About this Site

Contents

Syllabus

- Basic Facts
- Tools and Resources
- Grading
- Grading Contract Reference
- Schedule
- Grading Policies
- Support
- General URI Policies
- Office Hours & Comms

Notes

- 1. Welcome and Introduction
- 2. Course Logistics and Learning
- 3. How can I use GitHub offline?
- 4. Why Do I Need to Use a terminal?
- 5. How should I use git to stay organized this class?
- 6. How do we study Computer Systems?
- 7. What *is* git?
- 8. How can I fix things in git?
- 9. How does git *really* work?
- 10. What are git hashes and why are they alaphanumeric?
- 11. How do git references work?
- 12. Bash Scripts
- 13. How can I automate things on GitHub

Activities

- KWL Chart
- Team Repo
- Review
- Prepare for the next class
- More Practice
- KWL File Information
- Deeper Explorations
- Project Information

FAQ

- Syllabus and Grading FAQ
- Git and GitHub

Resources

- General Tips and Resources
- How to Study in this class

- Getting Help with Programming
- Getting Organized for Class
- More info on cpus
- Advice from Spring 2022 Students

Welcome to the course manual for Introduction to Computer Systems in Fall 2022 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.

course calendar

> **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.

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.

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

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 repository You can view the date of changes and exactly what changes were made on the Github commit history page.

Creating an issue is also a good way to ask questions about anything in the course it will prompt additions and expand the FAQ section.

About your instructor

Name: Dr. Sarah M Brown Office hours: listed on communication page

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.

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

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).
- 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

> **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.

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).

 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.
```

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.
```

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?

approximately 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 knw 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 sotware 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; cofirm 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 script	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_1902/2146052215.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.

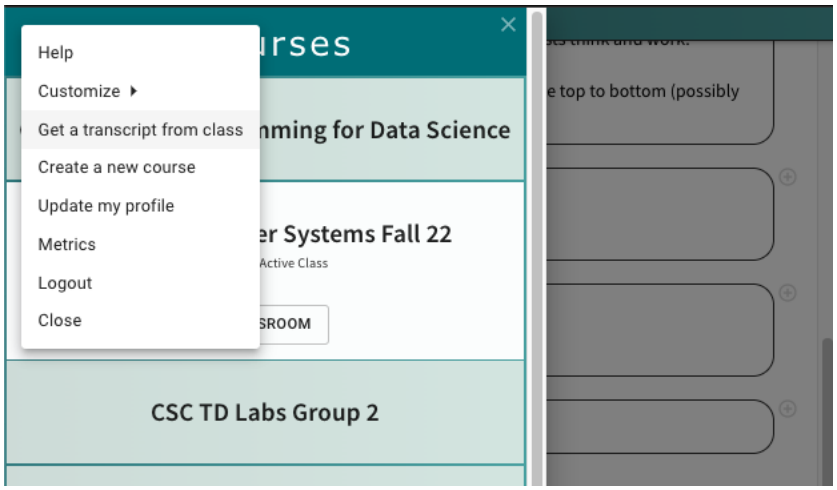
1.10.3.3. what kind of coding will we be doing?

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.

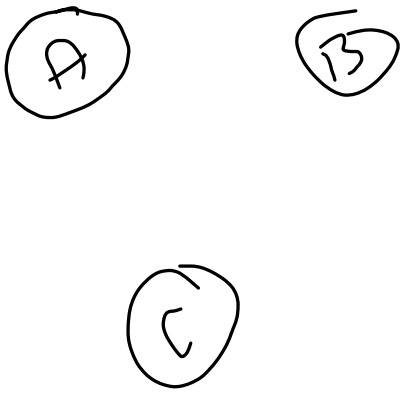


Fig. 2.1 a novice mental model is disconnected and has few concepts

As we learn more, they become more connected.

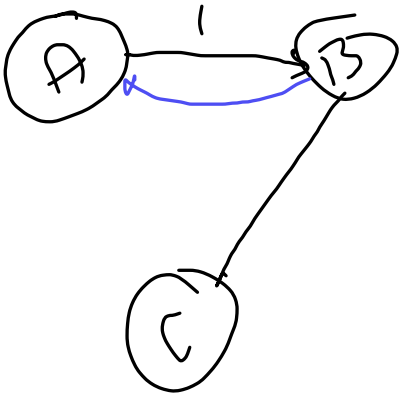


Fig. 2.2 a compententmental model starts to have some connections, with relationships between the concepts.

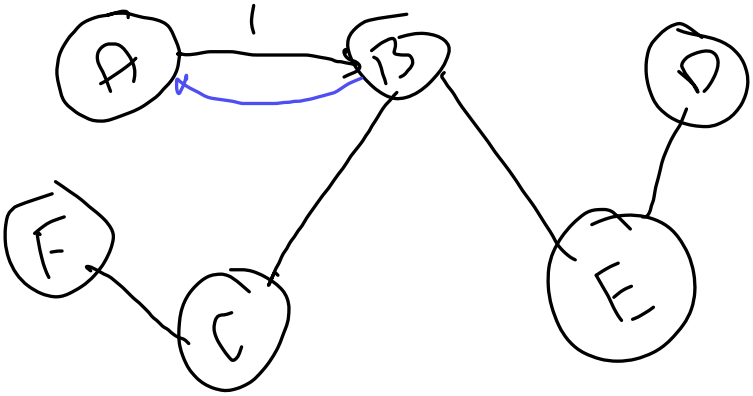


Fig. 2.3 an expert mentla model is densley connected and has more concepts in it.

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

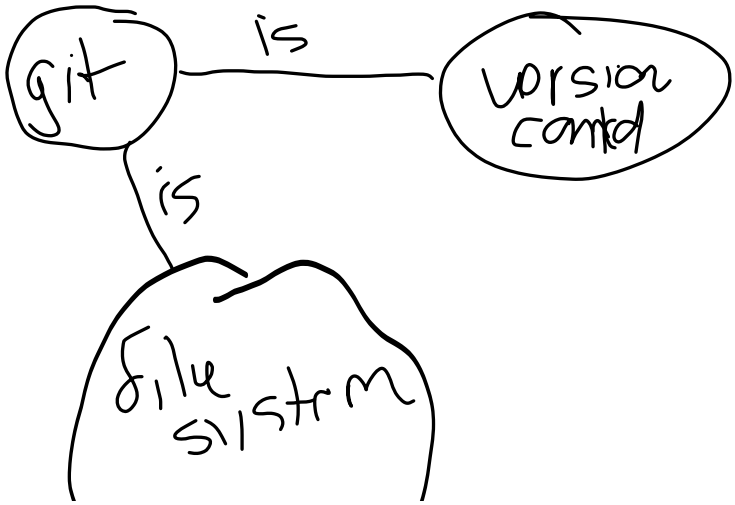


Fig. 2.4 a small concept map showing that git is an instance of both a file system and a version control system.

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.

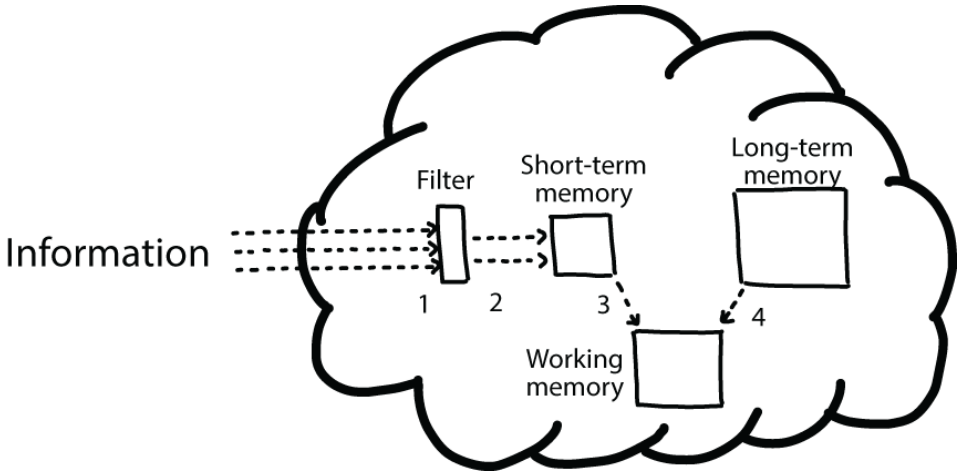


Fig. 2.5 An overview of the three cognitive processes that this book covers: STM, LTM, and working memory. The arrows labeled 1 represent information coming into your brain. The arrows labeled 2 indicate the information that proceeds into your STM. Arrow 3 represents information traveling from the STM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

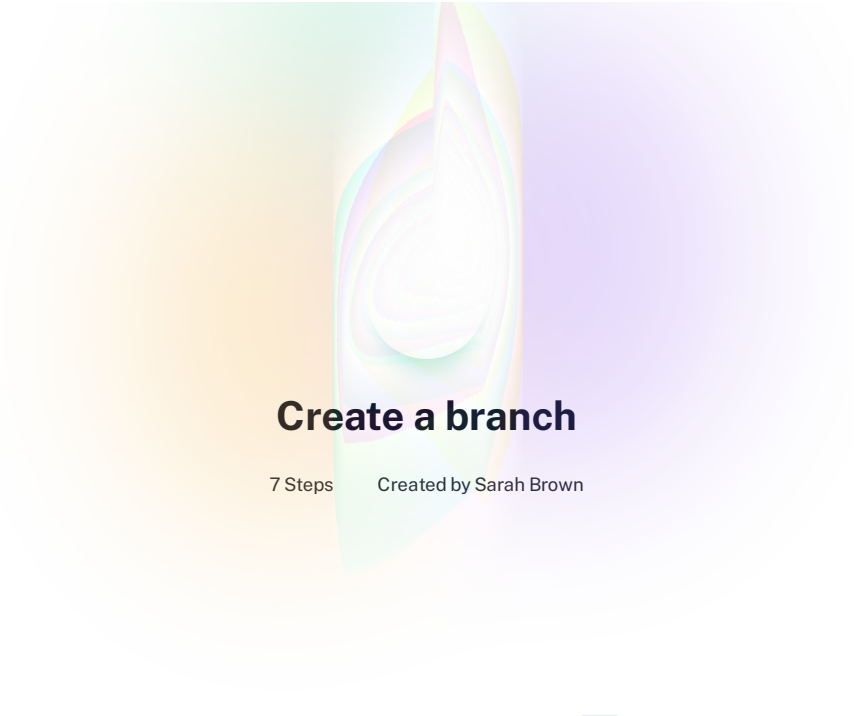
and then we can follow what git said to do

```
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
```

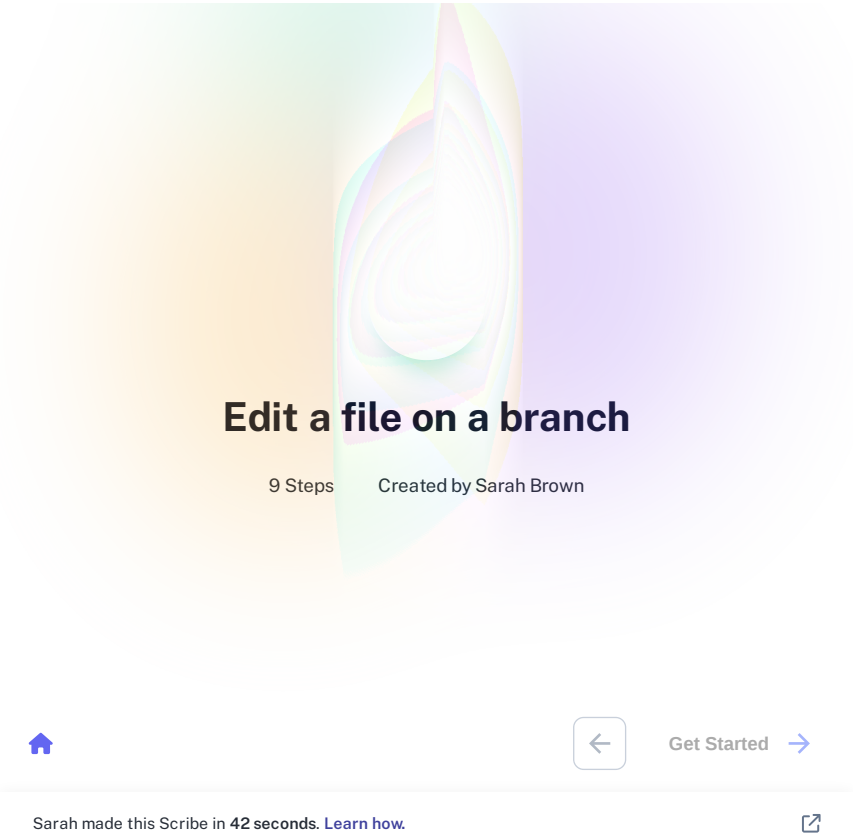
and add and commit the changes

```
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.

6. How do we study Computer Systems?

6.1. Admin

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

6.2. 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.

6.2.1. 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).

6.2.2. 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.

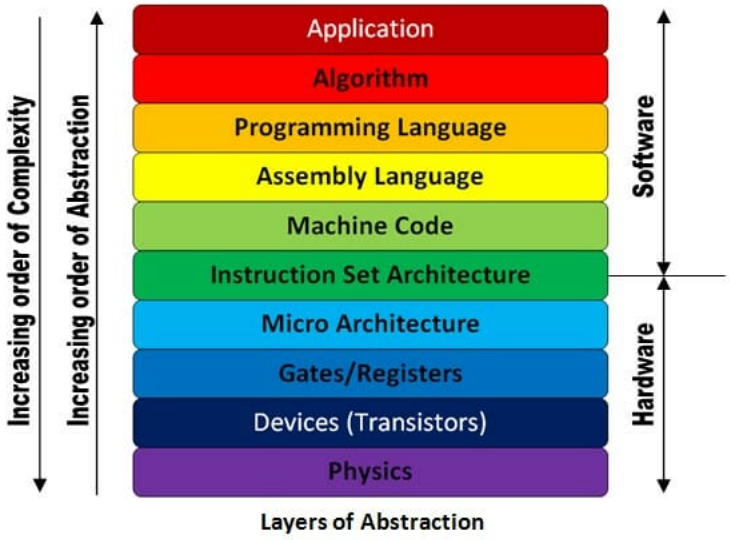
Or we can take an individual component and isolate it to study it alone. For a mouse this would be hard. Without a computer attached its output is not very visible. To do this, we would need additional tools to interpret its output and examine it. Most computer components actually would need additional 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.

6.2.3. 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.

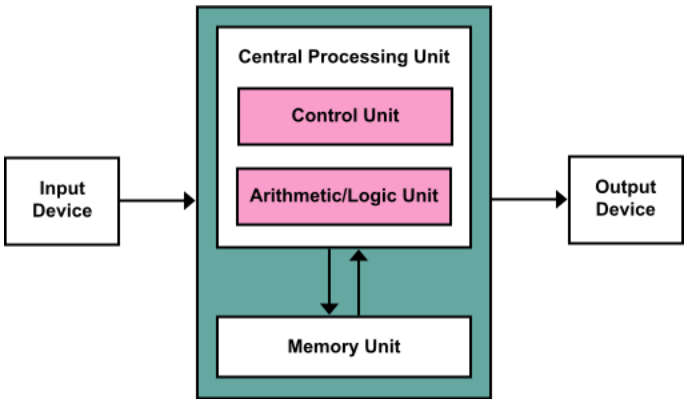
The actual gates (components that implement logical operations) and registers (components that hold values) break everything down to logical operations. Instead of adding, we have a series of `and`, `nand`, `or`, and `xor` put together over individual bits. Instead of numbers, we have `registers` that 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.



6.3. Using the simulator

On MacOS:

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

On Windows: Double click on the CPUEmulator.bat file

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

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

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

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

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

6.4. 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)?

6.5. Hardware Overview

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

6.6. 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

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

6.7. 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.

6.8. 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?

1. 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?
```

6.9. Questions After Class

6.9.1. Content

6.9.1.1. More about the hardware architecture abstractions

We will come back to this

6.9.1.2. 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.

6.9.1.3. 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.

6.9.2. 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”.

6.9.3. Logistics

6.9.3.1. 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

6.9.3.2. 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.

6.9.3.3. 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.

6.9.3.4. 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.

7. What *is* git?

7.1. Admin

get credit for this class toward your major

- substitution of: CSC392 Intro to Computer Systems For: 300 level elective
- no rationale
- send the form to Professor Dipippo

Participate in research:see Prismia for a survey link

7.2. Git is a File system

> **Important**
>
> git book is the official reference on git.
>
> this includes other spoken languages as well if that is helpful for you.

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

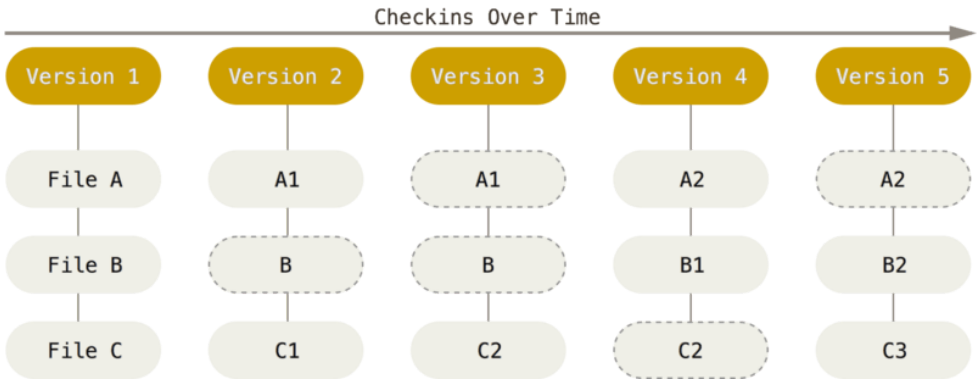
Content-addressable filesystem means a key-value data store.

what types of programming have you seen that use key- value pairs?

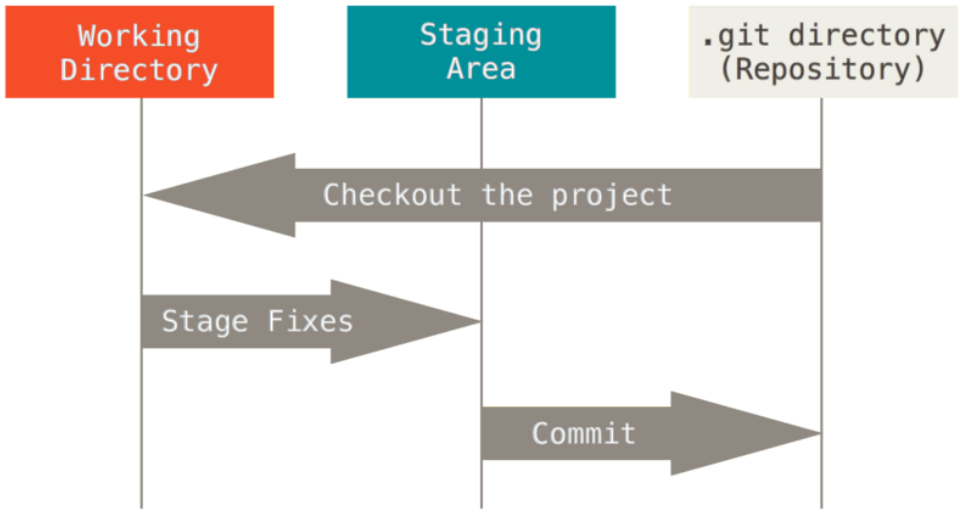
What this means is that you can insert any kind of content into a Git repository, for which Git will hand you back a unique key you can use later to retrieve that content.

7.3. Git is a Version Control System

git stores **snapshots** of your work each time you commit.



it uses 3 stages:



7.4. Git has two sets of commands

Porcelain: the user friendly VCS

Plumbing: the internal workings- a toolkit for a VCS

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

The plumbing commands reveal the *way* that git performs version control operations. This means, they implement file system operations for the git version control system.

You can think of the plumbing vs porcelain commands like public/private methods. As a user, you only need the public methods (porcelain commands) but those use the private ones to get things done (plumbing commands). We will use the plumbing commands over the next few classes to examine what git *really* does when we call the porcelain commands that we will typically use.

7.5. Git is distributed

What does that mean?

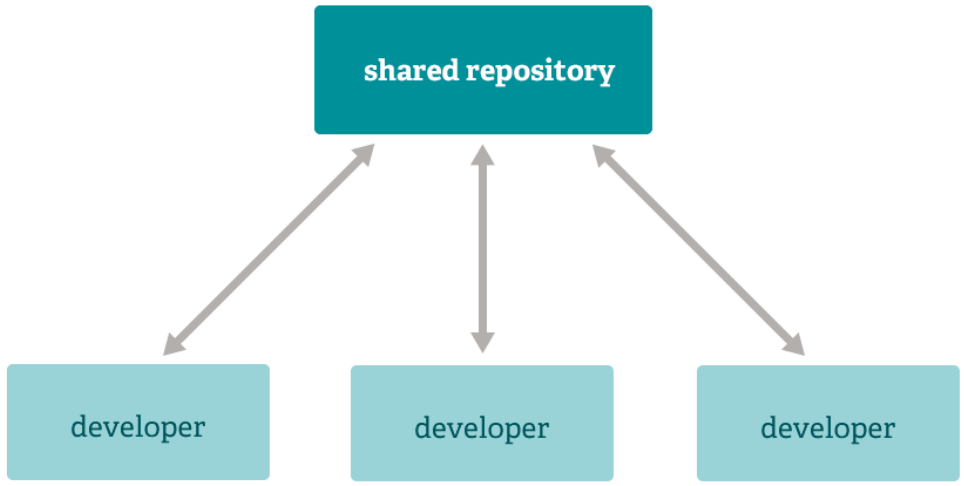
Git runs locally. It can run in many places, and has commands to help sync across remotes, but git does not require one copy of the repository to be the “official” copy and the others to be subordinate. git just sees repositories. For human reasons, we like to have one “official” copy and treat the others as local copies, but that is a social choice, not a technological requirement of git. Even though we will

typically use it with an offical copy and other copies, having a tool tht does not care, makes the tool more flexible and allows us to create workflows, or networks of copies that have any relationship we want.

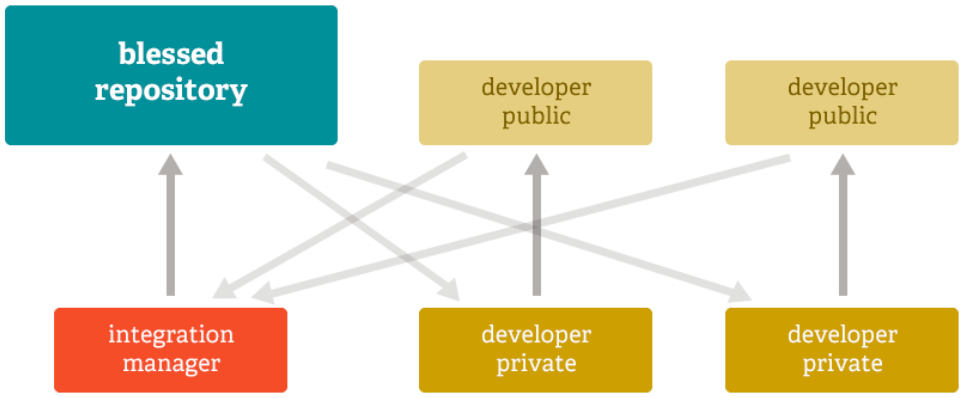
It's about the workflows, or the ways we socially *use* the tool.

Some example workflows

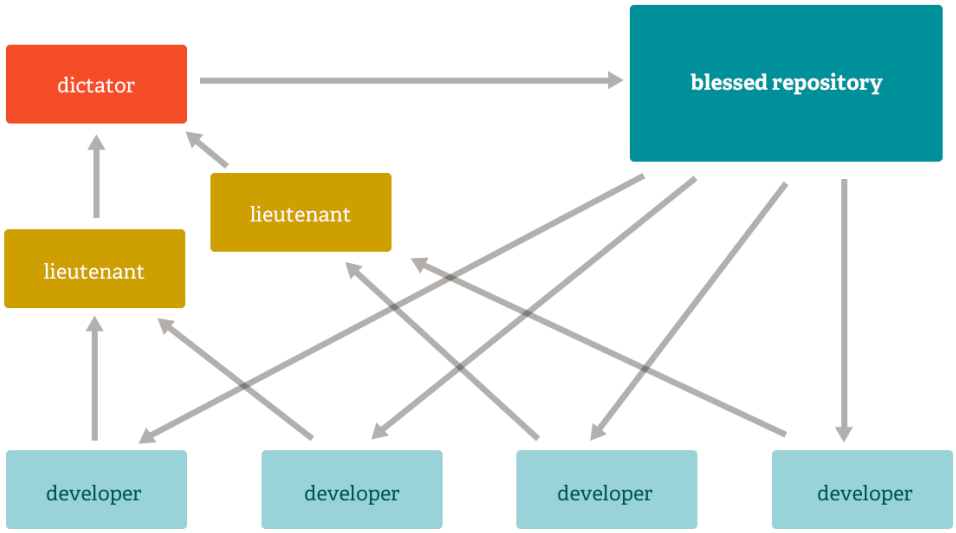
7.5.1. Subversion WOrkflow



7.5.2. Integration Manager



7.5.3. dictator and lieutenants



7.6. Anatomy of Git

Note

Contributing to the course site can count for contributions to your team repo. To contribute here, you will have to make a fork. Adding more explanation to these images is a great contribution

Let's look at git again in our `github-inclass` repo

As always, start by checking where we are:

```
git status
```

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

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

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

I was not on the main branch, so I'll switch back there

```
git checkout main
```

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

Remember, we can see hidden files with the `-a` option.

```
ls -a
```

```
.               .git            README.md       hello.md
..              .github         about.md
```

of the structure of git is contained in the `.git` directory, that is called the "database" this contains

```
cd .git
ls
```

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

We see a lot of files here. The most important ones are:

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

7.6.1. Git HEAD

```
cat HEAD
```

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

this indeed is the branh we have checked out and we can confirm with `git status`

```
git status
```

```
fatal: this operation must be run in a work tree
```

excpet we cannot call this in the `.git` directory, we have to got back up

```
cd ..
git status
```

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

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

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

and we indeed see that it matches.

We can also confirm that it changes, that it's not just always main by changing branches. We can first list the branches

```
git branch
```

```
  2-create-an-about-file
* main
  organtzation
```

then switch to one of the others

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

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

And again examine HEAD

```
cat .git/HEAD
```

```
ref: refs/heads/2-create-an-about-file
```

it indeed changes.

Next, what is that file that the HEAD file points to?

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

```
3c9980a4883782386ea3112d521b1b95997b0be6
```

this is a commit, the last one that was added to the branch we checked.

To confirm we switch back to main

```
git checkout main
```

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

and check the commit history

```
git log
```

```
commit 3c9980a4883782386ea3112d521b1b95997b0be6 (HEAD -> main, origin/main,
origin/HEAD)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 17:40:01 2022 -0400

    create about

commit ec3dd020d2767e45b6cd61f7c4ea5f6f2be9a3d8
Merge: 1613072 db2e41d
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 16:56:34 2022 -0400

    Merge pull request #4 from introcompsys/create_readme

    Create README.md

commit db2e41d48129b2c3ad09b78f1c14c2cd295e3eb2 (origin/create_readme)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 16:55:23 2022 -0400

    Create README.md

commit 1613072525c141b13d3ac7db68e8c1dbee70496b
Author: github-classroom[bot] <66690702+github-
classroom[bot]@users.noreply.github.com>
Date:   Wed Sep 14 20:51:29 2022 +0000

    Initial commit
```

Warning

when we run this it opens the log interactively, so we see something like what is to the left, but it might have a `:` at the end. Use enter/return to scroll and press `q` to exit.

Try it Yourself

use git log to draw a map that shows where the different branches are relative to one another

Also notice that next to the commits, it lists what branches point to that commit.

We can also see the whole list of branches

```
git branch
```

```
  2-create-an-about-file
* main
  organtzation
```

and compare that to the `refs/head` directory.

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

```
2-create-an-about-file  main                    organtzation
```

which they do match.

there are other types of refs

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

```
heads   remotes tags
```

tags are what I use to create releases on the course website (if you watch the course repo you can be notified when I make major updates that way)

remotes are other copies that are linked, like GitHub.

7.7. Git Config

```
cat config
```

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/introcompsys/github-inclass-brownsarahm.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[branch "organtzation"]
        remote = origin
        merge = refs/heads/organtzation
[branch "2-create-an-about-file"]
        remote = origin
        merge = refs/heads/2-create-an-about-file
```

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

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

7.8. Git Index

```
cd ..
```

```
ls
```

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

```
git status
```

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

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

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

```
git add .
```

```

```

```
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:   hello.md

```

```
git write-tree
```

```
9e404c8b4b67cc0572174531d3b7364e9c88fd94
```

```
git cat-file -p 9e404c8b4b67cc0572174531d3b7364e9c88fd94
```

```
040000 tree 95b60ce8cdec1bc4e1df1416e0c0e6ecbd3e7a8c    .github
100644 blob bcc1d9287b5cae329629cf3e0779b065b72dad7a    README.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    about.md
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad    hello.md
```

```
git cat-file -t 9e404c8b4b67cc0572174531d3b7364e9c88fd94
```

```
tree
```

To better understand this, we can compare it to the last commit.

To view an object we actually only need enough characters to make the name unique. 4 worked for me here:

```
git cat-file -p 3c99
```

```
tree 2c4495ca1dc6868006365dff726cccea781cea61
parent ec3dd020d2767e45b6cd61f7c4ea5f6f2be9a3d8
author Sarah M Brown <brownsarahm@uri.edu> 1663191601 -0400
committer Sarah M Brown <brownsarahm@uri.edu> 1663191601 -0400

create about
```

the object for a commit is the information about the commit:

- the `tree` is the snapshot that was indexed (what we want to compare the above to).
- the parent ithe previous commit
- then there's author & committer information and time
- last is the commit message

So, we can look at this tree object

```
git cat-file -p 2c4495
```

```
040000 tree 95b60ce8cdec1bc4e1df1416e0c0e6ecbd3e7a8c    .github
100644 blob bcc1d9287b5cae329629cf3e0779b065b72dad7a    README.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    about.md
```

we can see that the `hello.md` blob object is the difference and the other two files even have the same hash (so they were not changed).

We say that that part is only the the staging area, beacue this item is added to the objects, but it is not in the commit history yet.

7.9. Git Objects

All of these blob objects and trees are stored in the objects folder

```
cd .git/objects/
ls
```

it has folders with 2 character names

```
25      35      59      84      9d      aa      e6      f9
2c      3c      5d      8b      a0      ad      e7      info
30      40      7a      8c      a4      d2      ec      pack
```

```
cd 25
ls
```

that contain files that match commits and other objects' hashes.

```
ecf2cf579d20d41c1ae2f9844662bbd4e43315
```

For example, we can confirm the last commit to main

```
cd ../3c
ls
```

```
9980a4883782386ea3112d521b1b95997b0be6
```

These objects are not human readable though as is

```
cat 9980a4883782386ea3112d521b1b95997b0be6
```

```
x??A
?:m\x?;???^?<?x?n??2?4??Gp?<x?e[????x?E??0t□?z?%O]?9:???w       ¸??T|J???"E
                        ???"m????d??s#E?*?j>?>J ```
```

7.10. Review today's class

1. Practice with git log and redirects to write the commit history of your main branch for your kwl chart to a file `gitlog.txt` and commit that file to your kwl repo.
2. Review the notes
3. Update your kwl chart with what you have learned or new questions

7.11. Prepare for Next Class

1. Try exploring your a repo manually and bring more questions
2. Make sure that you have a grading contract that has been reviewed at least once

7.12. More Practice

1. ¸ Read about different workflows in git and add responses to the below in a `workflows.md` in your kwl repo. Two good places to read from are Git Book and the atlassian Docs
2. Contribute either a glossary term, cheatsheet item, or additional resource/reference to your group repo.
3. Complete one peer review of a team mate's contribution

```
## Workflow Reflection

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

7.13. Questions

7.13.1. Can you fork a repo you do not have access to?

You cannot fork a repository that you do not have read access to. If it is private, the owner can allow or not allow forking. You can fork public repos and then submit PRs to suggest changes. Try making a contribution to the course site. Add a glossary entry, link a term from the notes to the glossary or fill in a caption.

7.13.2. what should I do if git crashes and I lose progress?

git should not ever have this problem. You should not ever lose work again at all if you commit and push frequently.

I lost my whole computer a few months before I finished my PhD. I only lost 1 day's worth of work though. The morning's worth of work that caused the crash and the afternoon that day recovering the computer (new harddrive and Linux instead of Windows) and then I pulled my code and kept working.

7.13.3. Which is easier/better to use: the git CLI, or the GitHub website?

the git CLI is more complete and more flexible, you can batch changes together into larger commits, for example. It is better for most work. GitHub is simple and less intimidating for novices so for small changes it is great.

7.13.4. What most important function that git can perform?

restoring previous versions.

7.13.5. When encountering a merge conflict, is it best to use the terminal/bash window or the GitHub merging features?

You can use the GitHub one for basic merge conflicts. For complex ones GitHub will give you terminal commands to download the work and work offline.

7.13.6. Why is there not an undo command if you delete a file or repo on git?

There is a way to undo with git, there is not a way to undo `rm` in bash, unless you are using some sort of backup tool, or git.

7.13.7. Would a merge conflict or any kind of problem emerge if there's a file exclusive to a branch that isn't main, when we try to merge that branch's contents into main?

No, there is nothing that there can be a conflict with. That is one strategy to use to avoid conflicts, to have multiple, smaller files.

7.13.8. How often should I make a new branch in my KWL repo?

One for each group of work you want graded. Ideally about one per course session.

7.13.9. how is git different than other coding languages?

Git is a program that manages files, it is not a programming language. It is a program with many subroutines and designed for use on a shell, but still not a language. Languages have to have certain features.

7.13.10. where is git used mostly (outside this classroom/university)?

Literally almost everywhere people write code.

7.13.11. One git scenario, I feel like I will reach a point where there are just too many branches if I'm supposed to make a new branch every time I edit something.

You can delete a branch

7.13.12. whats the difference between git fork and git clone

a fork keeps the copy on GitHub (or another server) a clone makes it local

7.13.13. Can we add arguments to make git more specific with any type of errors that may occur, ex merge conflicts?

Git is very powerful and can be given more strict guidance on how to do a merge that may change if conflicts are produced or not.

7.13.14. How does GitHub differ from bitbucket?

Pricing, being owned by Microsoft instead of Atlassian, and some in browser features. GitHub has a more advanced API and continuous integration lately. It also has classroom features that I use (to let you all create repos that I can manage and Mark can see automatically) that other git hosts do not have.

7.13.15. will we be constantly updating a .md file to the kwl repo for a detailed knowledge base on each of the items listed in the chart?

You only need to add files as advised.

7.13.16. how often should we push information from the kwl table into each .md file?

You should commit and push your work frequently. This allows you to have a back up and restore your work freqently?

7.13.17. If we run into errors using CLI tools what are ways or resources that can help debug and fix errors.

It depends on the error. We'll see some and I encourage you to expand upon this question.

8. How can I fix things in git?

8.1. Moving a Commit

Start on your main branch of your github in class repo with a clean working directory

```
git status
```

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

nothing to commit, working tree clean
```

Now let's create a new file

```
touch new_feature.md
```

and add commit and push it to the main branch

```
git add .
```

```

```

```
git comit -m "new feature"
```

```
git commit -m "new feature"
```

```
[main 0830c39] new feature
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new_feature.md
```

```
git push
```

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 266 bytes | 266.00 KiB/s, done.
Total 2 (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
   3c9980a..0830c39  main -> main
```

We can look in GitHub:

```
gh repo view
```

```
introcompsys/github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom


   github-inclass-brownsarahm

  github-inclass-brownsarahm created by GitHub Classroom



View this repository on GitHub: https://github.com/introcompsys/github-inclass-
brownsarahm
```

This command prints the README with a link to the repo online.

We had wanted to make a new feature but we realize that our work is on the main branch, not a separate feature branch.

Fortunately, we can fix it!

```
git status
```

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

nothing to commit, working tree clean
```

We can reset the the staging area and undo the last commit (remove the commit itself, but not the changes) to the last state that the staging area had with

```
git reset HEAD^
```

we can see what this did with

```
git status
```

```
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

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

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

Our file is still there, it is now not in the git commit history and the changes are back in the staging area.

Now we can switch branches

```
git checkout -b new_feature
```

```
Switched to a new branch 'new_feature'
```

and confirm

```
git status
```

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

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

and then commit to the new branch.

```
git add .
git commit -m ' new feature ready for PR'
```

```
[new_feature a399978]  new feature ready for PR
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new_feature.md
```

Remember branches are pointers, so what we did was remove the commit (but not the snapshot) from a commit to the main branch (with info about main in it) to a new branch. This leave the main branch pointer pointed at the previous commit and moves only the new branch to the new commit. We get a new hash because the time and commit message are different, but the content is the same.

```
git push
```

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

    git push --set-upstream origin new_feature
```

```
git push --set-upstream origin new_feature
```

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 278 bytes | 278.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'new_feature' on GitHub by visiting:
remote:      https://github.com/introcompsys/github-inclass-
brownsarahm/pull/new/new_feature
remote:
To https://github.com/introcompsys/github-inclass-brownsarahm.git
 * [new branch]      new_feature -> new_feature
branch 'new_feature' set up to track 'origin/new_feature'.
```

If we go back to main

```
git checkout main
```

```
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

Our changes are not here locally, but they still are on GitHub

```
ls
```

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

We can try to push our local to GitHub

```
git push
```

```
To https://github.com/introcompsys/github-inclass-brownsarahm.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/introcompsys/github-inclass-
brownsarahm.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

but it does not work because this is will result in losing information, git (and GitHub) will not let you lose information by accident.

the `-f` option stands for force and makes it do things that we otherwise might not want to/

```
git push -f
```

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/introcompsys/github-inclass-brownsarahm.git
 + 0830c39...3c9980a main -> main (forced update)
```

```
git log
```

```
commit 3c9980a4883782386ea3112d521b1b95997b0be6 (HEAD -> main, origin/main,
origin/HEAD)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 17:40:01 2022 -0400

    create about

commit ec3dd020d2767e45b6cd61f7c4ea5f6f2be9a3d8
Merge: 1613072 db2e41d
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 16:56:34 2022 -0400

    Merge pull request #4 from introcompsys/create_readme

    Create README.md

commit db2e41d48129b2c3ad09b78f1c14c2cd295e3eb2 (origin/create_readme)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 16:55:23 2022 -0400

    Create README.md

commit 1613072525c141b13d3ac7db68e8c1dbee70496b
Author: github-classroom[bot] <66690702+github-
classroom[bot]@users.noreply.github.com>
Date:   Wed Sep 14 20:51:29 2022 +0000

    Initial commit
```

```
git status
```

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

nothing to commit, working tree clean
```

8.2. merging

We can merge offline too

```
git merge new_feature main
```

```
Updating 3c9980a..a399978
Fast-forward
 new_feature.md | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new_feature.md
```

```
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
```

8.3. Reverting changes

We can undo *changes* by picking out a specific commit by its name. First elts make two changes and commit eahc.

```
echo "change 1" >> new_feature.md
git add .
git commit -m 'change 1'
```

```
[main 83690bc] change 1
 1 file changed, 1 insertion(+)
```

```
echo "change 2" >> new_feature.md
git add .
git commit -m 'change 2'
```

```
[main 948eda1] change 2
 1 file changed, 1 insertion(+)
```

SO we hav e a file with two chagnes

```
cat new_feature.md
```

```
change 1
change 2
```

We need to know the hash of the commit that we want to undo.

```
git log
```

```
commit 948eda10a2b03a168e9d0291a294a4922afecda5 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 17:08:49 2022 -0400

    change 2

commit 83690bccd9be5da7c3a816dea4f14d6836674623
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 17:08:32 2022 -0400

    change 1

commit a399978f0f20e1b560fbaf59ec831ab42dd6a82a (origin/new_feature, new_feature)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 16:55:41 2022 -0400

     new feature ready for PR

commit 3c9980a4883782386ea3112d521b1b95997b0be6 (origin/main, origin/HEAD)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 17:40:01 2022 -0400

    create about

commit ec3dd020d2767e45b6cd61f7c4ea5f6f2be9a3d8
Merge: 1613072 db2e41d
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 16:56:34 2022 -0400
```

We use git revert

```
git revert 83690bc
```

```
Auto-merging new_feature.md
CONFLICT (content): Merge conflict in new_feature.md
error: could not revert 83690bc... change 1
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git revert --continue".
hint: You can instead skip this commit with "git revert --skip".
hint: To abort and get back to the state before "git revert",
hint: run "git revert --abort".
```

In this case, we get a merge conflict becaues the file was so simple before, git cannot be sure how to undo that small change.

```
cat new_feature.md
```

```
<<<<<<< HEAD
change 1
change 2
=======
>>>>>>> parent of 83690bc (change 1)
```

```
nano new_feature.md
```

we can use nano to resolve the merge conflict. and then check

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

You are currently reverting commit 83690bc.
  (fix conflicts and run "git revert --continue")
  (use "git revert --skip" to skip this patch)
  (use "git revert --abort" to cancel the revert operation)

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

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

we have to add and commit to finish

```
git add .
```

```

```

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

You are currently reverting commit 83690bc.
  (all conflicts fixed: run "git revert --continue")
  (use "git revert --skip" to skip this patch)
  (use "git revert --abort" to cancel the revert operation)

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

```
git commit -m "revert chang 1"
```

```
[main c1807f4] revert chang 1
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
git status
```

and then we are all set.

```
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

We can see git revert undoes the *changes* but **adds* an additional comit, it is keeping track of every change and by default the commit history only moves forward.

```
git log
```

```
commit c1807f4c764bfc47ad6cdea81453ae4c75b4df20 (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 17:16:59 2022 -0400

    revert chang 1

commit 948eda10a2b03a168e9d0291a294a4922afecda5
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 17:08:49 2022 -0400

    change 2

commit 83690bccd9be5da7c3a816dea4f14d6836674623
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 17:08:32 2022 -0400

    change 1

commit a399978f0f20e1b560fbaf59ec831ab42dd6a82a (origin/new_feature, new_feature)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 3 16:55:41 2022 -0400

     new feature ready for PR

commit 3c9980a4883782386ea3112d521b1b95997b0be6 (origin/main, origin/HEAD)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Sep 14 17:40:01 2022 -0400

    create about

commit ec3dd020d2767e45b6cd61f7c4ea5f6f2be9a3d8
Merge: 1613072 db2e41d
```

8.4. Installing from source

Store this outside of your inclass repo

```
cd ..
ls
```

```
2022-09-19                    github-inclass-brownsarahm
```

```
git clone https://github.com/introcompsys/courseutils.git
```

```
Cloning into 'courseutils'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 57 (delta 26), reused 44 (delta 18), pack-reused 0
Receiving objects: 100% (57/57), 9.99 KiB | 3.33 MiB/s, done.
Resolving deltas: 100% (26/26), done.
```

```
cd courseutils/
```

then you can install

> **Important**
>
> on Windows this will probably need to run in either command prompt or Anaconda Prompt if you have that.

```
pip install .
```

```
Processing /Users/brownsarahm/Documents/inclass/systems/courseutils
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Click in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
syscourseutils==0.1.0) (7.1.2)
Requirement already satisfied: pandas in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
syscourseutils==0.1.0) (1.4.4)
Requirement already satisfied: lxml in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
syscourseutils==0.1.0) (4.6.3)
Requirement already satisfied: numpy in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
syscourseutils==0.1.0) (1.23.2)
Requirement already satisfied: pytz>=2020.1 in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
pandas->syscourseutils==0.1.0) (2020.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
pandas->syscourseutils==0.1.0) (2.8.2)
Requirement already satisfied: six>=1.5 in
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages (from
python-dateutil>=2.8.1->pandas->syscourseutils==0.1.0) (1.15.0)
Building wheels for collected packages: syscourseutils
  Building wheel for syscourseutils (setup.py) ... done
  Created wheel for syscourseutils: filename=syscourseutils-0.1.0-py3-none-any.whl
size=3890 sha256=32b131a267661070bdd7cd1d858eb4921283c53f5861cf98ff1464ef878086c6
  Stored in directory: /private/var/folders/8g/px8bm7bj0_j31j71yh6mfd_r0000gn/T/pip-
ephem-wheel-cache-
d08bf6rq/wheels/b2/4a/cc/b21f40ef23301a77d7427d6c6766b0ed01411098736a71e5c5
Successfully built syscourseutils
Installing collected packages: syscourseutils
  Attempting uninstall: syscourseutils
    Found existing installation: syscourseutils 0.1.0
    Uninstalling syscourseutils-0.1.0:
      Successfully uninstalled syscourseutils-0.1.0
Successfully installed syscourseutils-0.1.0
```

Important

The package is updated so for the following to work use

```
cd courseutils
git pull
pip install .
```

```
sysgetassignment --date 2022-09-28
```

```
- [ ] Try exploring your a repo manually and bring more questions
- [ ] Make sure that you have a grading contract that has been reviewed at least
once
```

Then we can *pipe* the output into another command to create an issue.

```
sysgetassignment | gh issue create --title "more practice 1" --body-file -
```

```
Creating issue in introcompsys/github-inclass-brownsarahm

https://github.com/introcompsys/github-inclass-brownsarahm/issues/8
```

8.4.1. Trouble shooting

If anaconda or miniconda is installed: `conda activate`. Otherwise activate a python environment of your choosing

Try pulling updates and reinstalling.

Make sure you reactive bash.

try both `pip` and `pip3`

if you want to make yourself a linkux computer for the cost of a flash drive

8.5. Review today's class

1. Fix any open PRs you have that need to have a commit moved to a different branch, etc.
2. In your github in class repo, create a series of commits that tell as story of how you might have made a mistake and fixed it. Use git log and redirects to write that log to a file in your KWL repo and then annotate your story in `gitstory.md`.
3. Create tracking issues for last week's activities and link them to PRs for any activities you have already completed.

8.6. Prepare for Next Class

1. In a `gitunderstanding.md` list 3-5 items from the following categories (1) things you have had trouble with in git in the past and how they relate to your new understanding (b) things that your understanding has changed based on today's class © things about git you still have questions about
2. Follow up on your grading contract as needed

8.7. More Practice

1. Create an issue on your group repo for a new vocab term or cheatsheet item from your terminal, check that there is not an exisiting issue first. Write the history from this activity to `offlineissue.md` in your kwl repo.
2. If you plan to do projects. Create milestones for the intermediate deadlines (proposal 1, proposal 2, project 1, project 2 with deadlines)

8.8. Questions After Class

8.8.1. Will the use of the courseutils and commands be necessary for the class? or would it just make the class easier?

They're for tracking your progress. We will help you get it working or work around it.

8.8.2. My kwlfilecheck command is still not working how do I fix that?

There's a missing section on the course website for that command, focus on getting `sysgetassignment` for now. Try the things above and if it still does not create a detailed issue with what error you get and what OS you are using on the repo

8.8.3. Can you post directions for how to get the kwl tracking to work on m1 macs

a student did this last year

8.8.4. Does a flash drive that I have made to have ubuntu in 2017 works?

yes

9. How does git *really* work?

9.1. Creating a repo from scratch

We will start in the top level course directory.

```
cd systems
ls
```

Mine looks like this:

```
2022-09-19                    courseutils                   github-inclass-
brownsarahm
```

Yours should also have your kwl repo, group repo, etc.

We can create an empty repo from scratch using `git init`

```
git init test
```

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

It creates a folder and gives us a warning about branch names. If you have a new install you will not see this, because new versions of git have this by default.

fundamentlaly the default branch is not special. you can name it whatever you want.

Historically it was called master.

derived from a master/slave analogy which is not even how git works, but was adopted terminology from other projects

GitHub no longer does

the broader community is as well

git allows you to make your default not be master

literally the person who chose the names “master” and “origin” regrets that choice the name main is a more accurate and not harmful term and the current convention.

```
ls
```

we can see that we hav ea new directory

```
2022-09-19                    github-inclass-brownsarahm
courseutils                   test
```

Now we want to change the name of the default branch

```
git branch -m main
```

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

but we have to cd into it first.

```
cd test/
```

```
git branch -m main
```

```
git status
```

```
On branch main

No commits yet

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

and we have a completely empty repo.

```
ls .git/
```

```
HEAD            description     info            refs
config          hooks           objects
```

we've looked at most of these, but we have not been able to see the objects before. We will work with those now.

9.2. Searching the file system

We can use the bash command `find` to search the file *system* note that this does not search the **contents** of the files, just the names.

```
find .git/objects
```

we have a few items in that directory and the directory itself.

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

We can limit by type, to only files with the `-type` option set to `f`

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

And we have no results. We have no objects yet.

```

```

9.3. Git Objects

There are 3 types:

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

Let's create our first one. git uses hashes as the key. We give the hashing function some content, it applies the algorithm and returns us the hash as the reference to that object. We can also write to our database wit this.

```
echo "test content" | git hash-object -w --stdin
```

- git hash-object would take the content you handed to it and merely return the unique key
- w option then tells the command to also write that object to the database
- –stdin option tells git hash-object to get the content to be processed from stdin instead of a file
- the `|` is called a pipe (what we saw before was a redirect) it pipes a *process* output into the next command
- echo would write to stdout, withthe pip it passes that to std in of the git-hash

> **Important**
>
> pipes are an important content too. we're seeing them in context of real uses, and we will keep seing them. Pipes connec tthe std out of one command t othe std in of the next.

we get back the hash:

```
d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

and we can check if it wrote to the database.

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

```
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

and we see a file that it was supposed to have!

We can use `cat-file` to use the object by referencing at least 4 characters that are unique from the full hash, not the file name. (`70460` will not work)

```
git cat-file -p d6704
```

`cat-file` requires *an* option `-p` is for pretty print

```
test content
```

we see it stored the contet we expected.

```
cat .git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

but without `git`'s `cat-file`' we cannot read that file.

```
xK??OR04f(I-.QH??+I?+?K?        ```
```

9.4. Hashing a file

let's create a file

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

and store it in our database, by hashing it

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

```
83baae61804e65cc73a7201a7252750c76066a30
```

we can look at what we have.

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

```
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

and what is in the working directory.

```
ls
```

```
test.txt
```

> **Note**
>
> the workign directory and the git repo are not strictly the same thing, and can be different like this. Mostly they will stay in closer relationship that we currently have unless we use plumbling commands, but it is good to build a solid understanding of how the `.git` directory relates to your working directory.

9.5. Writing to the index

So far, even though we have hashed the object, git still thinks the file is untracked, because it is not in the tree and there are no commits that point to that part of the tree.

```
git status
```

```
On branch main

No commits yet

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

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

First, let's edit the file

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

so it looks like this:

```
cat test.txt
```

```
version 1
version 2
```

and then hash the new version of the file too.

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

```
0c1e7391ca4e59584f8b773ecdbbb9467eba1547
```

And we can look at our objects again

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

```
.git/objects/0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

We have the string we wrote directly and the two versions of the file.

we can verify the last one that we have not looked at yet.

```
git cat-file -p 0c1e
```

```
version 1
version 2
```

TO add this to the index

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

> **Note**
>
> all of the hashes we have so far, are the same for all of us and are repeatable. Try making a new repo with a different name and repeating these steps, same hashes every time.

> **Deeper Exploration idea**
>
> What changes would make the hashes different

- this the plumbing command `git update-index` updates (or in this case creates an index, the staging area of our repository)
- the `--add` option is because the file doesn't yet exist in our staging area (we don't even have a staging area set up yet)
- `--cacheinfo` because the file we're adding isn't in your directory but is in the database.
- in this case, we're specifying a mode of 100644, which means it's a normal file.
- then the hash object we want to add to the index (the content) in our case, we want the hash of the first version of the file, not the most recent one.
- finally the file name of that content

> **Note**
>
> the \ here allows for line wrapping

Now let's see what git knows:

```
git status
```

```
On branch main

No commits yet

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

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:   test.txt
```

We added the first version of the file to the staging area, so that version is ready to commit but we have changed the version in our working directory relative to the version from the hash object that we put in the staging area so we *also* have changes not staged.

we can see what those changes are with `git diff`

```
git diff test.txt
```

```
diff --git a/test.txt b/test.txt
index 83baae6..0c1e739 100644
--- a/test.txt
+++ b/test.txt
@@ -1 +1,2 @@
version 1
+version 2
```

the first few lines tell us what git is doing; it says that it is comparing the content at hash 83baae6 to hash 0c1e739 (as expected, 83baae6 is the one we put in the index; 0c1e739 is the last thing we hashed, which is also still what is our working directory)

the next three lines say that we the a one is missing lines relative to the b one and a quantitative desription.

the last two lines are the file with changes between the two versions marked the second version has "version 2" added to it relative to the first one.

9.6. Creating a commit manually

We can echo a commit message through a pip into the commit-tree plumbing function to cmmit a particular hashed object.

```
echo "first commit" | git commit-tree 83baa
```

```
fatal: 83baae61804e65cc73a7201a7252750c76066a30 is not a valid 'tree' object
```

But we can actually only commit `tree` objects

we have

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

```
.git/objects/0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

the `-t` option of cat-file can tell us the type:

```
git cat-file 0c1e -t
```

```
blob
```

```
git cat-file d6704 -t
```

```
blob
```

```
git cat-file 83baa -t
```

```
blob
```

These are all blob objects, the actual *content* that we are storing

we can write a tree though:

```
git write-tree
```

```
d8329fc1cc938780ffdd9f94e0d364e0ea74f579
```

and look at this:

```
git cat-file -p d8329
```

```
100644 blob 83baae61804e65cc73a7201a7252750c76066a30    test.txt
```

and check its type:

```
git cat-file -t d8329
```

```
tree
```

Now that we hav ea tree object we can commit the tree.

```
echo "first commit" | git commit-tree d8329
```

```
e09139a38f4fd6d82715c32aab9adfed67a87ba5
```

and we get back a hash. But notice that this hash is unique for each of us. Because the commit has information about the time stamp and our user. The above hash is the one I got during class, but when I re-ran this while typing the notes I got a different hash (`d450567fec96cbd8dd514313db9bcb96ad7664b0`) even though I have the same name and e-mail because the time changed.

> **Important**
>
> verify that you have the same git status and list of objects as below

```
git status
```

```
On branch main

No commits yet

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

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:   test.txt
```

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

```
.git/objects/0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/d8/329fc1cc938780ffdd9f94e0d364e0ea74f579
.git/objects/e0/9139a38f4fd6d82715c32aab9adfed67a87ba5
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

we will all have 4 of the same objects and one unique one.

9.7. Review today's class

1. Review the notes
2. For the core "Porcelain" git commands we have used (add, commit), make a table of which git plumbing commands (of those we have seen) they use in `gitplumbingreview.md` in your KWL repo. it might be multiple Porcelain for each plumbing.
3. Contribute to your group repo and review a classmate's contribution

9.8. Prepare for Next Class

1. Make notes on *how* you use IDEs for the next couple of weeks using the template file in the course notes (will provide prompts and tips). We will come back to these notes in class later, but it is best to record over a time period instead of trying to remember at that time. Store your notes in your kwl repo in `idethoughts.md` on an `ide_prep` branch.
2. make sure that you have a `test` git repo that matches the notes.

```
# IDE Thoughts

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


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

Further Reading

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

More on ethnography in CS

9.9. More Practice

1. Read about git internals to review what we did in class in greater detail. Make `gitplumbingdetail.md` by copying your `gitplumbingreview.md` and then add in the full detail including all plumbing commands. Also add one more high level command (revert, reset, pull, fetch) to your table.
2. Add to your `gitplumbingdetail.md` file explanations of the main git operations we have seen (add, commit, push) in your own words in a way that will either help you remember or how you would explain it to someone else at a high level. This might be analogies or explanations using other programming concepts or concepts from a hobby. Add this under a subheading `##` with a descriptive title (for example "Git In terms of ")
3. For one thing your understanding changed or an open question you, look up or experiment to find the answer and contribute the question and answer to the course website.

9.10. Questions After Class

9.10.1. Would any of the pull requests we make in class count towards the Hacktoberfest requirements?

The PRs to your group and KWL repos will not, because those are private. I will make it so that contributions to the course website can count. To be honest, I will only give you each one qualifying PR to the course site for Hacktoberfest.

I will also though make the courseutils repo qualify and create some issues there to outline improvements to it that I want to make. That would require some python knowledge too.

The Hacktoberfest participation page also has links to good repos to contribute to.

9.10.2. how do you make a local repo that's private on github?

We will push this repo from beign created locally to GitHub next week.

9.10.3. Are hash algorithms supposed to always output a unique value?

Yes for each content you put in, it should gibe a unique output. If you put the same content in, most of the time you want a repeatable hashing function

9.10.4. How did you set up the courseutils for tracking. I just want to know out of curiosity.

It's a public repo of python code that I made installable. You can look at the source code if you would like. If you have questions, you can post an issue there or use office hours.

9.10.5. are hash objects exclusively backend mechanics, or can we use our knowledge of them to our advantage"

10. What are git hashes and why are they alaphanumeric?

10.1. What is a hash?

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

Common examples of hashing are lookup tables and encryption with a cyrptographic hash.

A cyrptographic hash is additionally:

- unique
- not reversible
- similar inputs hash to very different values so they appear uncorrelated

Hashes can then be used for a lot of purposes:

- message integrity (when sending a message, the unhashed message and its hash are both sent; the message is real if the sent message can be hashed to produce the same has)
- password verification (password selected by the user is hashed and the hash is stored; when attempting to login, the input is hashed and the hashes are compared)
- file or data identifier (eg in git)

10.2. Hashing in Git

in git, 40 characters that uniquely represent either each object and are used as the key to retrieve the object as a value. Recall there are multiple types of objects: tree, commit, blob.

Git as originally designed to use SHA-1

SHA-1 is weak. Git switched to hardened HSA-1 in response to a collision

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

Learn more about the SHA-1 collision attach

they will change again soon

For now though it's still SHA-1 like.

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

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

- minimum 4
- must be unique

For most project 7 characters is enough and by default, git will give you 7 digits if you use `--abbrev-commit` and git will automatically use more if needed.

```
git log --abbrev-commit --pretty=oneline
```

```
313c1d7 (HEAD -> main) Revert "c4"
a47d362 c5
94d236b c4
7d17fdc c3
c1807f4 revert chang 1
948eda1 change 2
83690bc change 1
a399978 (origin/new_feature, new_feature)  new feature ready for PR
3c9980a (origin/main, origin/HEAD) create about
ec3dd02 Merge pull request #4 from introcompsys/create_readme
db2e41d (origin/create_readme) Create README.md
1613072 Initial commit
```

> **Further Reading**
>
> the pro git book has more details on this

git uses the SHA hash primarily for uniuqeness, not privacy

It does provide some *security* assurances, because we can check the content against the hash to make sure it is what it matches.

We can use git to hash things for us, without writing them:

```
echo "learning hashes" | git hash-object --stdin
```

```
3ba345cea32208d6612e97830fdb0b1ae70ea8bd
```

The SHA-1 digest is 20 bytes or 160 bits, which is 40 characters in hexadecimal. The number of randomly hashed objects needed to ensure a 50% probability of a single collision is about 2^{80} (the formula for determining collision probability is p = (n(n-1)/2) * (1/2^160)). 2^{80} is 1.2 x 1024 or 1 million billion billion. That's 1,200 times the number of grains of sand on the earth.

10.3. What is a Number ?

a mathematical object used to count, measure and label

10.4. What is a number system?

While numbers represent are quantities that conceptually, exist all over, the numbers themselves are a cultural artifact. For example, we all have a value representing a single item.

In modern, western cultures our is a a hindu-arabic

- invented by Hindu mathematicians in India 600 or earlier
- called “Arabic” numerals in the West because Arab merchants introduced them to Europeans
- slow adoption

We use a **place based** system. That means that the position or place of the symbol changes its meaning. So 1, 10, and 100 are all different values. This system is also a decimal system, or base 10. So we refer to the places and the ones (10^0), the tens (10^1), the hundreds(10^2), etc for all powers of 10.

10.4.1. Roman Numerals

Not all systems are place based, for example Roman numerals. In this system the symbols are added if they are the same value or decreasing and subtracted if increasing. There are symbols for specific values: 1=I, V=5, X=10, L =50, C = 100, D=500, M = 1000. Then III = 1+1+1 = 3 and IV = -1 + 5 = 4, VI = 5+1 = 6 and XLIX = -10 + 50 -1 +10 = 49.

10.4.2. Binary

Binary is any base two system, and it can be represented using any different characters.

Binary number systems have origins in ancient cultures:

- Egypt (fractions) 1200 BC
- China 9th century BC
- India 2nd century BC

In computer science we use binary because mechanical computers began using relays (open/closed) to implement logical (boolean) operations and then digital computers use on and off in their circuits.

We represent binary using the same hindu-arabic symbols that we use for other numbers, but only the 0 and 1(the first two). We also keep it as a place-based number system so the places are the ones(2^0), twos (2^1), fours (2^2), eights (2^3), etc for all powers of 2.

so in binary, the number of characters in the word binary is 110.

10.4.3. Octal

Is base 8. This too has history in other cultures, not only in computer science. It is rooted in cultures that counted using the spaces *between* fingers instead of counting using fingers.

use by native americans from present day CA

and

Pamean languages in Mexico

As in binary we use hindu-arabic symbols, 0,1,2,3,4,5,6,7 (the first eight). Then nine is 11.

In computer science we use octal a lot because it reduces every 3 bits of a number in binary to a single character. So for a large number, in binary say `101110001100` we can change to `5614` which is easier to read, for a person.

10.4.4. Hexadecimal

base 16, commonin CS because its 4 bits. we use 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

This is how the git hash is 160 bits, or 20 bytes (one byte is 8 bits) but we represent it as 40 characters. 160/4=40.

10.5. Review today's class

1. review the notes
2. find 2 more real world examples of using other number systems (either different bases or different symbols and bases) that are current. Describe them in `numbers.md`
3. Read about hexpeak from Wikipedia for an overview and one additional source in your kwl repo in `hexspeak.md`. Come up with a word or two on your own.
4. Create a single branch for all of the work related to today's class in your KWL

10.6. Prepare for Next Class

1. Bring to class a scenario where you think a small command line program or bash script could be useful. A command line program is a program that we execute on the command line. For example the courseutils kwlcheck is one I wrote.
2. Bring one scenario in git that you have seen or anticipate that we have not seen the solution for

10.7. More Practice

1. Learn more about how git is working on changing from SHA-1 to SHA-256 and answer the transition questions below)
2. Research the use of non base 10 numbers systems in another culture and contribute one to your group repo
3. (priority) In a language of your choice or pseudocode, write a short program to convert, without using libraries, between all pairs of (binary, decimal, hexidecimal) in `numbers.md`. Test your code, but include in the markdown file enclosed in three backticks so that it is a "code block" write the name of the language after the ticks like:

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

```
## transition questions
1. Why make the switch?
2. Learn more about one collision
3. What impact will the swith have on how git works?
4. If you have scripts that operate on git repos, what might you do to prepare for
the switch?
```

10.8. Questions After Class

10.8.1. Are git commit hashes predictable/significant in any way? Or is the risk of using a low-bit hash generator negligible for github usage?

10.8.2. I am still confused on what TRULY a git hash is. Is it an object? Is it the string to the object?

This is a place where being casual with language makes it hard. A hash, in general is the output of a hashing algorithm. In git, everything that is stored gets hashed and git uses the hashes as the unique values to refer to each stored object. For example, if I take the hash that is the *commit number* from the most recent commit above, I can use `git cat-file` to look at the file, or git object that for the commit.

```
git cat-file -p 313c1d7
```

```
tree 47182af2099fc6075832c13798ee16b713ebb285
parent a47d362409bc4ad0cb57e73b0e7a4c1a1a586f43
author Sarah M Brown <brownsarahm@uri.edu> 1664849752 -0400
committer Sarah M Brown <brownsarahm@uri.edu> 1664849752 -0400

Revert "c4"

This reverts commit 94d236b459d5035ab3f2c2676a888b27cd77e80d.
```

The "contents" of the commit are several things:

- the hash of the snapshot of the contents as a tree object
- the hash of the previous commit
- author information
- commit message

We can also check the type:

```
git cat-file -t 313c1d7
```

```
commit
```

We can check the type of the parent to verify it points to the last commit.

```
git cat-file -t a47d3624
```

```
commit
```

We can then look at the contents of the tree as well, using its has has the file name to first check the type of

```
git cat-file -t 47182af
```

```
tree
```

then display:

```
git cat-file -p 47182af
```

```
040000 tree 95b60ce8cdec1bc4e1df1416e0c0e6ecbd3e7a8c    .github
100644 blob bcc1d9287b5cae329629cf3e0779b065b72dad7a    README.md
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    about.md
100644 blob 96d35c05b19fc42bc44153b8a97512001c37a09e    new_feature.md
```

this is all of the contents of the directory at the time of that commit, one blob object for each file and one tree object for the single directory.

We can also look at a file

```
git cat-file -p bcc1d9
```

```
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
```

and verify it is the same contents as on disk

```
cat README.md
```

```
# github-inclass-brownsarahm
github-inclass-brownsarahm created by GitHub Classroom
(base) brownsarahm@github-inclass-brownsarahm $
```

10.8.3. What are uses of Octal

Octal is convenient *because* it is consistent with binary representations easily. 1 place in octal is 3 bits (and 4 in hexadecimal is 4 bits).

We will see it for file representations, because they have 3 parts, each of which can be on or off (ie it's 3 bits of information).

10.8.4. What is the most interesting number system created?

That is a good thing to explore, but interesting is relative.

10.8.5. the `-w` option to `git hash-object` writes the hash number to a file?

No, it writes the *content* to a file named based on the hash in the `.git/objects` folder.

10.8.6. With the information given to us last class, is it possible to complete the work for gitplumbingreview.md and gitplumbingdetail.md?

Do what you can and that task will come back after next class.

> **Important**
>
> I revised the text of the the activities as posted here relative to what I posted in class to clarify

11. How do git references work?

We will go back to the test repo and review waht we have there.

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

```
test.txt
```

```
cat test.txt
```

```
version 1
version 2
```

```
git status
```

```
On branch main

No commits yet

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

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:   test.txt
```

We have a few files and we have content staged and unstaged as changes to those files.

```
git log
```

```
fatal: your current branch 'main' does not have any commits yet
```

Notice, we have no commits yet even though we had written a commit. This is because the main branch does not point to any commit.

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

```
-bash: fin: command not found
```

To get a better look, we can list the objects.

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

```
.git/objects//0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects//d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects//d8/329fc1cc938780ffdd9f94e0d364e0ea74f579
.git/objects//e0/9139a38f4fd6d82715c32aab9adfed67a87ba5
.git/objects//83/baae61804e65cc73a7201a7252750c76066a30
```

We can see that indeed we have one object that is a commit

```
git cat-file -t e0913
```

```
commit
```

And we can look at its contents

```
git cat-file -p e0913
```

```
tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579
author Sarah M Brown <brownsarahm@uri.edu> 1665005715 -0400
committer Sarah M Brown <brownsarahm@uri.edu> 1665005715 -0400

first commit
```

Let's add one more new file to review

```
echo "new file" > new.txt
```

first we hash the object. The `-w` option writes to the directory.

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

then we get the hash back

```
fa49b077972391ad58037050f2a75f74e3671e92
```

Then we stage it.

```
git update-index --add --cacheinfo 100644 \
> fa49b077972391ad58037050f2a75f74e3671e92 new.txt
```

and again we can look on the overall repo

```
git status
```

```
On branch main

No commits yet

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

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:   test.txt
```

and list out the `.git` objects

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

```
.git/objects//0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects//d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects//d8/329fc1cc938780ffdd9f94e0d364e0ea74f579
.git/objects//fa/49b077972391ad58037050f2a75f74e3671e92
.git/objects//e0/9139a38f4fd6d82715c32aab9adfed67a87ba5
.git/objects//83/baae61804e65cc73a7201a7252750c76066a30
```

```
git cat-file -p 0c1e
```

```
version 1
version 2
```

then we can stage the second version of our first file

```
git update-index --add --cacheinfo 100644 \
> 0c1e7391ca4e59584f8b773ecdbbb9467eba1547 test.txt
```

```
git status
```

```
On branch main

No commits yet

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

and now we see that everything is staged.

Then we confirm what hash is our previous commit

```
git cat-file -t e0913
```

```
commit
```

```
git cat-file -p e0913
```

```
tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579
author Sarah M Brown <brownsarahm@uri.edu> 1665005715 -0400
committer Sarah M Brown <brownsarahm@uri.edu> 1665005715 -0400

first commit
```

Next we write a tree object from the staging area

```
git write-tree
```

```
163b45f0a0925b0655da232ea8a4188ccec615f5
```

```
git cat-file -p 163b4
```

```
100644 blob fa49b077972391ad58037050f2a75f74e3671e92    new.txt
100644 blob 0c1e7391ca4e59584f8b773ecdbbb9467eba1547    test.txt
```

This tree has the current version of each file.

Now we can make a new commit. We will echo the message into the commit-tree plumbing function with a pipr and then we will tell this commit to point to the tree we just created and that its parent is the first commit.

```
echo "second commit" | git commit-tree 163b4 -p e0913
```

```
cf857a865c6088ab9bf90e2732df967f8e2582ab
```

But still when we check status

```
git status
```

```
On branch main

No commits yet

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

The files look staged. This is beacuse of what `git status` does. It compares the current working directory (what we see when we do ls) to the staging area and the content that the current branch points to.

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

```
.git/objects/0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects/d8/329fc1cc938780ffdd9f94e0d364e0ea74f579
.git/objects/cf/857a865c6088ab9bf90e2732df967f8e2582ab
.git/objects/16/3b45f0a0925b0655da232ea8a4188ccec615f5
.git/objects/fa/49b077972391ad58037050f2a75f74e3671e92
.git/objects/e0/9139a38f4fd6d82715c32aab9adfed67a87ba5
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
```

11.1. Git References

Recall the `.git` directory has many other files in it.

```
ls .git
```

```
HEAD            description     index           objects
config          hooks           info            refs
```

Let's look at the refs dir.

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

```
heads   tags
```

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

```

```

there's nothing in the heads dir

Though when we use `git status`

```
git status
```

```
On branch main

No commits yet

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

we see it is “on main” this is because we set the branch to main , but since we have not written there, we have to do it directly. Notice that when we use the porcelain command for commit, it does this automatically; the porcelain commands do many things.

We can write tothat file directly

```
echo cf857a865c6088ab9bf90e2732df967f8e2582ab > .git/refs/heads/main
```

And now if we check git status

```
git status
```

```
On branch main
nothing to commit, working tree clean
```

we see what we expec.

Alternatively, we can use only a short reference to the hash if we use `git update-ref`

```
git update-ref refs/heads/main cf857
```

Now git log also works.

```
git log
```

```
commit cf857a865c6088ab9bf90e2732df967f8e2582ab (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 17 16:55:10 2022 -0400

    second commit

commit e09139a38f4fd6d82715c32aab9adfed67a87ba5
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Oct 5 17:35:15 2022 -0400

    first commit
```

Lets create a branch at the point of our first commit.

```
git update-ref refs/heads/test e0913
```

```
git log
```

```
commit cf857a865c6088ab9bf90e2732df967f8e2582ab (HEAD -> main)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 17 16:55:10 2022 -0400

    second commit

commit e09139a38f4fd6d82715c32aab9adfed67a87ba5 (test)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Oct 5 17:35:15 2022 -0400

    first commit
```

Notice that the commit history in `git log` also lists the branches.

We can also look at the HEAD pointer

```
git symbolic-ref HEAD
```

```
refs/heads/main
```

update the HEAD directly

```
git symbolic-ref HEAD refs/heads/test
```

and see what this does.

```
git status
```

```
On branch test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   new.txt
        modified:   test.txt


```

This is only part of what `git checkout` does, because it switches the branch, but does not update the working directory.

11.2. Git Tags

Just like branches, but used differently.

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

```
heads   tags
```

```
git update-ref refs/tags/v1.0
```

```
usage: git update-ref [<options>] -d <refname> [<old-val>]
   or: git update-ref [<options>]    <refname> <new-val> [<old-val>]
   or: git update-ref [<options>] --stdin [-z]

    -m <reason>           reason of the update
    -d                    delete the reference
    --no-deref            update <refname> not the one it points to
    -z                    stdin has NUL-terminated arguments
    --stdin               read updates from stdin
    --create-reflog       create a reflog


```

```
git update-ref refs/tags/v1.0 cf857a
```

```

```

```
git checkout main
```

```
Switched to branch 'main'
```

```
git log
```

```
commit cf857a865c6088ab9bf90e2732df967f8e2582ab (HEAD -> main, tag: v1.0)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Mon Oct 17 16:55:10 2022 -0400

    second commit

commit e09139a38f4fd6d82715c32aab9adfed67a87ba5 (test)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Wed Oct 5 17:35:15 2022 -0400

    first commit
```

11.3. Adding a remote

We created this repo locally, and we have not set a remote.

```
git remote
```

we can add one with

```
git remote add origin <url/to/remote>
```

in my case, I'll use `https://github.com/introcompsys/toy-repo-brownsarahm.git` you should use your own.

```
git remote
```

```
origin
```

```
ls .git
```

```
HEAD            description     index           logs            refs
config          hooks           info            objects
```

```
ls .git/config
```

```
.git/config
```

```
cat .git/config
```

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

```
git push
```

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

    git push --set-upstream origin main
```

adding a remote directly doesn't link the local branch to the remote branch, so we do that next.

```
git push -u origin main
```

```
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (7/7), 497 bytes | 497.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/introcompsys/toy-repo-brownsarahm.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

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

```
.git/objects//0c/1e7391ca4e59584f8b773ecdbbb9467eba1547
.git/objects//d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
.git/objects//d8/329fc1cc938780ffdd9f94e0d364e0ea74f579
.git/objects//cf/857a865c6088ab9bf90e2732df967f8e2582ab
.git/objects//16/3b45f0a0925b0655da232ea8a4188ccec615f5
.git/objects//fa/49b077972391ad58037050f2a75f74e3671e92
.git/objects//e0/9139a38f4fd6d82715c32aab9adfed67a87ba5
.git/objects//83/baae61804e65cc73a7201a7252750c76066a30
```

11.4. Review today's class

1. Read the notes and repeat the activity if needed
2. use `git cat-file` over the objects to draw a graph diagram of your current status in your test directory include your drawing in `test_repo_map.md` using mermaid syntax to diagram it. Name each node in your graph with 5-7 characters of the hash and the type. eg `0c913 commit`

11.5. Prepare for Next Class

1. Bring ideas of what you want to write a bash script for and/or a small command line program
2. **Windows only, but important** get python working in GitBash
3. Make sure the gh command line tool works in a bash terminal (either MacOs, Linux, GitBash, or WSL)

11.6. More Practice

1. Add "version 3" to the test.txt file and hash that object
2. Add that to the staging area
3. Add the tree from the first commit to the staging area as a subdirectory with `git read-tree --prefix=back <hash>`
4. Write the new tree
5. Make a commit with message "Commit 3" point to that tree and have your second commit as its parent.
6. Update your diagram in `test_repo_map.md` after the following.
7. Update your `gitplumbingdetail.md`

12. Bash Scripts

We'll start today in the top level folder for class

```
ls
```

```
2022-09-19                      github-inclass-brownsarahm
courseutils                     kwl-brownsarahm
fall2022                        test
```

12.1. Bash has programming constructs

We can create variables

```
NAME="Sarah"
```

and use them with a `$`

```
echo $NAME
```

```
Sarah
```

If we forget it, it treats it as a literal

```
echo NAME
```

```
NAME
```

We can also make loops

```
for name in "sarah" "mark" "linda" "david"
do
echo $name
done
ls
```

To run a command and make it a variable we can use `$(command)`. For example:

```
for file in $(ls); do echo $file; done
```

```
2022-09-19
courseutils
fall2022
github-inclass-brownsarahm
kwl-brownsarahm
test
```

12.2. Searching files

`grep` searches files

This searches all of the files in the `test/` repo for “version”

```
grep "version" test/*
```

```
test/test.txt:version 1
test/test.txt:version 2
```

It finds two occurrences, prints the line, and tells us where it found each one

To have something to search we’ll clone the course website

```
git clone https://github.com/introcompsys/fall2022.git
```

```
2022-09-19                      github-inclass-brownsarahm
courseutils                     kwl-brownsarahm
fall2022                        test
```

I set it up in the `_review/`, `_prepare`, and `_practice` folders so that there is the syntax to denote the files that need to be created:

```
```{index} file.md
```
```

To find them all this way, we can `grep` for “index”

```
grep "index" fall2022/_review/*
```

```
fall2022/_review/2022-09-19.md:```{index} terminal
fall2022/_review/2022-09-21.md:```{index} branches.md
fall2022/_review/2022-09-26.md:```{index} abstraction.md
fall2022/_review/2022-09-28.md:```{index} gitlog.txt
fall2022/_review/2022-10-03.md:```{index} gitstory.md
fall2022/_review/2022-10-05.md:```{index} gitplumbingreview.md
fall2022/_review/2022-10-12.md:```{index} numbers.md
fall2022/_review/2022-10-12.md:```{index} hexpeak.md
fall2022/_review/2022-10-17.md:```{index} test_repo_map.md
```

We can use `awk` to pull out a subset of these results. Remember a pipe `|` takes the stdout of one command and sends it to the in of the next.

```
grep "index" fall2022/_review/* | awk '{print $2}'
```

```
terminal
branches.md
abstraction.md
gitlog.txt
gitstory.md
gitplumbingreview.md
numbers.md
hexpeak.md
test_repo_map.md
```

Now we have just the files as a list.

Try `awk '{print $1}'` instead to understand `awk` better

12.3. Check if a file exists

We can see if a file exists with `test`

```
test -f kwl-brownsarahm/branches.md
```

but it doesn't return the value, so we combine it with `if`

```
if test -f kwl-brownsarahm/branches.md
> then
> echo "exists"
> fi
exists
```

Note that `if` is ended with `fi`

12.4. Checking if the your KWL has all the files it needs

We'll save this longer one in a script file

```
nano checker.sh
```

Then we can enter the following content:

```
for file in $(grep "index" fall2022/_review/* |awk '{print $2}')
do
if ! test -f kwl-brownsarahm/$file; then
echo $file
fi
done
```

For reference, we can look at what files are there

```
ls kwl-brownsarahm/
```

```
README.md       branches.md     check.sh        terminal
```

And then use the checker to denote the missing ones

```
bash checker.sh
```

```
abstraction.md
gitlog.txt
gitstory.md
gitplumbingreview.md
numbers.md
hexpeak.md
test_repo_map.md
```

12.5. Review today's class

1. Update your KWL Chart learned column with what you've learned
2. Make a contribution to your group repo and do a peer review of a team member's PR.
3. Use the gh cli, grep, and bash to create `group_contributions.md` to your KWL repo with a list of all of your PRs. Append your history from creating this to the bottom of the file after `## Commands`
4. Move your checker script into your kwl repo and update the paths so that it still works

12.6. Prepare for Next Class

1. On Windows, install Putty (we will use this Monday)
2. Get ready for class by creating `networking.md` in your KWL repo with notes about what you know about networking
3. Add an issue on your KWL repo titled "Self Reflection 10/24" (there will be time on Monday for this, but giving you warning so you have time to think about it). Tag @brownsarahm on this issue.

```
1. Are you where you want to be in this course?
1. Have you been getting feedback on your work?
1. If not, what could help you get back on track?
1. What if any concepts are you most struggling with?
```

> **Important**
>
> Prepare for class tasks are not always based on things that we have already done. Sometimes they are to have you start thinking about the topic that we are *about* to cover. Getting whatever you know about the topic fresh in your mind in advance of class will help what we do in class stick for you when we start.

12.7. More Practice

1. Make your script form class a nested loop to check for all 3 types of files
2. Make a script that gets the updates to the course site and creates a single `todo-YYYY-MM-DD.md` file that has the review, prepare, and practice tasks in it for each date that does not already exist in a `todo/` folder outside of your KWL repo. Save the script as `gathertasks.sh`

12.8. Questions after Class

12.8.1. Can I add another language like C to git bash?

Bash is a shell scripting language. A shell is an interface to your computer's operating system. A terminal is how we access it. We will see that we can use a terminal to call a c compiler.

12.8.2. Does git rely on scripts to do low-level commands for higher ones?

git is written mostly in `C` but there are some shell scripts in its source code

12.8.3. How could I make the checker check a branch on my repo that isn't currently the selected branch?

The easiest way is to switch branches and check again. You might be able to traverse the git repo, but that is not straight forward.

12.8.4. what is one thing I should try and automate with bash

For now see the More practice

13. How can I automate things on GitHub

We can use bash scripts on GitHub as GitHub Actions.

13.1. Let's recall the course utils

We can test it on the URI JupyterHub

To log in, Look in your e-mail for an e-mail about `jupyterhub access`

this is not secure or backed up

> **Warning**
>
> do not use the Jupyter hub server for any sensitive work (it is not secure) or as the only place for any work (it is not backed up; if something goes poorly data could be lost)

We can start with the courseutils

```
git clone https://github.com/introcompsys/courseutils.git
```

```
Cloning into 'courseutils'...
remote: Enumerating objects: 79, done.
remote: Counting objects: 100% (79/79), done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 79 (delta 39), reused 59 (delta 25), pack-reused 0
Unpacking objects: 100% (79/79), done.
```

And then cd and install it

```
cd courseutils/
pip install .
```

```
Defaulting to user installation because normal site-packages is not writeable
Processing /mnt/homes4celsrs/brownsarahm/courseutils
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Click in
/mnt/homes4celsrs/brownsarahm/.local/lib/python3.6/site-packages (from
syscourseutils==0.2.0) (8.0.4)
Requirement already satisfied: pandas in /usr/local/lib64/python3.6/site-packages
(from syscourseutils==0.2.0) (1.0.3)
Requirement already satisfied: lxml in
/mnt/homes4celsrs/brownsarahm/.local/lib/python3.6/site-packages (from
syscourseutils==0.2.0) (4.9.1)
Requirement already satisfied: numpy in /usr/local/lib64/python3.6/site-packages
(from syscourseutils==0.2.0) (1.19.5)
Requirement already satisfied: requests in /usr/lib/python3.6/site-packages (from
syscourseutils==0.2.0) (2.21.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.6/site-
packages (from Click->syscourseutils==0.2.0) (4.8.3)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/site-
packages (from pandas->syscourseutils==0.2.0) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/lib/python3.6/site-
packages (from pandas->syscourseutils==0.2.0) (2.8.0)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/lib/python3.6/site-packages
(from requests->syscourseutils==0.2.0) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/lib/python3.6/site-
packages (from requests->syscourseutils==0.2.0) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3.6/site-
packages (from requests->syscourseutils==0.2.0) (2019.3.9)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/lib/python3.6/site-
packages (from requests->syscourseutils==0.2.0) (1.24.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-packages
(from python-dateutil>=2.6.1->pandas->syscourseutils==0.2.0) (1.15.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/usr/local/lib/python3.6/site-packages (from importlib-metadata->Click-
>syscourseutils==0.2.0) (3.7.4.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/site-packages
(from importlib-metadata->Click->syscourseutils==0.2.0) (3.1.0)
Building wheels for collected packages: syscourseutils
  Building wheel for syscourseutils (setup.py) ... done
  Created wheel for syscourseutils: filename=syscourseutils-0.2.0-py3-none-any.whl
size=3955 sha256=4721a1f1b878d3cd67fa4f5f23445cc6ed967135bd2c20f7245907a295815019
  Stored in directory: /tmp/pip-ephem-wheel-cache-
yigr4d0i/wheels/59/c1/9a/20014d9884280647cef49c48abdc4fda76f1d452ca9110a7ad
Successfully built syscourseutils
Installing collected packages: syscourseutils
Successfully installed syscourseutils-0.2.0
```

```
sysgetassignment --date 2022-10-19
```

By default it gets the prepare for class tasks for that date

````
- [ ] On Windows, install Putty ( we will use this Monday)
2. Get ready for class by creating `networking.md` in your KWL repo with notes about
what you know about networking
3. Add an issue on your KWL repo titled "Self Reflection 10/24" (there will be time
on Monday for this, but giving you warning so you have time to think about it). Tag
@brownsarahm on this issue.
```{index} networking.md
```
```
- [ ] Are you where you want to be in this course?
- [ ] Have you been getting feedback on your work?
- [ ] If not, what could help you get back on track?
- [ ] What if any concepts are you most struggling with?
```
````

```
sysgetassignment --date 2022-10-19 --type practice
```

````
- [ ] Make your script form class a nested loop to check for all 3 types of files
- [ ] Make a script that gets the updates to the course site and creates a single
`todo-YYYY-MM-DD.md` file that has the review, prepare, and practice tasks in it
for each date that does not already exist in a `todo/` folder outside of your KWL
repo. Save the script as `gathertasks.sh`


```{index} checker.sh
```
```{index} gathertasks.sh
```
````

13.2. Writing GitHub Action as bash script practice

GitHub Actions allow us to run code on GitHub's servers.

As a first script, we can put the following YAML content in.

```yaml
on:
  workflow_dispatch

jobs:
  check-contents:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    # Install dependencies
    - name: Set up Python 3.8
      uses: actions/setup-python@v1
      with:
        python-version: 3.8

    - name: Install checker
      run: |
        pip install git+https://github.com/introcompsys/courseutils@main


    - name: check
      run: |
        pwd
```

The first two keys are configuration information.

- `on` tells GitHub what triggers this action
- `jobs` lists the things it has to do

In this case, we will trigger by the a button and our job is named `check-contents`

The next two things are settings for this job; first that it runs on ubuntu and then a list of steps to complete.

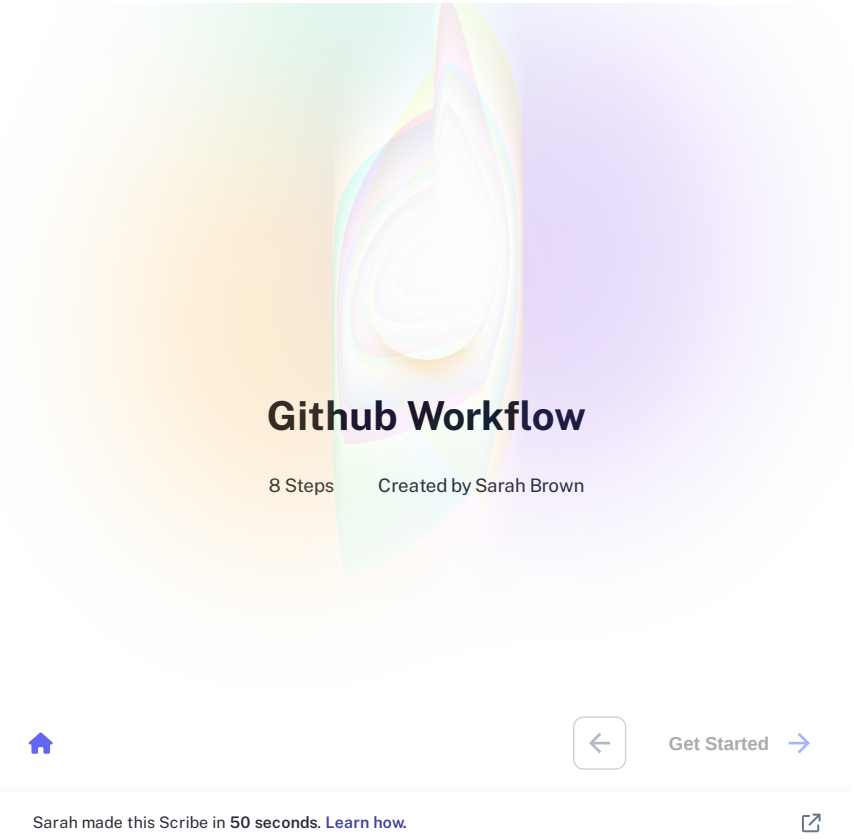
For each step (beginning with a `-`) we can give it a name (optional) and then either `uses` to choose something from the GitHub Action Marketplace or `run` to enter bash commands.

The first two steps are from the marketplace: checkout the repo and setup python.

Then we install the courseutils directly via git and the last step prints the working directory.

To use the action, add it as a file to `.github/workflows/get_assignment.yml`

Then run it and check the output



Then we can modify the script by replacing the `pwd` with the steps we actually need.

```
- name: check
  run: |
    sysgetassignment --date 2022-10-19 | gh issue create --title "review" --body-
file -
  env:
    GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

This gets the todos from last week and creates an issue.

The `env` allows the GitHub action bot to actually edit the repo (make an issue)

13.3. Preview SSH

13.4. Review today's class

1. Review the notes
2. Update your `Get Assignments` action to use the current date. Once the notes are posted, run it to get today's tasks.

13.5. Prepare for Next Class

1. preview ssh materials
2. Review your networking notes before class on Wednesday

13.6. More Practice

1. Brainstorm three potential uses for GitHub actions in `actions.md`

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 | _ | _ | _ |
```

Required Files

Prepare

gitreflection.md gitunderstanding.md idethoughts.md networking.md

Review

terminal.md branches.md abstraction.md gitlog.txt gitstory.md gitplumbingreview.md numbers.md hexpeak.md test_repo_map.md group_contributions.md checker.sh

More Practice

gitoffline.md abstraction.md assemblyexplore.md worflows.md offlineissue.md gitplumbingdetail.md numbers.md gitplumbingdetail.md test_repo_map.md checker.sh gathertasks.sh actions.md

Team Repo

Contributions

Your team repo is a place to build up a glossary of key terms and a “cookbook” of “recipes” of common things you might want to do on the shell, bash commands, git commands and others.

For the glossary, follow the jupyterbook syntax.

For the cookbook, use standard markdown.

to denote code inline `use single backticks`

```
to denote code inline `use single backticks`
```

to make a code block use 3 back ticks

````
```
to make a code block use 3 back ticks
```
````

To nest blocks use increasing numbers of back ticks.

To make a link, `[show the text in squarebrackets](url/in/parenthesis)`

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.

> **Important**
>
> The grade free zone covers classes 1-5, ending on 2022-09-21.

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.

2022-09-07

related notes

Activities:

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

2022-09-12

related notes

Activities:

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 the first two columns of your KWL chart

2022-09-14

related notes

Activities:

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

related notes

Activities:

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

related notes

Activities:

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

related notes

Activities:

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

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

2022-09-28

related notes

Activities:

1. Practice with git log and redirects to write the commit history of your main branch for your kwl chart to a file `gitlog.txt` and commit that file to your kwl repo.
2. Review the notes
3. Update your kwl chart with what you have learned or new questions

2022-10-03

related notes

Activities:

1. Fix any open PRs you have that need to have a commit moved to a different branch, etc.

2. In your github in class repo, create a series of commits that tell as story of how you might have made a mistake and fixed it. Use git log and redirects to write that log to a file in your KWL repo and then annotate your story in `gitstory.md`.
3. Create tracking issues for last week's activities and link them to PRs for any activities you have already completed.

2022-10-05

related notes

Activities:

1. Review the notes
2. For the core "Porcelain" git commands we have used (add, commit), make a table of which git plumbing commands (of those we have seen) they use in `gitplumbingreview.md` in your KWL repo. it might be multiple Porcelain for each plumbing.
3. Contribute to your group repo and review a classmate's contribution

2022-10-12

related notes

Activities:

1. review the notes
2. find 2 more real world examples of using other number systems (either different bases or different symbols and bases) that are current. Describe them in `numbers.md`
3. Read about hexpeak from Wikipedia for an overview and one additional source in your kwl repo in `hexspeak.md`. Come up with a word or two on your own.
4. Create a single branch for all of the work related to today's class in your KWL

2022-10-17

related notes

Activities:

1. Read the notes and repeat the activity if needed
2. use `git cat-file` over the objects to draw a graph diagram of your current status in your test directory include your drawing in `test_repo_map.md` using mermaid syntax to diagram it. Name each node in your graph with 5-7 characters of the hash and the type. eg `0c913 commit`

2022-10-19

related notes

Activities:

1. Update your KWL Chart learned column with what you've learned
2. Make a contribution to your group repo and do a peer review of a team member's PR.
3. Use the gh cli, grep, and bash to create `group_contributions.md` to your KWL repo with a list of all of your PRs. Append your history from creating this to the bottom of the file after `## Commands`
4. Move your checker script into your kwl repo and update the paths so that it still works

2022-10-24

related notes

Activities:

1. Review the notes

2. Update your `Get Assignments` action to use the current date. Once the notes are posted, run it to get today's tasks.

> **Important**
>
> These tasks are not always based on things that we have already done. Sometimes they are to have you start thinking about the topic that we are *about* to cover. Getting whatever you know about the topic fresh in your mind in advance of class will help what we do in class stick for you when we start.

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

2022-09-07

> **Important**
>
> The grade free zone covers classes 1-5, ending on 2022-09-21.

related notes

Activities:

- 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

related notes

Activities:

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

related notes

Activities:

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, called `gitreflection.md` 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

related notes

Activities:

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

related notes

Activities:

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

related notes

Activities:

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`

2022-09-28

related notes

Activities:

1. Try exploring your a repo manually and bring more questions
2. Make sure that you have a grading contract that has been reviewed at least once

2022-10-03

related notes

Activities:

1. In a `gitunderstanding.md` list 3-5 items from the following categories (1) things you have had trouble with in git in the past and how they relate to your new understanding (b) things that your understanding has changed based on today's class © things about git you still have questions about
2. Follow up on your grading contract as needed

2022-10-05

related notes

Activities:

1. Make notes on *how* you use IDEs for the next couple of weeks using the template file in the course notes (will provide prompts and tips). We will come back to these notes in class later, but it is best to record over a time period instead of trying to remember at that time. Store your notes in your kwl repo in `idethoughts.md` on an `ide_prep` branch.
2. make sure that you have a `test` git repo that matches the notes.

```
# IDE Thoughts

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


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

2022-10-12

related notes

Activities:

1. Bring to class a scenario where you think a small command line program or bash script could be useful. A command line program is a program that we execute on the command line. For example the courseutils kwlcheck is one I wrote.
2. Bring one scenario in git that you have seen or anticipate that we have not seen the solution for

2022-10-17

related notes

Activities:

1. Bring ideas of what you want to write a bash script for and/or a small command line program
2. **Windows only, but important** get python working in GitBash
3. Make sure the gh command line tool works in a bash terminal (either MacOs, Linux, GitBash, or WSL)

2022-10-19

related notes

Activities:

1. On Windows, install Putty (we will use this Monday)
2. Get ready for class by creating `networking.md` in your KWL repo with notes about what you know about networking
3. Add an issue on your KWL repo titled "Self Reflection 10/24" (there will be time on Monday for this, but giving you warning so you have time to think about it). Tag @brownsarahm on this issue.

```
1. Are you where you want to be in this course?
1. Have you been getting feedback on your work?
1. If not, what could help you get back on track?
1. What if any concepts are you most struggling with?
```

2022-10-24

related notes

Activities:

1. preview ssh materials
2. Review your networking notes before class on Wednesday

> **Important**
>
> The grade free zone covers classes 1-5, ending on 2022-09-21.

More Practice

> **Note**
>
> these are listed by the date they were *posted*

2022-09-07

related notes

Activities:

2022-09-12

related notes

Activities:

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

related notes

Activities:

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

related notes

Activities:

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

related notes

Activities:

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

2022-09-26

related notes

Activities:

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?

1. 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?
```

2022-09-28

related notes

Activities:

1. ¸ Read about different workflows in git and add responses to the below in a `workflows.md` in your kwl repo. Two good places to read from are Git Book and the atlassian Docs
2. Contribute either a glossary term, cheatsheet item, or additional resource/reference to your group repo.
3. Complete one peer review of a team mate's contribution

```
## Workflow Reflection

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

2022-10-03

related notes

Activities:

1. Create an issue on your group repo for a new vocab term or cheatsheet item from your terminal, check that there is not an exisiting issue first. Write the history from this activity to `offlineissue.md` in your kwl repo.
2. If you plan to do projects. Create milestones for the intermediate deadlines (proposal 1, proposal 2, project 1, project 2 with deadlines)

2022-10-05

related notes

Activities:

1. Read about git internals to review what we did in class in greater detail. Make `gitplumbingdetail.md` by copying your `gitplumbingreview.md` and then add in the full detail including all plumbing commands. Also add one more high level command (revert, reset, pull, fetch) to your table.
2. Add to your `gitplumbingdetail.md` file explanations of the main git operations we have seen (add, commit, push) in your own words in a way that will either help you remember or how you would explain it to someone else at a high level. This might be analogies or explanations using other programming concepts or concepts from a hobby. Add this under a subheading `##` with a descriptive title (for example “Git In terms of ”)
3. For one thing your understanding changed or an open question you, look up or experiment to find the answer and contribute the question and answer to the course website.

2022-10-12

related notes

Activities:

1. Learn more about how git is working on changing from SHA-1 to SHA-256 and answer the transition questions below)
2. Research the use of non base 10 numbers systems in another culture and contribute one to your group repo
3. (priority) In a language of your choice or pseudocode, write a short program to convert, without using libraries, between all pairs of (binary, decimal, hexidecimal) in `numbers.md`. Test your code, but include in the markdown file enclosed in three backticks so that it is a “code block” write the name of the language after the ticks like:

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

```
## transition questions
1. Why make the switch?
2. Learn more about one collision
3. What impact will the swith have on how git works?
4. If you have scripts that operate on git repos, what might you do to prepare for
the switch?
```

2022-10-17

related notes

Activities:

1. Add “version 3” to the test.txt file and hash that object
2. Add that to the staging area
3. Add the tree from the first commit to the staging area as a subdirectory with `git read-tree --prefix=back <hash>`
4. Write the new tree
5. Make a commit with message “Commit 3” point to that tree and have your second commit as its parent.
6. Update your diagram in `test_repo_map.md` after the following.
7. Update your `gitplumbingdetail.md`

2022-10-19

related notes

Activities:

1. Make your script form class a nested loop to check for all 3 types of files
2. Make a script that gets the updates to the course site and creates a single `todo-YYYY-MM-DD.md` file that has the review, prepare, and practice tasks in it for each date that does not already exist in a `todo/` folder outside of your KWL repo. Save the script as `gathertasks.sh`

2022-10-24

related notes

Activities:

1. Brainstorm three potential uses for GitHub actions in `actions.md`

KWL File Information

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.

Examples of Deeper Explorations:

- Discussed Cryptography and Hashing in class:
  - Writing a multi-paragraph summary in a .md file of how cryptography evolved and what caused it to evolve (i.e. SHA-1 being decrypted)
- Discussed number systems:
  - Creating a .md file discussing number systems and their importance. This file could include code blocks demonstrating number system conversions.

- Discussed a topic that left you still a little confused or their was one part that you wanted to know more about.
    - Create a .md file that details your journey from confusion or shallow understanding to a full understanding. This file would include the sources that you used to gather a deeper understanding.

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.

Project Examples

- One type of project would be to do a research project on a topic we cover in class and create a .md file with your findings that demonstrates your knowledge of the topic. The .md file would include an **Abstract**, **Body**, **Reflection** including what you did and what you learned from it, and a **Bibliography**. Potential research topics include:
  - Motherboards
  - CPUs: Their History, Evolution, and How They Work
  - GPUs: A Graphics Card That Revolutionized Machine Learning
  - The Differences Between Operating Systems: MacOS vs Windows VS Linux
  - Abstraction For Dummies: Explaining Abstract Concepts to the Layman
- Another type of project could be to create a program using the tools taught in class to maintain the program. What would be included in this would be a .md reporting your findings that demonstrates an understanding of the tools used and a link to the repository hosting the program including **documentation** written for the program.

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

What is the difference between porcelain and plumbing commands?

Porcelain commands are high level commands (add, commit, push) while plumbing commands are low-level (hash-obejct, send-pack). Porcelain are used more often because they are often easier to use unless there is a specific need to use a plumbing command.

My command line says I cannot use a password

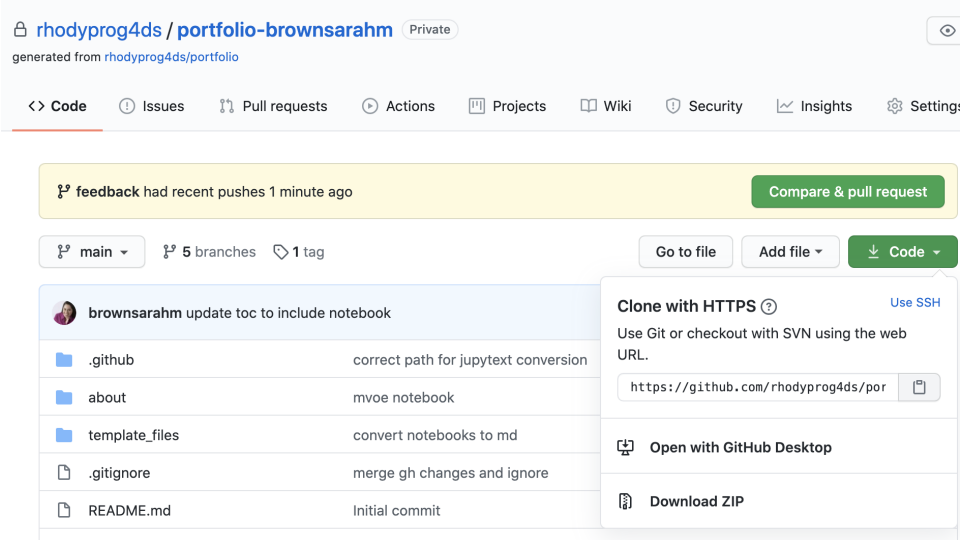
GitHub has strong rules about authentication You need to use SSH with a public/private key; HTTPS with a Personal Access Token or use the GitHub CLI auth

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
```

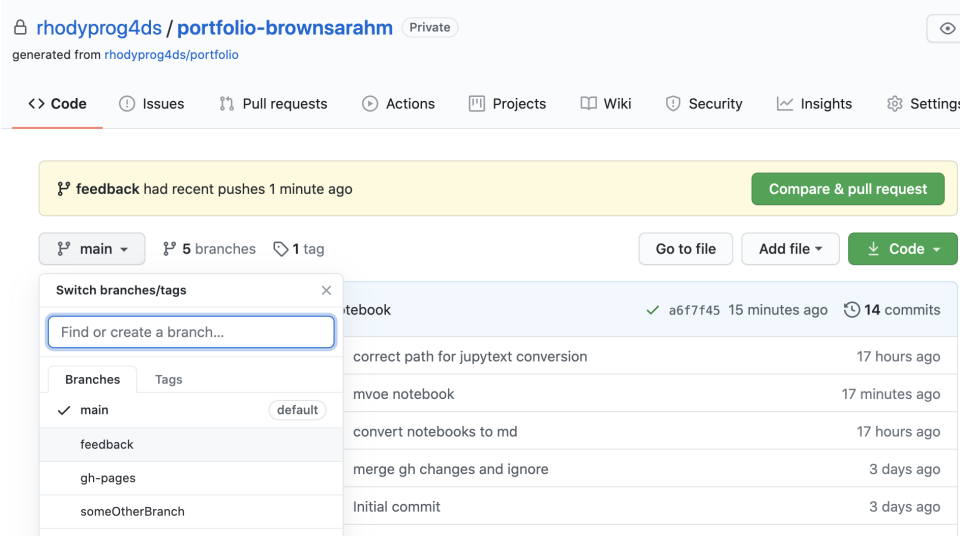
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

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

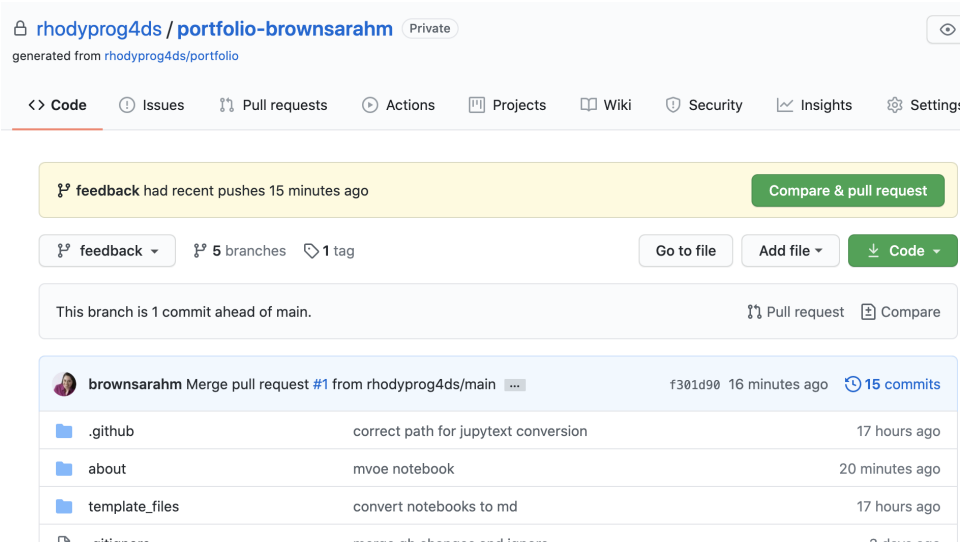
When you merged the Feedback pull request you advanced the `feedback` branch, so we need to hard reset it back to before you did any work. To do this, first check it out, by navigating into the folder for your repository (created when you cloned above) and then checking it out, and making sure it's up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

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

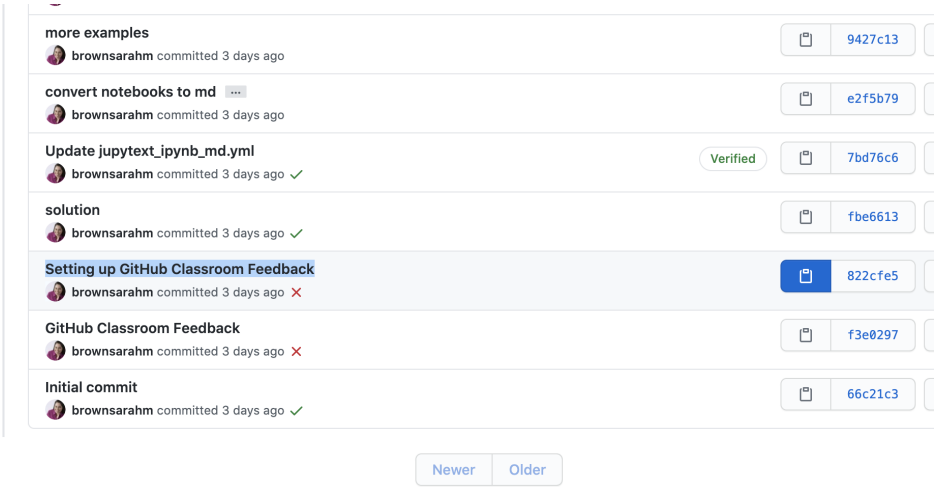
Now, you have to figure out what commit to revert to, so go back to GitHub in your browser, and swithc to the feedback branch there. Click on where it says `main` on the top right next to the branch icon and choose feedback from the list.



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.

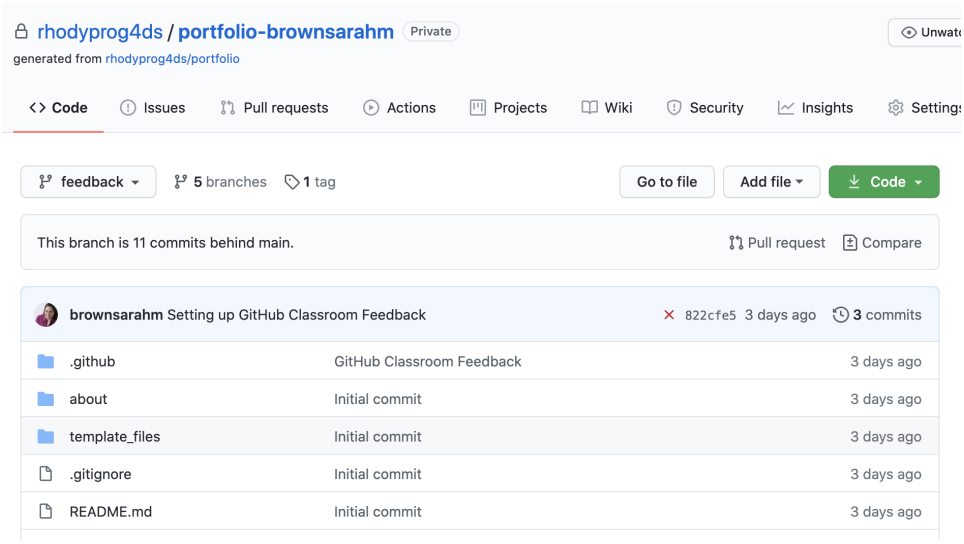
```
git push origin -f
```

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

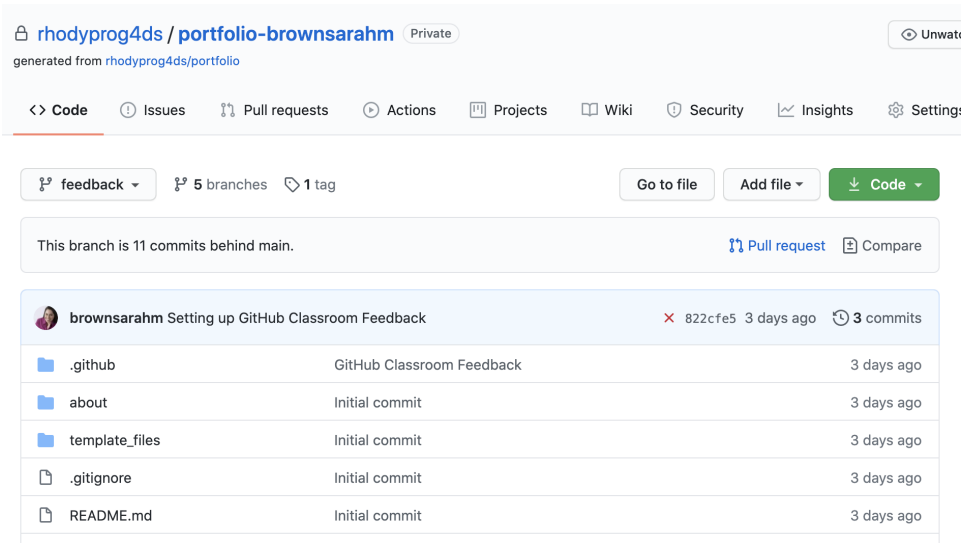
```
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
 + f301d90...822cfe5 feedback -> feedback (forced update)
```

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

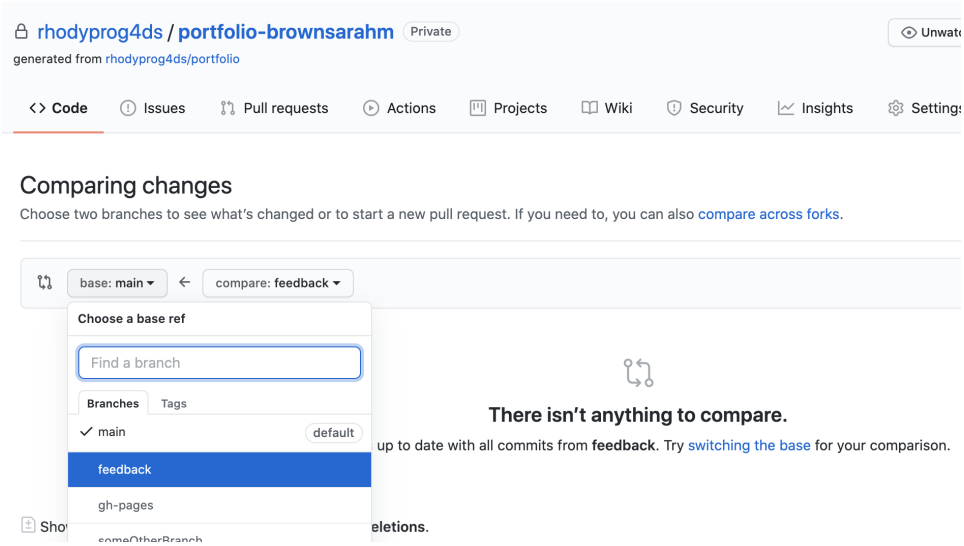
Now back on GitHub, in your browser, click on the code tab. It should look something like this now. Notice that it says, “This branch is 11 commits behind main” your number will be different but it should be 1 less than the number you had when you checked `git status`. This is because we reverted the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



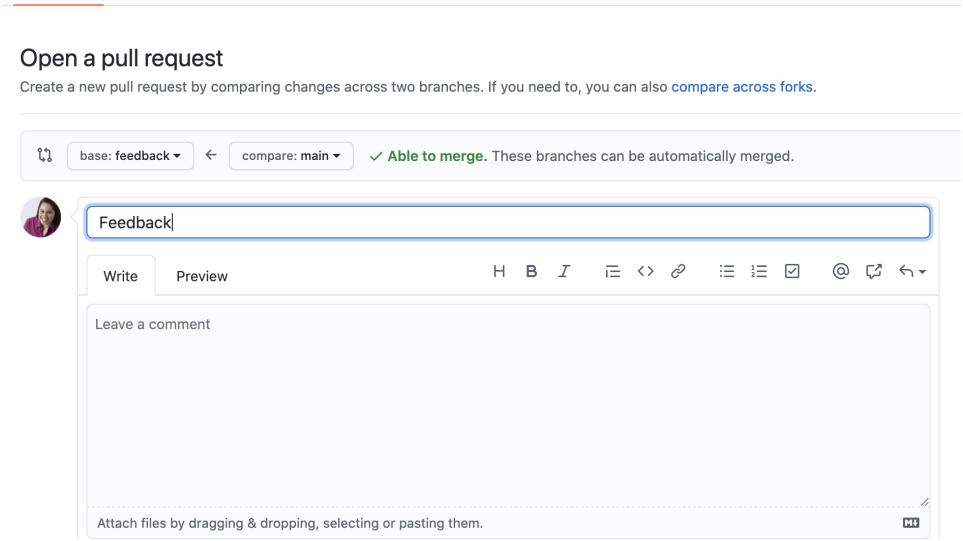
Now, you need to recreate your Pull Request, click where it says pull request.



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.
```

How would I continue to add to this FAQ page by myself?

Getting everything set up is really simple! Though it gets a little bit technical. This will describe the process to fork any remote repo and allow you to use `git fetch` to get information from the remote repo you created the fork from and `git push` to your fork (which can then be merged by the original owner you forked from).

This is useful for suggesting changes to a repository that you do not own or have access to at all, with the original owner still being able to accept or deny the changes.

On the GitHub repository you wish to fork, click the “Fork” icon at the top of the page. You can then rename the repo, only copy the main branch etc., however we just want to continue.

It then brings you to the new remote repository that is now under your name.

Clone the repo using `git clone <clone-link>` in your local directory of choice.

> NOTE: This repo is not connected to the original remote repo that you forked from. It is connected to your forked repo you just created.

You can use the below command

```
git remote -v
```

where you will get (most times) the following back:

> origin <https://github.com/.git> (fetch) origin <https://github.com/.git> (push)

Your local repository does not have the ability to pull or fetch from the original repository, only the fork you have created. You can fix this easily though.

Using the command

```
git remote add upstream <original-repo-clone-link>
```

adds the original repository as a remote ref that you can use in other commands, for example `git fetch` or `git pull`

> NOTE: “upstream” in the above command can be replaced with whatever you want, just note that you will use that name when using other commands referencing it.

When using `git remote -v` again, you now get the following back:

> origin <https://github.com/.git> (fetch) origin <https://github.com/.git> (push) upstream <https://github.com/.git> (fetch) upstream <https://github.com/.git> (push)

You can now run the command

```
git fetch upstream
```

to successfully fetch all information from the original remote repo.

Running

```
git merge upstream/main
```

then merges the fetched information from the remote repo into your local forked repo.

When pushing your changes, you will continue to either use

```
git push
```

or

```
git push origin
```

of course after pushing upstream for the first time.

Remember, you are not pushing to the original repo, you are pushing to your fork. You can only pull and fetch from the original remote repo.

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.

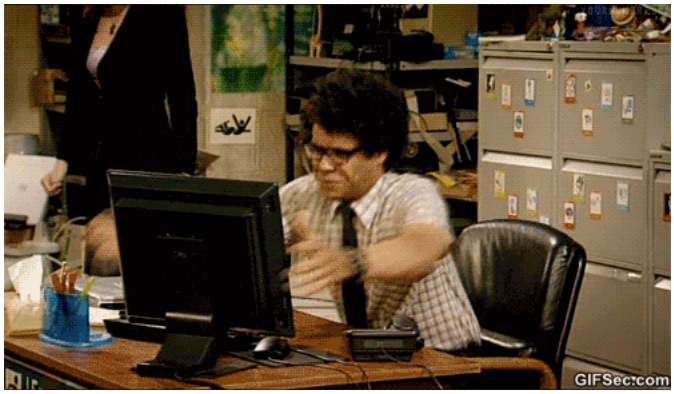
on email

- 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.

More info on cpus

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.

By Professor Sarah M Brown