What are we doing this semester?	Create GitHub and Prismia accounts, take stock of dev environments	introductions, tool practice
2022-09-12	How does knowledge work in computing?	Read through the class site, notes, reflect on a thing you know well	course FAQ, knowledge discussion
2022-09-14	How do I use git offline?	review notes, reflect on issues, check environment, map cs knowledge	cloning, pushing, terminal basics
2022-09-19	Why do I need to use a terminal?	review notes, practice git offline 2 ways, update kwl	bash, organizing a project
2022-09-21	What are the software parts of a computer system?	practice bash, contribute to the course site, examine a software project	hardware simulator
2022-09-26	What are the hardware parts of a computer system?	practice, install h/w sim, review memory	hardware simulation
2022-09-28	How does git really work?	practice, begin contract, understand git	grading contract Q&A, git diff, hash
2022-20-03	What happens under the hood of git?		git plumbing and more bash (pipes and find)
2022-10-05	Why are git commit numbers so long?	review, map git	more git, number systems
2022-10-12	How can git help me when I need it?	reveiw numbers and hypothesize what git could help with	git merges
2022-10-17	How do programmers build documentation?	review git recovery, practice with rebase, merge, revert, etc; confirm jupyterbook is installed	templating, jupyterbook
2022-10-19	How do programmers auotmate mundane tasks?	convert your kwlrepo	shell scripting, pipes, more redirects, grep
2022-10-24	How do I work remotely ?	install reqs, reflect on grade, practice scrip	ssh/ ssh keys, sed/ awk, file permissions
2022-10-26	How do programmers keep track of all these tools?	summarize IDE reflections	IDE anatomy
2022-10-31	How do Developers keep track of all these tools?	[compare languages you know]	
2022-11-02	How do we choose among different programming languages?	[install c compiler]	
2022-11-07	What happens when I compile code?		
2022-11-09	Why is the object file unreadable?	[what are operators]	bits, bytes, and integers/character representtion
2022-11-14	What about non integer numbers?		floating point representation
2022-11-16	Where do those bitwise operations come from?	[review simulator]	gates, registers, more integer

Date	Key Question	Prepation	Activities
2022-11-21	What actually is a gate?		physics, history
2022-11-23	How do components work together?		memmory, IO, bus, clocks,
2022-11-28	(sub)		
2022-11-30	(sub)		
2022-12-05			
2022-12-07			

Table 1 Schedule

Grading Policies

Deadlines

You will get feedback on items at the next feedback period after it is submitted. During each feedback hours (twice per week) you can get feedback on new submissions from up to 2 class sessions and revision feedback on an unlimited number of submissions.

> **Important**
>
> Work does not have specific deadlines, to give you more flexibility, but to ensure timely feedback and to be fair to me at the end of the semester, there is a limit of how much material you can get feedback at a time. The 2 class session limit means that you should aim to complete things within about 1 week most of the time, but no more than 2 weeks to ensure that all of your work can be reviewed.

Makeup Work

If you have extenuating circumstances and need to submit a large amount of work at once, first submit a PR to your grading contract outlining your plan to get caught back up for approval. Requests will typically be approved, but having a plan is required.

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 specific 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_1855/2146052215.py:1: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  help_df.style.hide_index()
```

Day	Time	Location	Host
Tuesday	2:30-4:15pm	online	Mark
Wednesday	7-8:30pm	online	Dr. Brown
Thursday	2:30-4:15pm	online	Mark
Friday	3:30-4:30pm	in person	Dr. Brown

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. Welcome and Introduction

1.1. Introductions

You can see more about me in the about section of the syllabus.

I look forward to getting to know you all better.

1.2. Some Background

- What programming environments do you have?
- What programming environments are you most comfortable with?

This information will help me prepare

1.3. My focus is for you to learn

- that means, practice, feedback, and reflection
- you should know that you have learned
- you should be able to apply this material in other courses

1.4. Getting started with KWL charts

Your KWL chart is where you will start by tracking what you know now/before we start and what you want to learn about each topic. Then you will update it throughout the semester.

Today we did the following:

1. Accept the assignment to create your repo: KWL Chart
2. Edit the README (only file there) to add your name by clicking the pencil icon (editing a file step 2)
3. adding a descriptive commit message (editing a file step 5)
4. created a new branch (named `add_name`) (editing a file step 7-8)
5. added a message to the Pull Request (pull request step 5)
6. Creating a pull request (pull request step 6)
7. Clicking Merge Pull Request

> **Further Reading**
>
> GitHub Docs are really helpful and have screenshots
>
> - editing a file
> - pull request

1.5. Git and GitHub terminology

We also discussed some of the terminology for git. You can review that with the GitHub Practice Assignment. We will also come back to these ideas in greater detail later.

1.6. What is this course about?

In your KWL chart, there are a lot of different topics that are not obviously related, so what is this course really about?

- practical exposure to important tools
- design features of those tool categories
- basic knowledge of many parts of the CS core
- focus on the connections

We will use learning the tools to understand how computer scientists think and work.

Then we will use the tools to examine the field of Computer Science top to bottom (possibly out of order).

1.6.1. How it fits into your CS degree

In CSC110, you learn to program in python and see algorithms from a variety of domain areas where computer science is applied.

> **Tip**
>
> knowing where you've been and where we're going will help you understand and remember

Then in CSC 340 and 440 you study the algorithms more mathematically, their complexity, etc.

In CSC211, 212, you learn the foundations of computer science: general programming and data structures.

Then in 301, 305, 411, 412 you study different aspects of software design and how computers work.

In this class, we're going to connect different ideas. We are going to learn the tools used by computer scientists, deeply. You will understand why the tools are the way they are and how to use them even when things go wrong.

1.7. Programming is Collaborative

There are two very common types of collaboration

- code review (workign independently and then reviewing)
- pair programming (sitting together and discussing while writing)

We are going to build your skill in the *code review* model. This means you need to collaborate, but collaboration in school tends to be more stressful than it needs to. If students have different goals or motivation levels it can create conflict. So **you will have no group graded work** but you will get the chance to work on something together in a low stakes way.

You will have a "home team" that you work with throughout the semester to build a glossary and a "cookbook" of systems recipes.

Your contributions and your **peer reviews** will be assessed individually for your grade, but you need a team to be able to practice these collaborative aspects.

> **Important**
>
> team formation survey

1.8. Review Today's Class

1. More practice with GitHub terminology
2. Review the notes after I post them

1.9. Prepare for Next Class

- Read the syllabus, explore website
- Bring questions about the course
- Think about one thing you've learned really well (computing or not) and how do you know that you know it? (bring your example)
- Post an introduction to your classmates on our discussion forum

1.10. Questions After Class

> **Important**
>
> I group the questions by category and paraphrase some of them. I do this to combine questions that make more sense for me to answer as one question, to make them more concise, and to make sure no personal information ends up on this site

1.10.1. On the material

1.10.1.1. Where do repositories go once added?

When you create a repository on GitHub it stays on GitHub's servers until you make copies of it elsewhere. Next week, we will see how to do that. You can find the most recent repositories that you have worked on on the left sidebar of github.com. You can find everything for our class on the course GitHub organization page, which is also linked in BrightSpace.

1.10.1.2. what is the definition of git?

We will build up our definition of git over the next few classes, but so far it is a version control system.

1.10.1.3. Is there a very large difference between git and github?

GitHub is a Microsoft owned company and platform for hosting git repositories. `git` is an open source version control system. You can run it locally without a cloud host, or through many different cloud hosts like, Bitbucket, GitLab or even self-host a git server via for example Gogs or Gitlite

1.10.2. On what we will cover

1.10.2.1. How in depth will this class go into networking?

Just a little bit, we will cover some networking topics, but not in great depth.

1.10.2.2. Will this class make me GitHub proficient?

Yes, proficient in git and GitHub.

1.10.3. Logistics

1.10.3.1. What is the best thing about this class?

I'm biased, so I won't answer this directly, but I will reach out to some former students to collect answers.

1.10.3.2. What types of assignments will there be?

You will have mostly short answer reflection questions to submit, and many will require you to d some shell scripting or git command lines. You will have some small code exercises, but mostly small modifications to programs and to run and evaluate the output of them.

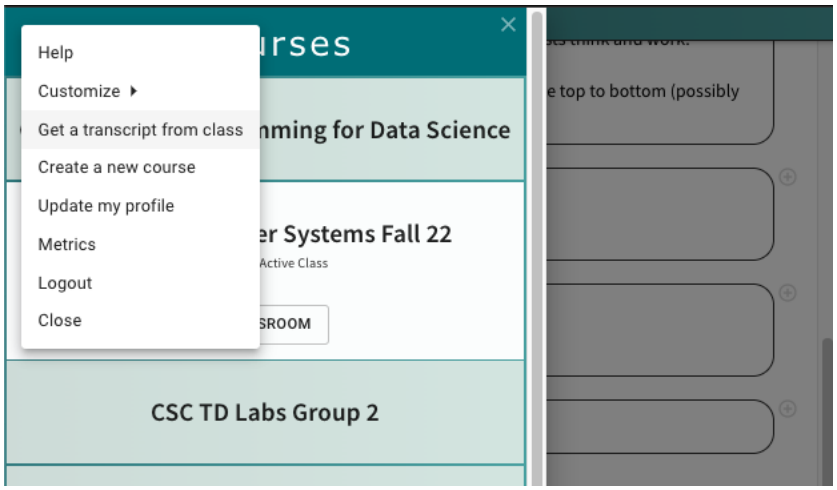
1.10.3.3. what kind of coding will we be doing?

This course is *about* programming, but it's not actually centered on a lot of new programming. We will run some python code and some C code. We will generate some HTML and CSS that we do not need to edit.

We will do a lot of shell scripting though. This is one of the things past students say the learned the most.

1.10.3.4. Will we be able to go back and review the things that we went over in class? As in, will there be resources on Brightspace to allow us to do this?

Not on Brightspace, but you can always get the transcript from Prismia and there will be notes like this after each class. Use the > menu in the top left corner and then click the three bars menu and select 'Get a transcript'. Then choose the course, pick the date and time and it will generate a transcript.



1.10.3.5. How many hours should we be spending outside of class to study?

For a 4 credit class, you should expect to spend approximately a total of 180 hours over the course of the semester, including class time. We will have approximately 2.5*14 = 35 hours of class,so there are 145 hours left, which works out to about 10 hours per week outside of class.

For this class, I expect you to use that time approximately as follows:

- 1 hour preparing for each class (2x per week)
- 2 hours reviewing notes & doing review exercises after each class
- 2-3 hours doing more practice exercises to get additional practice
- 3-4 hours extending and experimenting with things from class for deeper explorations

This means that I expect it to be fair for you to earn an A. If the review or prep are taking you much longer, please reach out to Mark or me so that we can help figure out where you're stuck. If things consistently take you to long, you might be doing something the long way, or I may have assigned more than I expected and need to cut back.

2. Course Logistics and Learning

2.1. Syllabus Review

- Read the navigation on the left carefully

2.1.1. 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?

Something went wrong in an assignment repo on GitHub, what should you check before asking for help?

2.1.2. Class template

In class:

1. Memory/ understanding check
2. Review/ clarification as needed
3. New topic demo with follow along, tiny practice
4. Review, submit questions

Outside of class:

1. Build your glossary and cookbook
2. Review Notes
3. Practice material that has been taught
4. Activate your memory of related things to what we will cover
5. Read/ watch videos to either fill in gaps or learn more details
6. Bring questions to class

(practice extending will vary depending on what grade you are working toward)

2.1.3. Grade Tracking

We will use a GitHub project to track your grade. Create a project on the course organization that is named `grading-<username>` where `<username>` is your GitHub username. We will help you populate it.

2.2. What does it mean to study Computer 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.

From ACM Transactionson Computer Systems

> ACM Transactions on Computer Systems (TOCS) presents research and development results on the design, specification, realization, behavior, and use of computer systems. The term "computer systems" is interpreted broadly and includes systems architectures, operating systems, distributed systems, and computer networks. Articles that appear in TOCS will tend either to present new techniques and concepts or to report on experiences and experiments with actual systems. Insights useful to system designers, builders, and users will be emphasized.

"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.

> **Important**
>
> In this course, we will take the time to understand all of this stuff. This means that we will use a different set of strategies to study it than we normally see in computer science.

We are going to be studying aspects *of* computer systems, but to really understand them, we also have to think about how and why they are the way they are. We will therefore study in a broad way.

We will look at blogs, surveys of developers, and actually examine the systems themselves.

2.3. Mental Models and Learning

2.3.1. 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.

We can visualize with concept maps.

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

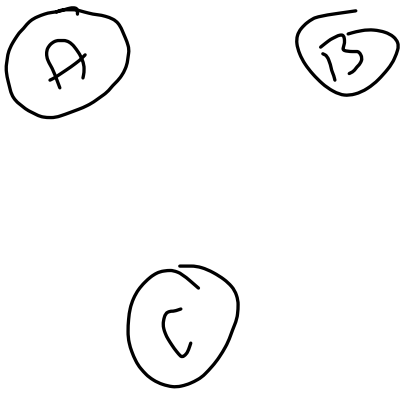


Fig. 2.1 a novice mental model is disconnected and has few concepts

As we learn more, they become more connected.

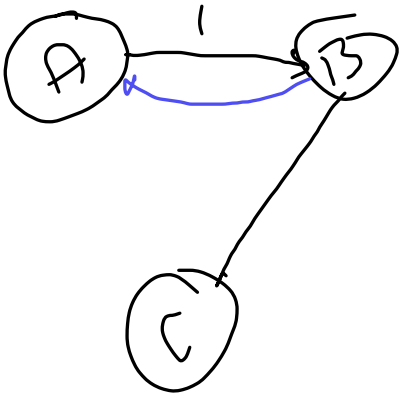


Fig. 2.2 a compententmental model starts to have some connections, with relationships between the concepts.

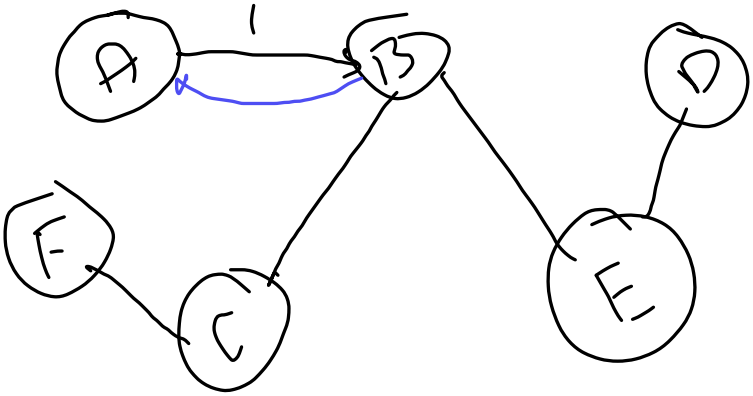


Fig. 2.3 an expert mentla model is densley connected and has more concepts in it.

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

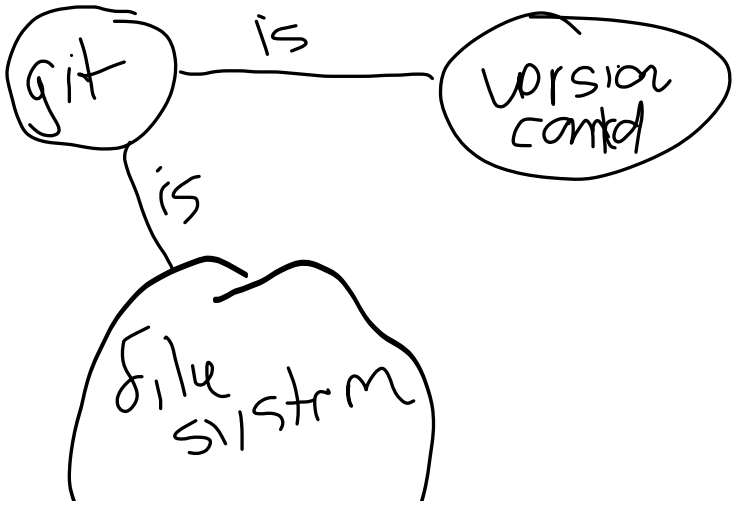


Fig. 2.4 a small concept map showing that git is an instance of both a file system and a version control system.

2.4. 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

2.4.1. 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.

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.

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.

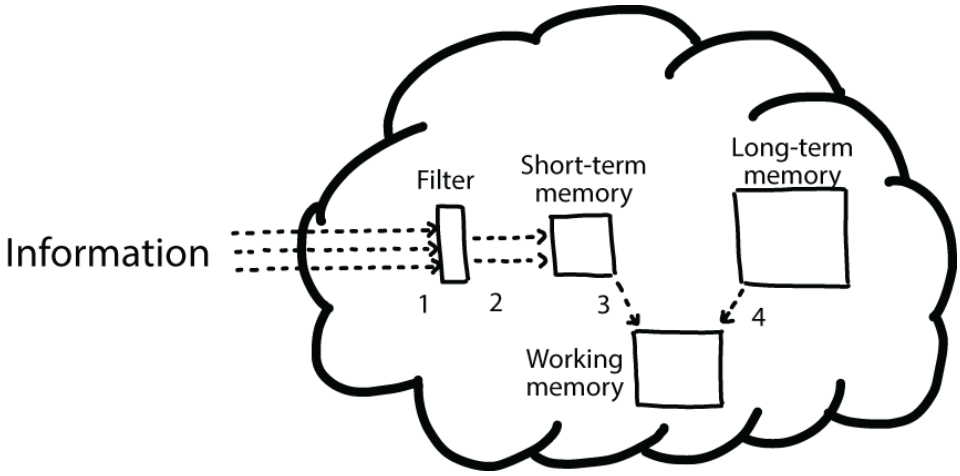


Fig. 2.5 An overview of the three cognitive processes that this book covers: STM, LTM, and working memory. The arrows labeled 1 represent information coming into your brain. The arrows labeled 2 indicate the information that proceeds into your STM. Arrow 3 represents information traveling from the STM 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.

2.4.2. 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.

2.4.3. 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.

2.5. Review Today's Class

> **Note**
>
> You are responsible for these actions, but they will be checked at varying times

1. review notes after they are posted, both rendered and the raw markdown
2. 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. (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later
3. fill in hte first two columns of your KWL chart

2.6. Prepare 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. 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. Add them to our course Discussions in General. Start a new thread and/or comment on others if theirs are related to yours.
2. Make sure you have a working environment for next week. Use slack to ask for help.
    - install GitBash on windows (optional for others)
    - make sure you have Xcode on MacOS
    - install the GitHub CLI on all OSs

2.7. Prepare for Next Class

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

1. read Chapter 1, "Decoding your confusion while coding" in The Programmer's Brain add a file called brain.md to your kwl repo that summarizes your thoughts on the chapter and how, if at all, it changes how you think about debugging and learning to program.

2.8. Questions After Class

2.8.1. Homework

2.8.1.1. what is today's homework?

Above, in the review, Prepare, and More Practice sections.

2.8.1.2. Are we filling out our KWL charts as homework?

yes

2.8.1.3. what are rendered notes and raw notes?

This is the notes. If you are viewing it at introcompsys.github.io/… this is the rendered version. If you are viewing it at https://raw.githubusercontent.com.introcompsys/… then it is the raw version. The source version is at github.com

2.8.1.4. Is chapter 1 of the programmer's brain meant to just be part 1?

Chapter 1 is "Decoding your confusion while coding" and it is also available to listen to.

2.8.1.5. How do we install the gitbash and the CLI.

At the links above. There are different instructions per operating system, but those sites should give you the right verion.

2.8.2. Tools and Content

2.8.2.1. Do we have to regularly update git and anything else that is local to our computers?

Yes, but probably not within one semester. I usually update mine about each semester, so that I stay on the new version that I have students install. When I was not teaching, I only updated when I learned about new major updates somewhere.

2.8.2.2. What is mermaid syntax?

It's a plain text syntax for diagrams. Read more at the link.

2.8.2.3. When it comes to the programming aspects of this course, exactly what programming language(s) will we be using?

We will not be writing a lot of code from scratch. You will observe things you do in other classes you are taking. We will write bash shell scripts. There will be a few programming problems where you can chose any language to write something out and a few where you have to modify code that you are given in C/++ or Python. We will also read machine code and assembly, but not write code in those languages.

2.8.2.4. are we just learning commands with xcode/gitbash to use it in combination with github

We will learn command that we can use from a local terminal in bash to do a lot of helpful things. We will also learn git commands that can be used with GitHub or with any other git client. The git commands we learn will also change less frequently than the GitHub website or desktop applications.

2.8.2.5. Why do we need Gitbash for this course?

GitBash is like a translator that allows windows computers to udnerstand bash, linux (including MacOS) automatically do. GitBash provides a bash terminal for Windows. Bash is the most common shell scripting language and while learning others can be useful, since bash is the most common, it's the *most* useful. Also, it is old enough that most others will have similar structure, so being good at bash will help you learn other shell scripts as well.

Also, because teaching a class only works if we all use the same language.

However, you could instead use windows subsystem for linux or install linux on a flash drive

2.8.2.6. How is the git terminal different to the one I would use for my own other classes like for C++.

We will use git on a bash-supporting terminal.

Also, understanding what a terminal is, is something we will come back to over the next few classes.

2.8.3. Grading

2.8.3.1. How does the grading work?

I will give you a tempalte, you will write a contract, I will approve or recommend edits. Once you have an approved contract, you do all of the work in th econtract, correctly and completely and you earn the grad you contracted for.

2.8.3.2. where do I get graded ?

Most of your work will go into the KWL repo.

2.8.3.3. Do I create a separate repo for the glossary or should I wait till we get more info

Wait for more info

2.8.3.4. I would like to know more about how the letter grade we choose will effect the workload of the course. For example if I were to choose the letter C and my friend chooses

the letter A, will my friend get more assignments to boost the workload?

Earning higher grades requires deeper understanding, so this does require some additional work to give me evidence of that deeper understanding.

---

[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 can I use GitHub offline?

3.1. Get Organized

Opening different terminals

- terminal app on mac, use 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

We can move around and examine the computer's file structure using shell commands.

`cd` is for change directory. We can use the relative path to get to where we want to go. We can see what files and folder are at our curent location with `ls` if we need a reference.

```
cd Documents/inclass/
```

We can use ls in the folder we get to. I chose to go to a place where I save content I use during my classes that I teach.

```
ls
```

I already have a folder for the other class:

```
prog4ds
```

I need a folder for this class, so I will make one with `mkdir`

```
mkdir systems
```

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

```
pwd
```

It prints out the *absolute* path, that begins with a `/` above, we used a relative path, from the home directory.

```
/Users/brownsarahm/Documents/inclass
```

> **Checkin**
>
> What is the absolute path of the home directory?

Next I will go into the folder I just made

```
cd systems/
```

3.2. Issues and Commits

create a test repo for today's class

First we're going to see how issues and commits relate.

Let's create the README on github and make a pull request with `closes #1` in the PR message.

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
- from the issue, we can see what the changes were that made are supposed to relate to this
- we can still comment on an issue that is already closed.

3.3. Authenticating with GitHub and cloning a repo

We have two choices to Download a repository:

1. clone to maintain a link using git
2. download zip to not have to use git, but have no link

For a public repo, it won't matter, you can use any way to download it that you wuold like, but for a private repo, we need to be authenticated.

Depending on how you have authenticated with GitHub, a different version of the URL will be highlighted.

For today, if you have ssh keys already set up, use that.

3.3.1. 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
4. GitBash built in authentication

Or we can use use the GitHub CLI tool to authenticate.

On Mac with GitHub CLI:

```
gh auth login
```

On Windows, Try the clone step and then follow the authentication instructions.

On Mac, clone after you are all logged in.

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

```
Cloning into 'github-inclass-brownsarahm'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

We can also see that it created a new folder:

```
ls
```

```
github-inclass-brownsarahm
```

3.4. What is in a repo?

We can enter that folder

```
cd github-inclass-brownsarahm/
```

and see what is inside

```
ls
```

```
README.md
```

Notice that the `.github/workflows` that we see on GitHub is missing, that is because it is hidden. All file names that start with `.` are hidden.

the `-a` option allows us to see them

```
ls -a
```

We also see some special “files”, `.` the current location and `..` up one directory

```
.               ..              .git            .github         README.md
```

3.5. Relative paths

We can see clearly where `..` goes by printing our our location before and after changing directory to `..`

```
pwd
```

```
/Users/brownsarahm/Documents/inclass/systems/github-inclass-brownsarahm
```

```
cd ..
pwd
```

```
/Users/brownsarahm/Documents/inclass/systems
```

3.6. Adding a file from the command line

First back to the repo directory

```
cd github-inclass-brownsarahm/
```

We will make an empty file for now, using `touch`

```
touch about.md
```

> **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.

git tells us exactly what happened:

```
ls
```

We can se the two folders

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

We can see the file, but what does git know?

```
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)
        about.md

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

First we have to add it to a staging are to make a batch of files (or only one) that will commit.

```
git add .
```

THen we check in with git again

```
git status
```

```
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
```

THen we can commit the file

```
git commit -m "create about"
```

Git returns to us the commit number, with the message and a summary

```
[main 3c9980a] create about
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 about.md
```

and again checkin

```
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
```

Now we see that the local copy is ahead of GitHub (aka origin), so we need to push to make the changes go to GitHub.

```
git push
```

```
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 307 bytes | 307.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/introcompsys/github-inclass-brownsarahm.git
   ec3dd02..3c9980a  main -> main
(base) brownsarahm@github-inclass-brownsarahm $
```

3.7. Review

1. Follow along with the classmate issue in your inclass repo from today. Work with someone you know or find a collaborator in the Discussion board
2. read the notes
3. try using git in your IDE of choice, Share a tip in the course discussion repo

3.8. Prepare

1. Make a list of 3-5 terms you have learned so far and try defining them in your own words.
2. using your terminal, download your KWL repo and update your 'learned' column on a new branch **do not merge this until instructed**
3. answer the questions below in a new markdown file in your KWL

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?
```

3.9. More Practice

1. Download the course website and your group repo via terminal. Try these on different days to get "sapced repetition" and help remember better.
2. 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

3.10. Questions at the end of class

• can you add additional arguments to most git commands? ex git commit -…

• why can't we make all of these files on github instead of using gitbash?

• is there any advantages to using the github website vs using the github commands on your terminal

• One question I have is, how would I edit a text file using bash

• When working with a team what are ways we can prevent merging conflicts?

• will we do the bulk of our work in-console or in an IDE, or a mix of both?

• none

• If we are confused about what is due for next class, which locations should we look? I think right now there are multiple, unless I am wrong about that.

• I accidentally used the push command before the config command. Is that a problem? Is there anything I have to change?

4. Why Do I Need to Use a terminal?

1. replication/automation
2. more universal
3. cloud computing/ HPC often only have a terminal
4. some more powerful actions are only available on a terminal

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

```
cd Documents/inclass/systems/github-in-class-brownsarahm/
```

4.1. Review

We use `git status` to check on the state of a git repo.

If we get an error that says this is not a git repo, it means there is no `.git` directory in our current working directory or its parents.

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

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

```
cd .git
(base) brownsarahm@.git $ ls
COMMIT_EDITMSG  description     info            packed-refs
HEAD            hooks           logs            refs
config          index           objects
(base) brownsarahm@.git $ cd ..
```

Parents mean working to the left in the path and children are to the right.

```
pwd
```

So `systems` is a child of `inclass` and the parent directory of `github-inclass-brownsarahm`

```
/Users/brownsarahm/Documents/inclass/systems/github-inclass-brownsarahm
```

4.2. Scenario

> **Note**
>
> A common question is about 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

```
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
```

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

```
ls
```

Now we have all of these files, named in abstract ways to signal hypothetical 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
```

```
cat README.md
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
```

We can add contents to files with `echo` and `>>`

```
echo "Sarah Brown" >> README.md
```

```
cat README.md
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
Sarah Brown
```

4.2.1. one versus two >>

```
cat about.md
```

```
echo "a sample project" >> about.md
```

```
 echo "testing one >" > about.md
```

One *writes* a file and two *appends*

```
cat about.md
testing one >
```

```
 echo "|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 |" >> README.md
```

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.

```
cat README.md
```

```
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
Sarah Brown
|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 |
```

Note

using the open quote " then you stay inside that until you close it. when you press enter the command does not run until after you close the quotes

```
 echo " kasdlkfjsdljf
> kjsdfksdj
> sdfjdsl
> fsklsdjf
> "
 kasdlkfjsdljf
kjsdfksdj
sdfjdsl
fsklsdjf
```

You can use the up arrow to repeat a command

```
 echo " kasdlkfjsdljf
kjsdfksdj
sdfjdsl
fsklsdjf
" >> junk
```

and we can see that it keeps the new line characters from the terminal in the file.

```
 cat junk
 kasdlkfjsdljf
kjsdfksdj
sdfjdsl
fsklsdjf
```

```
rm junk
```

4.3. Making Directories

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

```
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
```

```
ls
```

and again check what happened

```
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
```

4.4. 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/
```

```
mv example.md docs/
```

4.4.1. Getting help in bash

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

For GitBash:

```
mv --help
```

for *nix (including macos)

```
man mv
```

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

> **Important**
>
> press `q` to exit the program that starts when you do this.

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.

4.4.2. Moving multiple files with patterns

let's look at the list of files again.

```
ls
```

```
API.md                  _toc.yml                overview.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          docs                    test_imp.py
Untitled01.ipynb        helper_functions.py     tests
Untitled02.ipynb        important_classes.py    tests_abc.py
_config.yml             mymodule
```

```
cat README.md
```

```
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
Sarah Brown
|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 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.

> **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.

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

```
 ls
API.md                  about.md                overview.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          helper_functions.py     test_imp.py
Untitled01.ipynb        important_classes.py    tests
Untitled02.ipynb        mymodule                tests_abc.py
```

4.4.3. 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
```

```
mv test_* tests/
```

> **Note**
>
> In class I did the mv tests* before renaming and we did not have to rename. This way is cleaner, but that way worked, albiet with an error

```
ls
```

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

```
ls tests/
test_alt.py     test_help.py    test_imp.py     test_abc.py
```

```
ls
```

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

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

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

4.5. Working with relative paths

Let's review our info again

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

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 `..`
```bash
mv setup.py ..
```

and then go back

```
cd ..
```

Now we'll move the last few docs files.

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

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

4.6. 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
```

```
ls mymodule/
__init__.py             alternative_classes.py  important_classes.py
abstract_base_class.py  helper_functions.py
```

4.7. 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/overview.md
```

```
cp about.md docs/
```

```
ls docs/
_config.yml     about.md        overview.md
_toc.yml        example.md      philosophy.md
```

```
cat about.md
testing one >
```

```
cat docs/about.md
testing one >
```

4.8. Removing files

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

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

```
rm Untitled*
```

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

```
 mv API.md docs/
```

Tab completion

If we type

```
rm U
```

then it canot tab complete the whole file name we get only

```
rm Untitled
```

because there are multiple. If we prest tab twice in a row, it will return the list of possible options

```
Untitled.ipynb
Untitled01.ipynb
Untitled02.ipynb
```

and re-seed the input with

```
rm Untitled
```

so we can type whatever else we want to add to pick the one we want.

4.9. Git Order of operations

> **Important**
>
> We did not push in class, but you are supposed to so I added some tips here

To create and switch to a new branch at once you can use

```
git checkout -b newbranch
```

where `newbranch` is the name of your new branch

```
git status
```

Then you can see which branch you are on with `git status`

```
On branch newbranch
...
```

4.10. 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?

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

4.11. Review today's 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. Make a PR that adds a glossary entry to your team repo to define a term we have learned so far. Create an issue and tag yourself to "claim" a term and check the issues and PRs that are open before your.
4. Review past classes activities (eg via the activities section on the left) and catchup if appropriate

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

```
### Terminal File moving reflection
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
```

4.12. Prepare 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. 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)
2. install h/w simulator
3. map out how you think about data moving through a small program and bring it with you to class (no need to submit)

```
## 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?
```

4.13. More Practice

1. Once your PRs in your kwl are merged, add a Table of Contents to the README with relative links to each file
2. Add cheatsheet entry to your team repo to do something of interest with git or shell. Make an issue for it first and assign yourself so that your team mates know you are working on that topic.

4.14. Questions After Class

4.14.1. What's the difference in creating a file using touch vs echo?

`touch` creates an empty file `echo` writes content to a place. We used with with `>` and `>>` to redirect that output to a file for today, but we will learn more about that later.

4.14.2. I thought you could restore the state of the system using snapshots? Say if you remove something or something gets corrupted

You can restore if a snapshot was created, not all computers have that, and most recovery snapshots are not created very frequently (like once a day or week).

4.14.3. what are other things I can do with the terminal?

There are lots of small programs and you can pipe them all together. We will come back to the shell in the coming weeks.

4.14.4. What are examples of what we are learning used in a professional setting?

Most companies use git to track their code and most dev jobs you will need to log into a server that only has a shell at some point.

4.14.5. if you commit something offline and then close the terminal, can you push it next time you log on or is it lost?

Yes, the file is saved in your computer. You do not actually even *need* to ever push it to a remote to be using git as version control.

4.14.6. How can we edit files on the terminal?

To edit files in more detailed ways, you use a text editor, some are built for the command line. We will see one later.

4.14.7. Where am I supposed to add the "terminal.md" file?

To your KWL repo

4.14.8. Show should the team repo be laid out and should we each determine our responsibilities for them? What should be expected to be inside the team repo?

The repo has some outline in it. Remember it is *not* a team project where you are graded together. You will have various activities assigned there throughout the semester in order to practice collaborating. Every student will need to do every role, so that you get to learn them all.

4.14.9. When will grading start?

Next week, we'll start your grading contracts on Wednesday.

5. How should I use git to stay organized this class?

> **Warning**
>
> this is currently only the raw output from the shell session. Annotation will be added tomorrow

5.1. How do I work with branches?

Let's go back to the github iclass repo.

```
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
        modified:   about.md

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

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

I had not pushed my content after Monday, so I have alot of files that are both not staged and fully untracked.

We do not want to commit direcly to main in general. Working with a branch is better, becuase it gives us more option.

We can use the `-b` option for git checkout to both create a new branch and switch to it.

```
git checkout -b organtzation
```

and git replies that it did what we asked.

```
Switched to a new branch 'organtzation'
```

```
git status
```

> **Note**
>
> Notice that this time in git status it does not compare to origin. this is because the new branch does not have a remote

```
On branch organtzation
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:   about.md

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

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

Now

```
git add .
```

we can commit right after staging, since we konw that is what we want.

```
git commit -m 'from inclass org acitivyt'
```

and it will go a bunch of things, beacuse we made a lot of changes.

```
[organtzation 8b62af8] from inclass org acitivyt
 23 files changed, 69 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/about.md
 create mode 100644 docs/example.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 overview.md
 create mode 100644 setup.py
 create mode 100644 tests/test_alt.py
 create mode 100644 tests/test_help.py
 create mode 100644 tests/test_imp.py
 create mode 100644 tests/tests_abc.py
 create mode 100644 typescript
```

Once we commit it all, we want to get it to GitHub.

```
git push
```

```
fatal: The current branch organtzation has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin organtzation
```

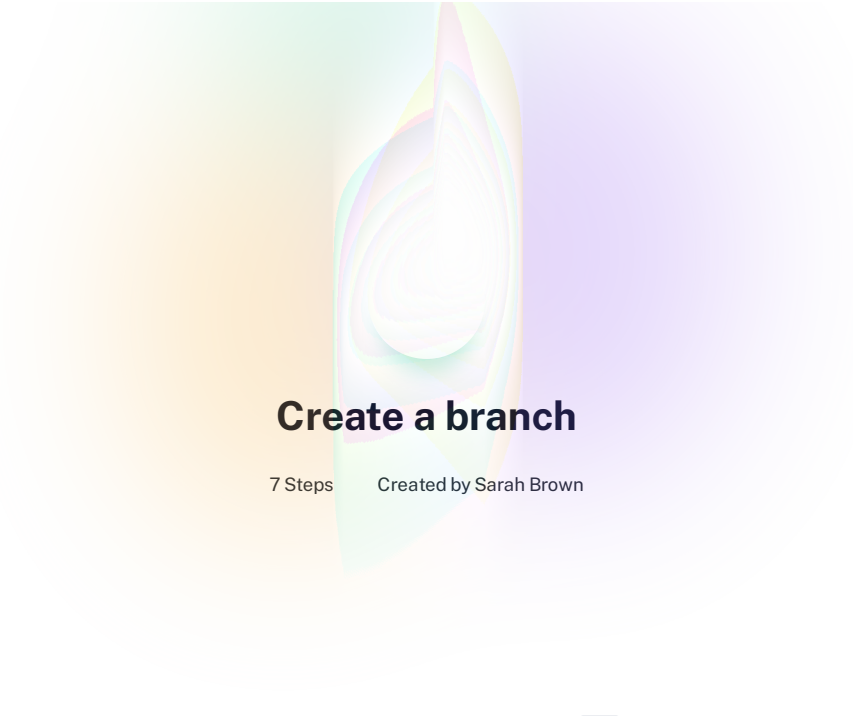
and then we can follow what git said to do

```
git push --set-upstream origin organtzation
```

```
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 1.68 KiB | 1.68 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'organtzation' on GitHub by visiting:
remote:      https://github.com/introcompsys/github-inclass-
brownsarahm/pull/new/organtzation
remote:
To https://github.com/introcompsys/github-inclass-brownsarahm.git
 * [new branch]      organtzation -> organtzation
branch 'organtzation' set up to track 'origin/organtzation'.
```

5.2. What happens when we start a new branch on GitHub.

Create a branch from an issue:



Get Started

Sarah made this Scribe in 22 seconds. Learn how.

```
git fetch
```

this step dowloads the content that is on the new branch

```
From https://github.com/introcompsys/github-inclass-brownsarahm
 * [new branch]      2-create-an-about-file -> origin/2-create-an-about-file
```

we use git satus to see what else it did

```
git status
```

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

nothing to commit, working tree clean
```

and see that it did not change our local status, or location.

```
git branch
```

```
  main
* organtzation
```

but we can see the new branch isn not even in our local repo, though it has been fetched. SO we should checkout

```
git checkout 2-create-an-about-file
```

```
branch '2-create-an-about-file' set up to track 'origin/2-create-an-about-file'.
Switched to a new branch '2-create-an-about-file'
```

this did a few things. It switched our local repference point, made our local directory match the remote branch, and it set the new branch to track the origin branch.

5.3. What if we edit the file in two places?

First, we edit the file locally

```
echo "test" >> about.md
```

```
git status
```

```
On branch 2-create-an-about-file
Your branch is up to date with 'origin/2-create-an-about-file'.

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")
```

```
cat about.md
test
```

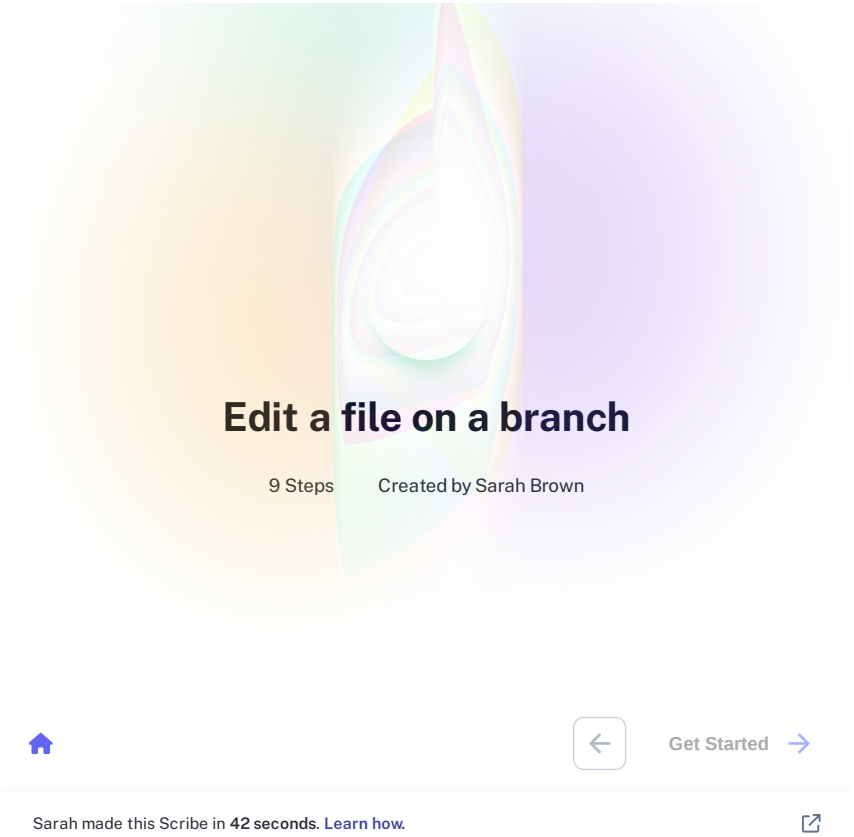
and add and commit the changes

```
git add .
```

```
git commit -m 'start about'
```

```
[2-create-an-about-file 30c4748] start about
 1 file changed, 1 insertion(+)
```

Then edit the file in your browser.



Now we can try to push it from the local copy

```
git push
```

```
To https://github.com/introcompsys/github-inclass-brownsarahm.git
 ! [rejected]        2-create-an-about-file -> 2-create-an-about-file (fetch first)
error: failed to push some refs to 'https://github.com/introcompsys/github-inclass-
brownsarahm.git'
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.
```

We can't push because we ahve a merge conflict.

So first, we have to pull

```
git pull
```

Now it tells us more about the merge conflict.

```
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

We can try the rebase option. We'll talk more about how this works in detail later, but it basically tries to undo one set of commits, apply the others then apply the first set on top. If it cannot, then there's still a conflict.

```
git pull --rebase
```

```
Auto-merging about.md
CONFLICT (content): Merge conflict in about.md
error: could not apply 30c4748... start about
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --
abort".
Could not apply 30c4748... start about
```

And we cannot, so we have more to do.

```
git status
interactive rebase in progress; onto 8ce03aa
Last command done (1 command done):
   pick 30c4748 start about
No commands remaining.
You are currently rebasing branch '2-create-an-about-file' on '8ce03aa'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   about.md

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

We will use nano to edit.

```
nano about.md
```

5.3.1. Resolving a Merge conflict

```
>>>>>>> HEAD
test
========
I am a Professor
<<<<<<<< 30c4748
```

Notes:

- HEAD is the current directory's content
- the other branch is indicated by its last commit number

To fix it, we manually edit the file for it to be what we want

```
test
I am a Professor
```

Then we can finish resoling

```
git status
```

```
interactive rebase in progress; onto 8ce03aa
Last command done (1 command done):
   pick 30c4748 start about
No commands remaining.
You are currently rebasing branch '2-create-an-about-file' on '8ce03aa'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   about.md

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

we add and commit:

```
git add .
```

```
git status
```

```
interactive rebase in progress; onto 8ce03aa
Last command done (1 command done):
   pick 30c4748 start about
No commands remaining.
You are currently rebasing branch '2-create-an-about-file' on '8ce03aa'.
  (all conflicts fixed: run "git rebase --continue")

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

```
git commit -m 'keep both changes'
```

```
[detached HEAD d2f136b] keep both changes
 1 file changed, 2 insertions(+)
```

```
git status
```

```
interactive rebase in progress; onto 8ce03aa
Last command done (1 command done):
   pick 30c4748 start about
No commands remaining.
You are currently editing a commit while rebasing branch '2-create-an-about-file' on
'8ce03aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working tree clean
```

but after that, it still says that it is in the interactive rebase mode, because there is one final step to do.

```
git rebase --continue
```

```
Successfully rebased and updated refs/heads/2-create-an-about-file.
```

and finally

```
git status
```

```
On branch 2-create-an-about-file
Your branch is ahead of 'origin/2-create-an-about-file' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

we see it is all set and ready to go.

```
git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 285 bytes | 285.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/introcompsys/github-inclass-brownsarahm.git
   8ce03aa..d2f136b  2-create-an-about-file -> 2-create-an-about-file
```

and indeed the push works!

5.4. A note on echo

```
echo "hello world"
hello world
```

```
echo "hello world" >> hello.md
```

5.5. Review today's class

1. Read today's notes
2. Update your KWL chart with the new items and any learned items.
3. Update the title to any discussion threads you have created to be more descriptive
4. add `branches.md` to your KWL repo and describe how branches work and what things to watch out for in your own words.

5.6. Prepare 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. Read through the Grading Contract README and sample contracts. Start drafting your contract. Bring questions to class on Monday.
2. Bring git questions or scenarios you want to be able to solve to class on Wednesday

> **Warning**
>
> the template contracts in the repo have an error in them; use the ones on the course website.

5.7. More Practice

1. Try creating a merge conflict and resolving it using your favorite IDE.

furtherphysical# How do we study Computer Systems?

6. Admin

- make sure you have a grading contract repo
- you'll track your progress in your

7. Studying Systems

When we think of something as a system, we can study it different ways:

- input/output behavior
- components
- abstraction layers

When we study a system we can do so in two main ways. We can look at the input/output behavior of the system as a whole and we can look at the individual components. For each component, we can look at its behavior or the subcomponents. We can take what we know from all fo the components and piece that together. However, for a complex system, we cannot match individual components up to the high level behavior. This is true in both computers and other complex systems. In the first computers in the 1940s, the only things they did was arithmetic and you could match from their components al the way up pretty easily. Modern computers connect to the internet, send signals, load complex graphics, play sounds and many other things that are harder to decompose all at once. Outside of computers, scientists have a pretty good idea of how neurons work and that appears to be the same across mammals and other species (eg squid) but we do not understand how the whole brain of a mammal works, not even smaller mammals with less complex social lives than humans. Understanding the parts is not always enough to understand all of the complex ways the parts can work together. Computers are much less complicated than brains. They were made by brains.

But that fact motivates another way to study a complex system, across levels of abstraction. You can abstract away details and focus on one representation. This can be tied literally to components, but it can also be conceptual. For example, in CSC211 you use a model of stack and heap for memory. It's useful for understanding programming, but is not exactly what the hardware does. At times, it is even more useful though than understanding exactly what the hardware does. These abstractions also serve a social, practical purpose. In computing, and society at large really, we use *standards* these are sets of guidelines for how to build things. Like when you use a function, you need to know it's API and what it is supposed to do in order to use it. The developers could change how it does that without impacting your program, as long as the API is not changed and the high level input/output behavior stays the same.

Let's take those three pieces, behavior, components, and abstraction in turn.

7.1. Behavior

This is probably how you first learned to use a computer. Maybe a parent showed you how to do a few things, but then you probably tried other things. For most of you, this may have been when you were very young and much less afraid of breaking things. Over time you learned how to do things and what behaviors to expect. You also learned categories of things. Like once you learned one social media app and that others were also *social media* you then looked for similar features. Maybe you learned one video game had the option to save and expected it in the next one.

Video games and social media are *classes* or *categories* of software and each game and app are *instances*. Similarly, an Integrated Development Environment (IDE) is a category of software and VS Code, … are instances. Also, version control is a category of software and git is an instance. A git host is also a category and GitHub is an instance. Just as before you were worried about details you transferred features from one instance to another within categories, I want you to think about what you know from one IDE and how that would help you learn another. We will study the actual features of IDE a=and what you might want to know about them so that you can choose your own. Becoming a more independent developer you'll start to have your own opinions about which one is better. Think about about a person in your life who finds computers and technology overall intimidating or frustrating. They likely only use one social media app if at all, or maybe they only know to make documents in Microsoft word and they think that Google Docs is too much to learn, because they didn't transfer ideas from one to the other.

We have focused on the behavior of individual applications to this point, but there is also the overall behavior of the system in broad terms, typing on the keyboard we expect the characters to show (and when they don't for example in a shell password, we're surprised and concerned it is not working).

7.2. Components

We have the high level parts: keyboard, mouse, monitor/screen, tower/computer. Inside we also tend to know there is a power supply, a motherboard, graphics card, memory, etc.

We can study how each of these parts works while not worrying about the others but having them there. This is probably how you learned to use a mouse. You focused your attention on the mouse and saw what else happened.

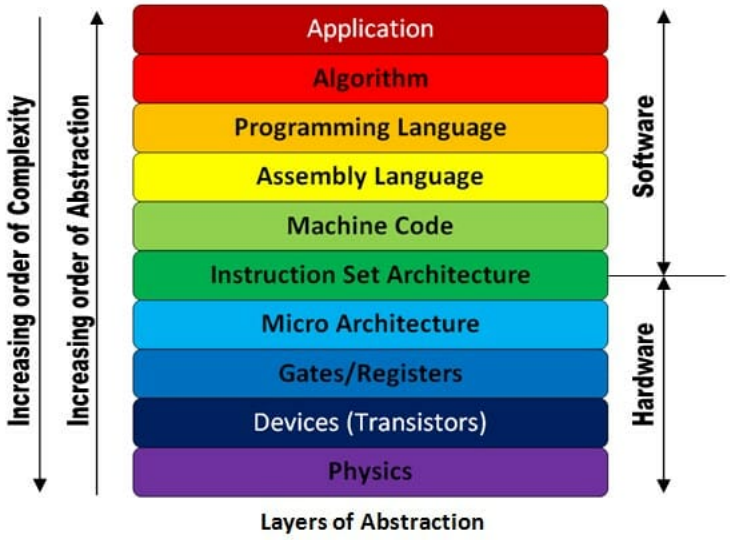
Or we can take an individual component and isolate it to study it alone. For a mouse this would be hard. Without a computer attached its output is not very visible. To do this, we would need additional tools to interpret its output and examine it. Most computer components actually would need additional tools, to measure the electrical signals, but we could examine what happens at each part one at a time to then build up what they do.

This idea, however that we can use another tool to understand each component is an important one. This is also a way to again, take care and study each piece even within a software-alone system without worrying about the hardware.

7.3. Abstraction

As we talked about the behavior and abstraction, we talked some about software and some about hardware. These are the two coarsest ways we can think about a computer system at different levels of abstraction. We can think about it only in physical terms and examine the patterns of electricity flow or we can think about only the software and not worry about the hardware, at a higher level of abstraction.

However, two levels is not really enough to understand how computer systems are designed.



This diagram is mostly what computer scientists and engineers use to describe a computer system.

Application - the software you run.

Algorithm - the way that it is implemented, in mathematical level

Programming language - the way that it is implemented for a computer.

Let's take a simple example, let's say we are talking about a simple search program that we wrote that finds xx. We can say that you put in a part of a file name and it shows you all the ones with a similar name. That description is at the application level it gives the high level behavior, but not the step by step of how it does it. Let's say we implemented it using bubble search then searches by … That's the algorithm level, this is still abstract and could be implemented in different ways, but we know the steps and we can use this to know some things about how fast it will be, what types of result swill make it slower, etc. At the programming level language then we know which language it was done in and we see more details. At this level, we can see the specific data structures and controls structures. These implementation details can also impact performance in terms of space(memory) or time. Still at this level, we do not need to know how the actual hardware works, but we see it in increasing detail. At each level we have different types of operations. At the application we might have input, press enter, get results. At the algorithm we have check the value, compare. At the programming language level we need more specifics too, like assign or append.

After the Programming language level, there is assembly. 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. It is much lower level. When you compile a program, it is translated to assembly. At this level, programs written in different level become indistinguishable. This has much lower level operations. We can do various calculations, but not things like compare. Things that were one step before, like assign become two, choose a memory location, then write to memory. This level of abstraction is the level of detail we will think about most. We'll look at the others, but spend much less time below here.

Machine code translates to binary from assembly.

The instruction set architecture is, notice, where the line between software and hardware lives. This is because these are specific to the actual hardware, this is the level where there are different instructions if you have an Intel chip vs an Apple chip. This level reduces down the instructions even more specifically to the specifics things that an individual piece of hardware does and how they dit.

The microarchitecture is the specific circuits: networks of smaller individual components. Again, we can treat the components as blocks and focus on how they work together. At this level we still have calculations like add, multiply, compare, negate, and we can store values and read them. That is all we have at this point though. At this level there are all binary numbers.

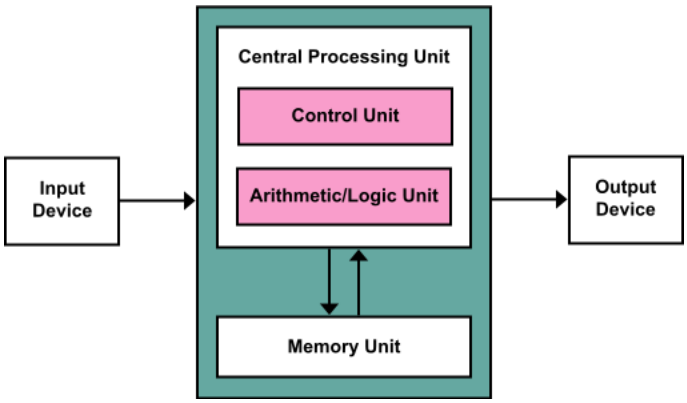
The actual gates (components that implement logical operations) and registers (components that hold values) break everything down to logical operations. Instead of adding, we have a series of `and`, `nand`, `or`, and `xor` put together over individual bits. Instead of numbers, we have `registers` that store individual zeros or ones. In a modern digital, electrical computer, at this level we have to actually watch the flow of electricity through the circuit and worry about things like the number of gates and whether or not the calculations finish at the same time or having other parts wait so that they are all working together. We will see later that when we try to allow multiple cores to work independently, we have to handle these timing issues at the higher levels as well. However, a register and gate can be implemented in different ways at the device level.

> **Note**
>
> We actually only need NAND, we will see how later.

The device (or transistor in modern electrical digital computers) level, is where things transition between analog and digital. The world we live in is actually all analog. We just pay attention to lots of things at a time scale at which they appear to be be digital. Over time devices have changed from mechanical switches to electronic transistors. Material science innovations at the physics level have improved the transistors further over time, allowing them to be smaller and more heat efficient. Because of abstraction, these changes could be plugged into new hardware without having to make any changes at any software levels. However, they do *enable* improvements at the higher levels.

> **Note**
>
> For example, Bayesian statistics is a philosophy that treats probability as subjective uncertainty instead of as frequency. This has some interpretative differences, but most importantly it means that we always need an extra factor (multiplied term) in our calculations. This makes all of the math **much** more complex. For many decades Bayesian statistics was not practical for anything but the simplest models. However, with improvements to computers, that opened new options at the algorithm level. The first large scale application of this type of statistics was by Microsoft after their researchers built a Bayesian player model for player matching in Halo.

We are going to focus today on the Assembly level and use an *abstraction* of computers called the von Neumann model for today.



8. 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.

9. How does the computer actually add 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

The simulator has a few key parts:

- address register
- program counter

If you prefer to read, see section 5.2.1- 5.2.6 of nan2tetris book

- This program the first instruction puts 2 in the address register from the instructions in ROM.
- Then it moves the 2 through the ALU to the data register (D)
- then it puts 3 on the address register
- then it adds the numbers at D and A
- then it puts 0 on the address register
- then it write the output from the ALU (D) to memory (at the location indicated by the A register)

> **Try if yourself**
>
> - What line do you change to change where it outputs the data?
> - How could you add a third number?
> - How could you add two pairs, saving the intermediate numbers?
> - How could you do (4+4)*(3+2)?

10. Hardware Overview

> **Important**
>
> Today is the first day outside of the grade free zone.

11. Review today's class

(required for a C or better)

1. Review and update your listing of how data moves through a program in your `abstraction.md`. Answer reflection questions.

2. practice with the hardware simulator, try to understand its assembly language enough to modify it and walk through what other steps happen.
3. Update your KWL chart with the new items and any learned items

12. Prepare for Next Class

(required for a C or better)

1. Bring questions about git to class on Wednesday.
2. Your grading contract proposal is due Thursday at 3pm.
3. Make sure that the `gh` CLI tool works by using it to create an issue called test on your kwl repo with `gh issue create`

> **⚠ Warning**
>
> the template contracts in the repo have an error in them; use the ones on the course website.

13. More Practice

(mostly required for a B or better)

1. add a hardware term to your group repo. Remember to check other issues and then post an issue and self-assign it before you start working so that two people do not make the same one. Review one other PR.
2. Expand on your update to `abstraction.md`: Can you reconcile different ways you have seen memory before?
3. Try understanding the max.hack and rect.hack. Make notes and answer the questions below in `assemblyexplore.md`.

```
1. Explain how max.hack works in detail.
1. Write code in a high level language that would compile into this program. Try
writing multiple different versions.
1. What does this max.hack assume has happened that it doesn't include in its body.
1. What does rect.hack do?
1. What did you learn trying to figure out how it works?
```

14. Questions After Class

14.1. Content

14.1.1. More about the hardware architecture abstractions

We will come back to this

14.1.2. how the CPU performs addition and stores the result in a memory location on the RAM

We will study the components of a CPU including the ALU and the adder circuit in greater detail later. Today was a preview to introduce these terms so that we can use them as we go forward with learning more tools that will help us study the implications of hardware design choices better.

14.1.3. Explain more levels of abstraction and how they are connected

I expanded some above and we will come back to these throughout the whole course.

14.2. Are there other simulators that we can play with?

- Simple Web-based CPU Simulator w/ Built in Functions

- Web-Based CPU Simulator also w/ Built in Functions

These are both CPU simulators with built in functions. For the first simulator, prewritten functions can be accessed by changing the drop down selection next to “Load Program” under the prewritten code. The functions for the second one can be chosen by changing the drop down menu labeled “SELECT”.

14.3. Logistics

14.3.1. When requesting a review for a PR or something similar, would the only thing to do be tagging (@mention) the instructional team?

Yes, tag @inrocompsys/instructors

14.3.2. With our collaborative repo, would we facilitate the merges and reviews or would you need to have the last review of the information?

You all can merge on your own, but if you have questions, tag us.

14.3.3. Should we assume that when you ask to write a document, we should create a markdown file with a similar name (as opposed to an assignment with the description “add branches.md to your KWL repo”)

Typically with a similar name is correct and never with any spaces in the file name. Mostly the instructions will say exactly what to add. If you have questions, you can (and should) always open an issue to ask for help.

14.3.4. for my grading contract if i choose to go for an A and stick with the A but don’t meet the requirements how much is my grade affected

I will ask you to change your contract or take an incomplete. If you do not respond, I will put the lowest grade you completed.

KWL Chart

Working with your KWL Repo

> **Important**
>
> The `main` branch should only contain material that has been reviewed and approved by the instructors.

1. Work on a specific branch for each activity you work on
2. when it is ready for review, create a PR from the item-specifc branch to `main`.
3. when it is approved, merge into main.

> **Tip**
>
> You could apply branch protections on your feedback branch if you like

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 | _ | _ | _ |
```

Review

Review After Class

After each class, you will need to review the day's material. This includes reviewing prismia chat to see any questions you got wrong and reading the notes. Some days there will be specific additional activities and questions to answer. These should be in your KWL repo.

```
import os
from IPython.display import Markdown, display

rev_file_list = sorted(os.listdir('../_review/'))
```

```
for rev_file in rev_file_list:
    date_str = rev_file[:-3]
    date_link = '[' + date_str + '](../notes/' + date_str + ')'
    display(Markdown(date_link))
    display(Markdown('../_review/' + rev_file))
```

2022-09-07

1. More practice with GitHub terminology
2. Review the notes after I post them

2022-09-12

1. review notes after they are posted, both rendered and the raw markdown
2. 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. (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later
3. fill in hte first two columns of your KWL chart

2022-09-14

1. Follow along with the classmate issue in your inclass repo from today. Work with someone you know or find a collaborator in the Discussion board
2. read the notes
3. try using git in your IDE of choice, Share a tip in the course discussion repo

2022-09-19

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. Make a PR that adds a glossary entry to your team repo to define a term we have learned so far. Create an issue and tag yourself to "claim" a term and check the issues and PRs that are open before your.
4. Review past classes activities (eg via the activities section on the left) and catchup if appropriate

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

```
### Terminal File moving reflection
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
```

2022-09-21

1. Read today's notes
2. Update your KWL chart with the new items and any learned items.
3. Update the title to any discussion threads you have created to be more descriptive
4. add `branches.md` to your KWL repo and describe how branches work and what things to watch out for in your own words.

2022-09-26

1. Review and update your listing of how data moves through a program in your `abstraction.md`. Answer reflection questions.
2. practice with the hardware simulator, try to understand its assembly language enough to modify it and walk through what other steps happen.
3. Update your KWL chart with the new items and any learned items

Collaboration

You will be in a “team” that is your built in collaboration group to practice using Git Collaboratively. There will be assignments that are to be completed in that repo as well. These activities will be marked accordingly. You will take turns and each of you is required to do the initialization step on a recurring basis.

This is also where you can ask questions and draft definitions to things.

Peer Review

If there are minor errors/typos, suggest corrections inline.

In your summary comments answer the following:

- Is the contribution clear and concise? Identify any aspect of the writing that tripped you up as a reader.
- Are the statements in the contribution verifiable (either testable or cited source)? If so, how do you know they are correct?
- Does the contribution offer complete information? That is, does it rely on specific outside knowledge or could another CS student not taking our class understand it?
- Identify one strength in the contribution, and identify one aspect that could be strengthened further.

Choose an action:

- If the suggestions necessary before merging, select **request changes**.
- If it is good enough to merge, mark it **approved** and open a new issue for the broader suggestions.
- If you are unsure, post as a **comment** and invite other group members to join the discussion.

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))
```

2022-09-07

- Read the syllabus, explore website
- Bring questions about the course
- Think about one thing you've learned really well (computing or not) and how do you know that you know it? (bring your example)
- Post an introduction to your classmates on our discussion forum

2022-09-12

1. 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. Add them to our course Discussions in General. Start a new thread and/or comment on others if theirs are related to yours.
2. Make sure you have a working environment for next week. Use slack to ask for help.
    - install GitBash on windows (optional for others)
    - make sure you have Xcode on MacOS
    - install the GitHub CLI on all OSs

2022-09-14

1. Make a list of 3-5 terms you have learned so far and try defining them in your own words.
2. using your terminal, download your KWL repo and update your 'learned' column on a new branch **do not merge this until instructed**
3. answer the questions below in a new markdown file in your KWL

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?
```

2022-09-19

1. 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)
2. install h/w simulator
3. map out how you think about data moving through a small program and bring it with you to class (no need to submit)

```
## 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?
```

2022-09-21

1. Read through the Grading Contract README and sample contracts. Start drafting your contract. Bring questions to class on Monday.
2. Bring git questions or scenarios you want to be able to solve to class on Wednesday

2022-09-26

1. Bring questions about git to class on Wednesday.
2. Your grading contract proposal is due Thursday at 3pm.
3. Make sure that the `gh` CLI tool works by using it to create an issue called test on your kwl repo with `gh issue create`

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))
```

2022-09-12

1. read Chapter 1, "Decoding your confusion while coding" in The Programmer's Brain add a file called brain.md to your kwl repo that summarizes your thoughts on the chapter and how, if at all, it changes how you think about debugging and learning to program.

2022-09-14

1. Download the course website and your group repo via terminal. Try these on different days to get "sapced repetition" and help remember better.
2. 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

2022-09-19

1. Once your PRs in your kwl are merged, add a Table of Contents to the README with relative links to each file
2. Add cheatsheet entry to your team repo to do something of interest with git or shell. Make an issue for it first and assign yourself so that your team mates know you are working on that topic.

2022-09-21

1. Try creating a merge conflict and resolving it using your favorite IDE.

2022-09-26

1. add a hardware term to your group repo. Remember to check other issues and then post an issue and self-assign it before you start working so that two people do not make the same one. Review one other PR.
2. Expand on your update to `abstraction.md`: Can you reconcile different ways you have seen memory before?
3. Try understanding the max.hack and rect.hack. Make notes and answer the questions below in `assemblyexplore.md`.

```
1. Explain how max.hack works in detail.
1. Write code in a high level language that would compile into this program. Try
writing multiple different versions.
1. What does this max.hack assume has happened that it doesn't include in its body.
1. What does rect.hack do?
1. What did you learn trying to figure out how it works?
```

Deeper Explorations

> **Warning**
>
> deeper explorations are not required, but an option for higher grades

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 completed fewer items.

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

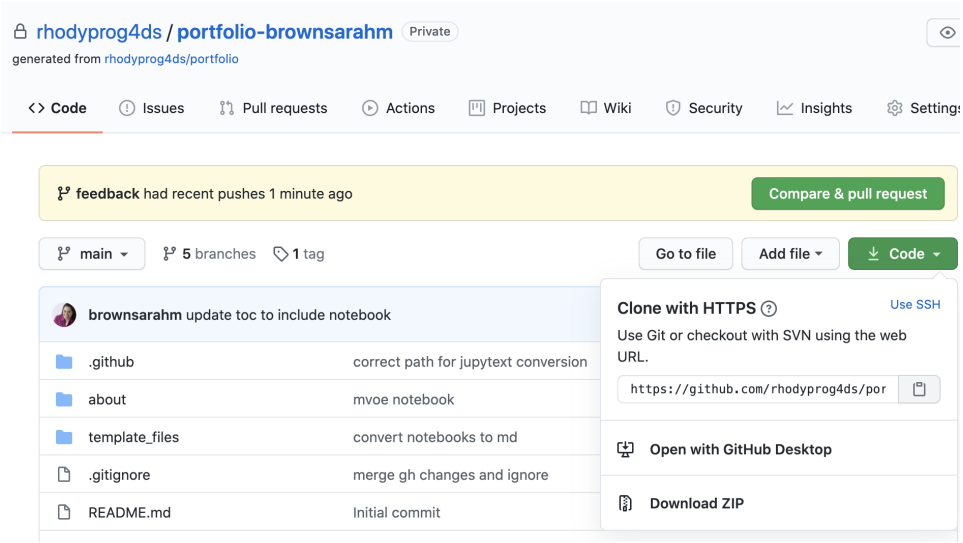
GitHub has strong rules about authentication You need to use SSH with a public/private key; HTTPS with a Personal Access Token or use the GitHub CLI auth

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
```

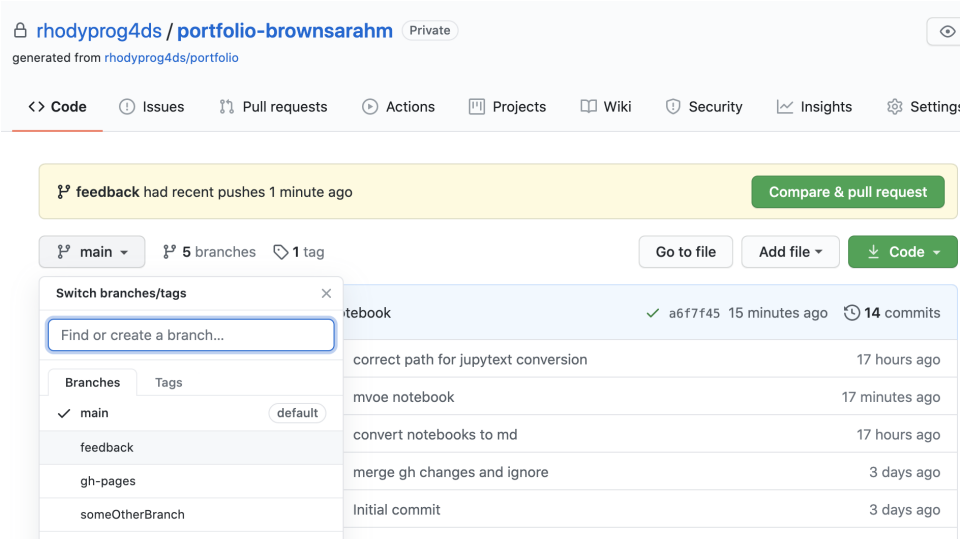
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
git clone https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
```

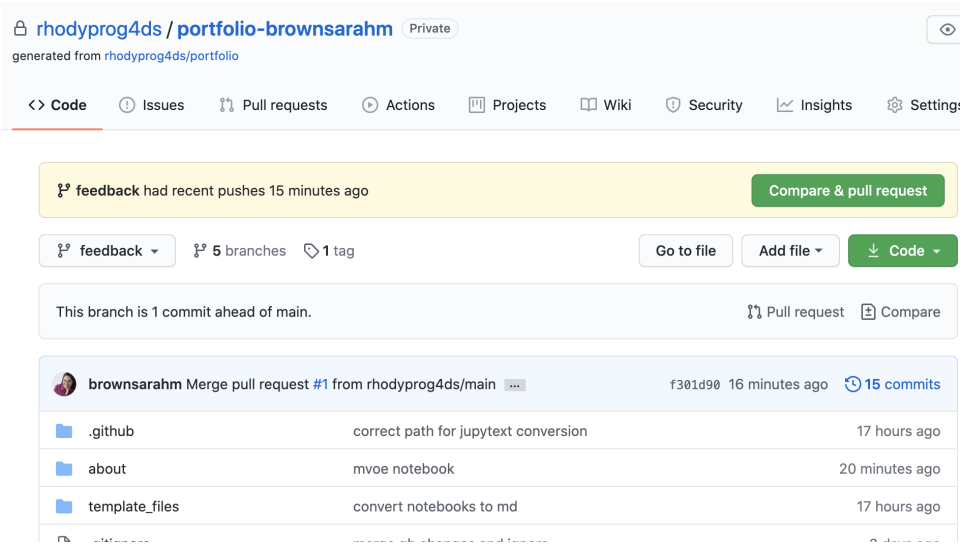
When you merged the Feedback pull request you advanced the `feedback` branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it's up to date with the `remote` (the copy on GitHub)

```
cd portfolio-brownsarahm
git checkout feedback
git pull
```

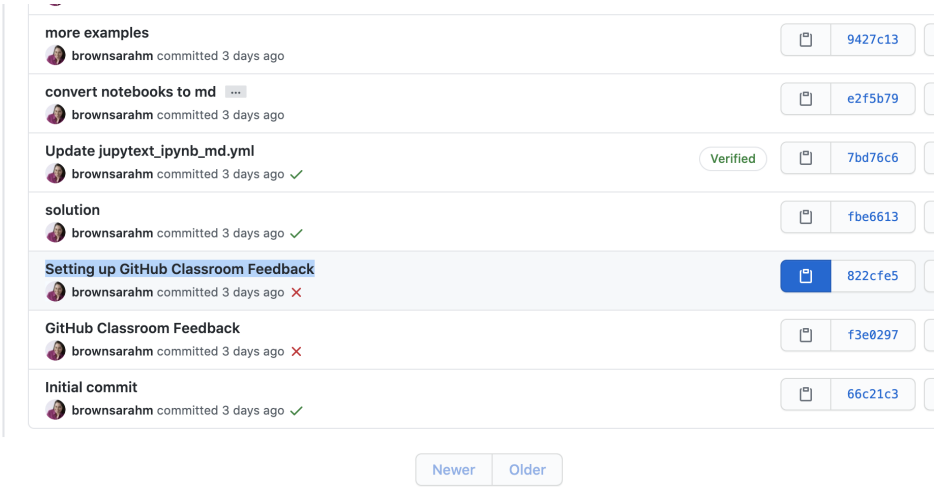
Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and swithc to the feedback branch there. Click on where it says `main` on the top right next to the branch icon and choose feedback from the list.



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.

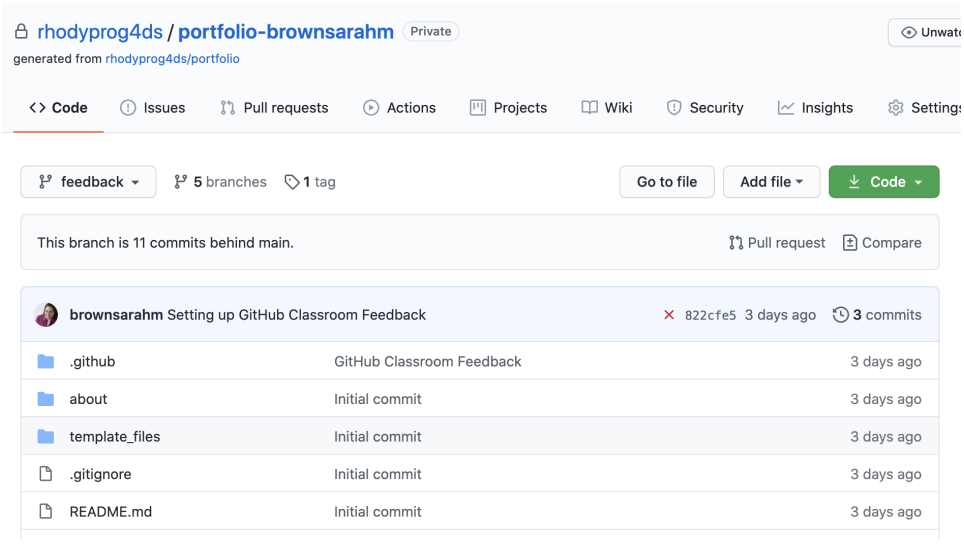
```
git push origin -f
```

and you'll get something like this to know that it worked

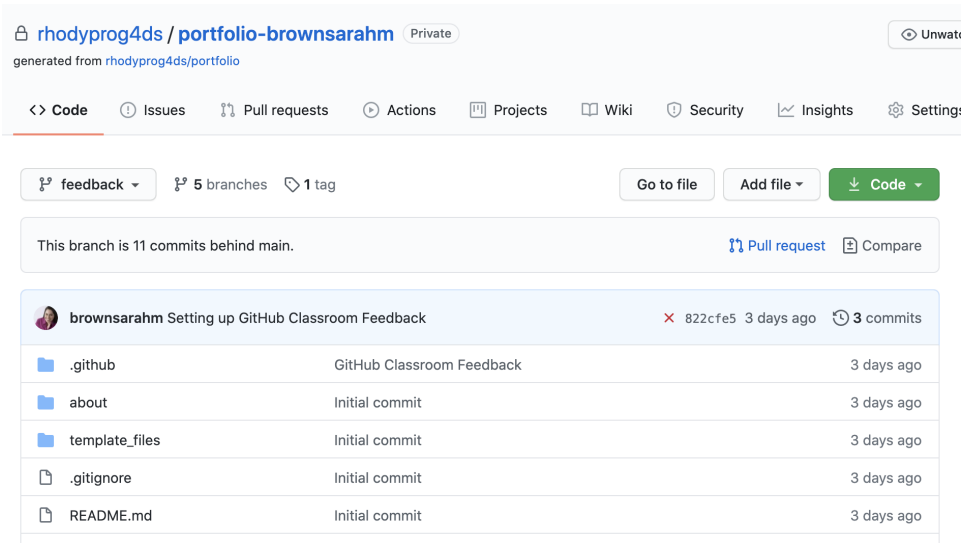
```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

Again, the numbers will be different and it will be your url, not mine.

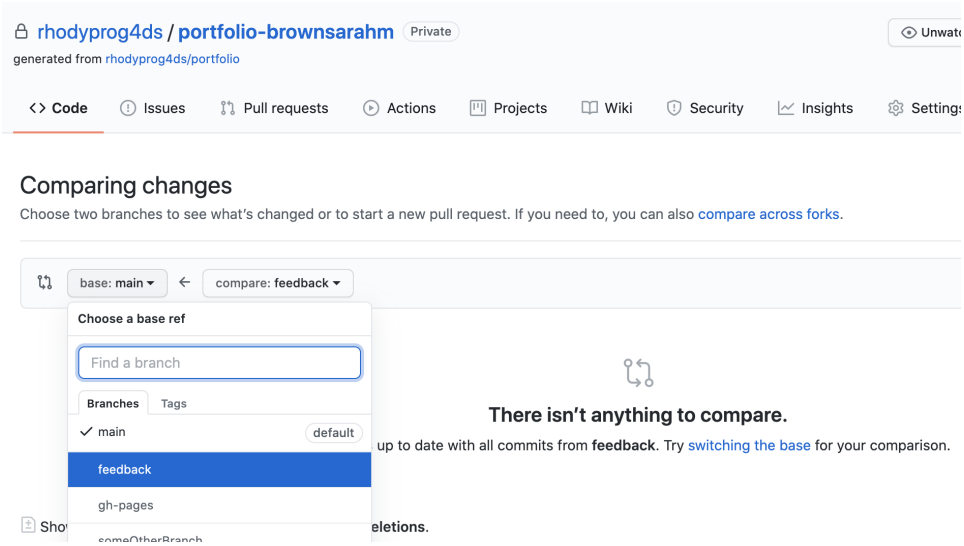
Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, “This branch is 11 commits behind main” your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the 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”).



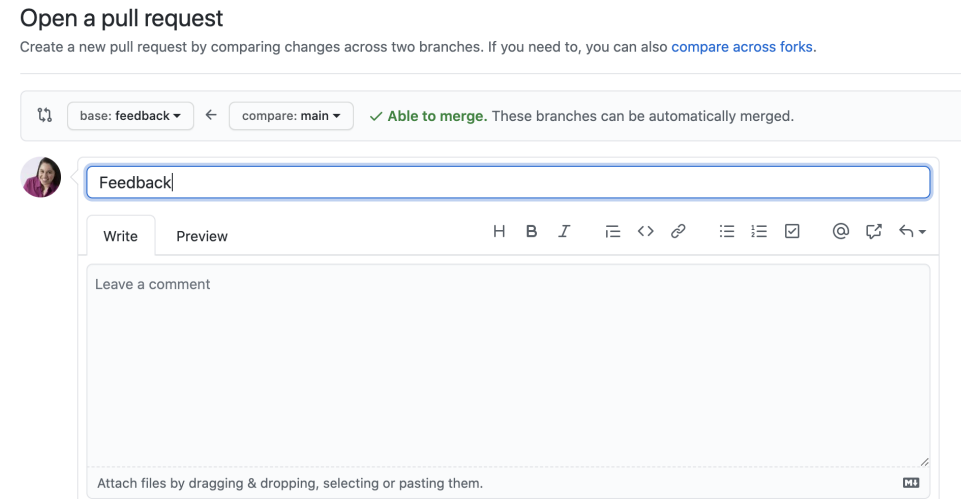
Now, you need to recreate your Pull Request, click where it says pull request.



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.
```

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.

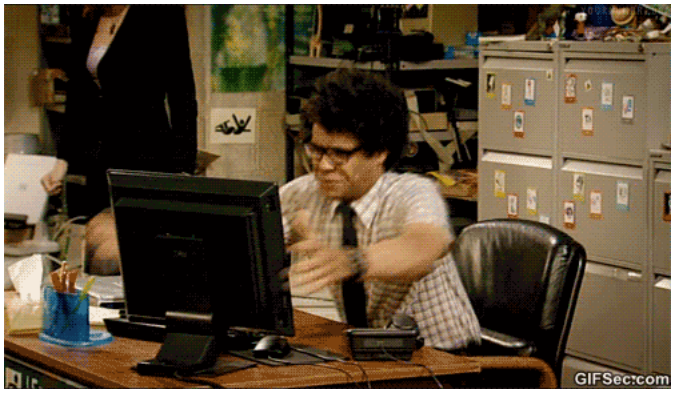
on email

- 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?

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.

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 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

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.

> **Note**
>
> A fun version of this is rubber duck debugging

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:

```
CSC310
  |- notes
  |- portfolio-username
  |- 02-accessing-data-username
  |- ...
```

This is one top level folder will all materials in it. A folder inside that for in class notes, and one folder per repository.

Please **do not** include all of your notes or your other assignments all inside your portflio, it will make it harder to grade.

Finding repositories on github

Each assignment repository will be created on GitHub with the `rhodyprog4ds` 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.

:header-rows: 1

- Resource
  - Level
  - Type
  - Summary
- What is a CPU, and What Does It Do?
  - 1
  - Article
  - Easy to read article that explains CPUs and their use. Also touches on "buses" and GPUs.
- Processors Explained for Beginners
  - 1
  - Video
  - Video that explains what CPUs are and how they work and are assembled.
- The Central Processing Unit
  - 1
  - Video
  - Video by Crash Course that explains what the Central Processing Unit (CPU) is and how it works.