About this Site

Contents

Skip to main content

Welcome to the course website for Computer Systems and Programming Tools in Fall2023 with Professor Brown.

This class meets TuTh 12:30-1:45 in library 166

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

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.

Skip to main content

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.

Ques
unans
answ
with a
will b

Click here!

Special tips will be formatted like this

Check your Comprehension

Questions to use to check your comprehension will looklike this

Contribute

Chances to earn community badges will sometimes be marked like this

Computer Systems and Programming Tools

About this course

In this course we will study the tools that we use as programmers and use them as a lens to study the computer system itself. We will begin with two fundamental tools: version control and the shell. We will focus on git and bash as popular examples of each. Sometimes understanding the tools requires understanding an aspect of the system, for example git uses cryptographic hashing which requires understanding number systems. Other times the tools helps us see how parts work: the shell is our interface to the operating system.

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.

Should you download the syllabus and rely on your offline copy?

No, because the syallabus changes

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

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

Land Acknowledgement

Important

The University of Rhode Island land acknowledgment is a statement written by members of the University community in close partnership with members of the Narragansett Tribe. For more information see the university land acknowledgement page

The University of Rhode Island occupies the traditional stomping ground of the Narragansett Nation and the Niantic People. We honor and respect the enduring and continuing relationship between the Indigenous people and this land by teaching and learning more about their history and present-day communities, and by becoming stewards of the land we, too, inhabit.

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

On BrightSpace, you will find links to other resource, this site and others. Any links that are for private discussion among those enrolled in the course will be available only from Brightspace.

Not

Seein
loggi
being

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.

Important

Prismia is **only** for use during class, we do not read messages there outside of class time

Skip to main content

Course Website

The course website 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.

You will also need the gh CLI. It will help with authentication and allow you to work with other parts of github besides the core git operations.

Important

You need to install this on Mac

Programming Environment

In this course, we will use several programming environments. In order to participate in class and 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)
- a C compiler
- Git
- A bash shell
- A web browser compatible with Jupyter Notebooks
- nano text editor (comes with GitBash and default on MacOS)
- one IDE with git support (default or via extension)
- the GitHub CLI on all OSs

Skip to main content

Windows- option A | Windows - option B | MacOS | Linux | Chrome OS

- Install python via Anaconda video install
- Git and Bash with GitBash (video instructions).

Zoom

(backup only & office hours only)

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 office hours and any online class sessions if needed best if you download the Zoom client on your computer. Please log in and configure your account. Please add a photo (can be yourself or something you like) to your account so that we can still see your likeness in some form when your camera is off. You may also wish to use a virtual background and you are welcome to do so.

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

Grading

This section of the syllabus describes the principles and mechanics of the grading for the course. The course is designed around your learning so the grading is based on you demonstrating how much you have learned.

Additionally, since we will be studying programming tools, we will use them to administer the course. To give you a chance to get used to the tools there will be a grade free zone for the first few weeks.

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. Apply common design patterns and abstractions to understand new code bases, programming tools, and components of systems.
2. Apply appropriate programming workflows using context-relevant tools that enable adherance to best practices for effective code, developer time efficiency, and collaboration.
3. 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
4. Identify how information flows across levels of abstraction.
5. Discuss implications of design choices across levels of abstraction
6. Describe the social context in 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 designed to reflect how deeply you learn the material, even if it takes you multiple attempts to truly understand a topic. The topics in this course are all topics that will come back in later courses in the Computer Science major, so it is important that you understand each of them *correctly* so that it helps in the next course.

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 material. You will be required to demonstrate understanding of the connections between ides from different parts of the course.

- Earning a C in this class means you have a general understanding; you will know what all the terms mean; you could follow along in a meeting where others were discussing systems concepts and use core tools for common tasks. You know where to start when looking things up.
- 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; you can know where to start and can form good hypotheses about why uncommon errors have occurred; you can confidently figure out new complex systems.

The course is designed for you to *succeed* at a level of your choice. As you accumulate knowledge, the grading in this course is designed to be cumulative instead of based on deducting points and averaging. No matter what level of work you choose to engage in, you will be expected to revise work until it is correct. The material in this course will all come back in other 300 and 400 level CSC courses, so it is essential that you do not leave this course with misconceptions, as they will make it harder for you to learn related material later.

If you made an error in an assignment what do you need to do?

Read the suggestions and revise the work until it is correct.

Penalty-free Zone

Since learning developer tools is a core learning outcome of the course, we will also use them for all aspects of administering the course. This will help you learn these tools really well and create accountability for getting enough practice with core operations, but it also creates a high stakes situation: even submitting your work requires you understanding the tools. This would not be very fair at the beginning of the semester.

For the first three weeks we will have a low stakes penalty-free zone where we will provide extra help and reminders for how to get feedback on your work. In this period, deadlines are more flexible as well. If work is submitted incorrectly, we will still see it because we will manually go look for all activities. After this zone, we will assume you *chose* to skip something if we do not see it.

During the Penalty-Free zone, we will help you figure that out and fix it so you get credit for it. After that, you have to fix it on your own (or in office hours) in order to get credit.

Important

If there are terms in the rest of this section that do not make sense while we are in the penalty-free zone, do not panic. This zone exists to help you get familiar with the terms needed.

During the third week, you will create a course plan where you establish your goals for the course and I make sure that you all understand the requirements to complete your goals.

What happens if you’re confused by the grading scheme right now?

Nothing to worry about, we will review it again in week three after you get a chance to build the right habits and learn vocabulary. We will also give you an activity that helps us to be sure that you understand it at that time.

Learning Badges

Your grade will be based on you choosing to work with the material at different levels and participating in the class community in different ways. Each of these represents different types of badges that you can earn as you accumulate evidence of your learning and engagment.

- experience: guided in class acitivies
- review: just the basics
- practice: a little bit more indepdendent
- explore: posing your own directions of inquiry
- build: in depth- application of course topics

All of these badges will be tracked through PRs in your kwl repo. Each PR must have a title that includes the badge type and associated date. We will use scripts over these to track your progress.

To earn a D you must complete:

- 22 experience badges
- 13 lab check outs

To earn a C you must complete:

- 22 experience badges
- 13 lab check outs
- 18 review badges

To earn a B you must complete:

- 22 experience badges
- 13 lab check outs

    - 18 practice badges
    - 12 review + 12 practice

For an A you must complete:

- 22 experience badges
- 13 lab check outs
- your choice:
    - 18 practice badges + 6 explore badges
    - 18 review badges + 3 build badges
    - 6 review badges + 12 practice badges + 4 explore badges + 1 build badges
    - 12 review badges + 6 practice badges+ 2 explore badges + 2 build badges

You can also mix and match to get +/-. For example (all examples below assume 22+ experience badges aand 13 lab checkouts)

- A-: 18 practice + 4 explore
- B+: 6 review + 12 practice + 4 explore
- B-: 6 review + 12 practice
- B+: 24 practice
- C+: 12 review + 6 practice

Warning

These counts assume that the semester goes as planned and that there are 26 available badges of each base type (experience, review, practice). If the number of available badges decreases by more than 2 for any reason (eg snowdays, instructor illness, etc) the threshold for experience badges will be decreased.

Important

There will be 20 review and practice badges available after the penalty free zone. This means that missing the review and practice badges in the penalty free zone cannot hurt you. However, it does not mean it is a good idea to not attempt them, not attempting them at all will make future badges harder, because reviewing early ideas are important for later ideas.

You cannot earn both practice and review badges for the same class session, but most practice badge requirements will include the review requirements plus some extra steps.

In the second half of the semester, there will be special *integrative* badge opportunities that have multipliers attached to them. These badges will count for more than one. For example an integrative 2x review badge counts as two review badges. These badges will be more complex than regular badges and therefore count more.

Can you do any combination of badges?

No, you cannot earn practice and review for the same date.

Experience Badges

- preparing for class
- following along with the activity (creating files, using git, etc)
- responding to 80% of inclass questions (even incorrect or `:idk:``)
- reflecting on what you learned
- asking a question at the end of class

You can make up an experience badge by:

- preparing for class
- reading the posted notes
- completing the activity from the notes
- producing an "experience report" OR attending office hours

An experience report is evidence you have completed the activity and reflection questions. The exact form will vary per class, if you are unsure, reach out ASAP to get instructions. These are evaluated only for completeness/ good faith effort. Revisions will generally not be required, but clarification and additional activity steps may be adcised if your evidence suggests you may have missed a step.

Do you earn badges for prepare for class?

No, prepare for class tasks are folded into your experience badges.

What do you do when you miss class?

Read the notes, follow along, and produce and experience report or attend office hours.

What if I have no questions?

Learning to ask questions is important. Your questions can be clarifying (eg because you misunderstood something) or show that you understand what we covered well enough to think of hypothetical scenarios or options or what might come next. Basically, focused curiosity.

Review and Practice Badges

The tasks for these badges will be defined at the bottom of the notes for each class session *and* aggregated to badge-type specific pages on the left hand side fo the course website.

You can earn review and practice badges by:

- creating an issue for the badge you plan to work on
- completing the tasks
- submitting files to your KWL on a new branch
- creating a PR, linking the issue, and requesting a review
- revising the PR until it is approved
- merging the PR after it is approved

Skip to main content

At the end of notes and on the separate pages in the activities section on the left hand side

You should create one PR per badge

The key difference between review and practice is the depth of the activity. Work submitted for review and practice badges will be assessed for correctness and completeness. Revisions will be common for these activities, because understanding correctly, without misconceptions, is important.

Important

Revisions are to help you improve your work **and** to get used to the process of making revisions. Even excellent work can be improved. The **process** of making revisions and taking good work to excellent or excellent to exceptional is a useful learning outcome. It will help you later to be really good at working through PR revisions; we will use the same process as code reviews in industry, even though most of it will not be code alone.

Explore Badges

Explore badges require you to pose a question of your own that extends the topic. For inspiration, see the practice tasks and the questions after class.

Details and more ideas are on the explore page.

You can earn an explore badge by:

- creating an issue proposing your idea (consider this ~15 min of work or less)
- adjusting your idea until given the proceed label
- completing your exploration
- submitting it as a PR
- making any requested changes
- merging the PR after approval

For these, ideas will almost always be approved, the proposal is to make sure you have the right scope (not too big or too small). Work submitted for explore badges will be assessed for depth beyond practice badges and correctness. Revisions will be more common on the first few as you get used to them, but typically decraese as you learn what to expect.

Important

Revisions are to help you improve your work **and** to get used to the process of making revisions. Even excellent work can be improved. The **process** of making revisions and taking good work to excellent or excellent to exceptional is a useful learning outcome. It will help you later to be really good at working through PR revisions; we will use the same process as code reviews in industry, even though most of it will not be code alone.

You should create one PR per badge

Build Badges

Skip to main content

You can earn a build badge by:

- creating an issue proposing your idea and iterating until it is given the “proceed” label
- providing updates on your progress
- completing the build
- submitting a summary report as a PR linked to your proposal issue
- making any requested changes
- merging the PR after approval

You should create one PR per badge

For builds, since they’re bigger, you will propose intermediate milestones. Advice for improving your work will be provided at the milestones and revisions of the compelte build are uncommon. If you do not submit work for intermediate review, you may need to revise the complete build. The build proposal will assessed for relevance to the course and depth. The work will be assessed for completeness in comparison to the propsal and correctness. The summary report will be assessed only for completeness, revisions will only be requested for skipped or incomplete sections.

Community Badges

Community badges are awarded for extra community participation. Both programming and learning are most effective in good healthy collaboration. Since being a good member of our class community helps you learn (and helps others learn better), some collaboration is required in other badges. Some dimensions of community participation can only be done once, for example fixing a typo on the course website, so while it’s valuable, all students cannot contribute to the course community in the same way. To reward these unique contributions, you can earn a community badge.

You can see some ideas as they arise by issues labeled `community`.

Community badges can replace missed experience, review, and practice badges, upgrade a review to a practice badge, or they can be used as an alternate way to earn a + modifier on a D,C,or B (URI doesn’t award A+s, sorry). Community badges are smaller, so they are not 1:1 replacements for other badges. You can earn a maximum of 14 community badges, generally one per week. Extra helpful contributions may be awarded 2 community badges, but that does not increase your limit. When you earn them, you can plan how you will use it, but they will only be officially applied to your grade at the end of the semester. They will automatically be applied in the way that gives you the maximum benefit.

Community Badge values:

- 3 community = 1 experience badge
- 4 community = 1 review
- 7 community = 1 practice.
- 3 community badges + 1 review = 1 practice.
- 10 community = add a `+` to a D,C, or B, **note that this is more efficient.**

You can earn community badges by:

- fixing small issues on the course website (during penalty free zone only)
- contributing extra terms or reviews to your team repo
- sharing articles and discussing them in the course discussions

Skip to main content

You will maintain a list of your contributions in your KWL repo in the `community_contributions.md` file. Every individual change to this file (representing one contribution) should be commited to a new branch and then submitted as a PR, with a review requested from @brownsarahm.

> **Note**
>
> Some participation in your group repo and a small number of discussions will be required for experience, review, and practice badges. This means that not every single contribution or peer review to your team repo will earn a community badge.

Example(nonexhaustive) uses:

- 22 experience + 17 review + 11 community = C (replace 2 experience, 1 review)
- 24 experience + 17 review + 5 community = C (replace 1 review)
- 24 experience + 18 review + 10 community = C+ (modifier)
- 24 experience + 18 practice + 10 community = B+ (modifier)
- 23 experience + 18 practice + 13 community = B+ (modifier, replace 1 experience)
- 24 experience + 16 practice + 2 review + 10 community = B (upgrade 2 review)
- 24 experience + 10 review + 10 community + 6 practice + 3 explore + 2 build = A (replace 2 review)
- 24 experience + 14 review + 10 community + 4 practice + 3 explore + 2 build = A (upgrade 2 review to practice)
- 24 experience + 12 review + 14 community + 4 practice + 3 build =A (replace 2 practice)

These show that community badges can save you work at the end of the semester by reducing the number of practice badges or simplifying badges

🎁 Free corrections

All work must be correct and complete to earn credit. In general, this means that when your work is not correct, we will give you guiding questions and advice so that you can revise the work to be correct. Most of the time asking you questions is the best way to help you learn, but sometimes, especially for small things, showing you a correct example is the best way to help you learn.

Additionally, on rare occasions, a student can submit work that is incorrect or will have down-the-line consquences but does not demonstrate a misunderstanding. For example, in an experience badge, putting text below the `#` line instead of replacing the hint within the `< >`. Later, we will do things within the kwl repo that will rely on the title line being filled in, but it's not a big revision where the student needs to rethink about what they submitted.

In these special occasions, good effort that is not technically correct may be rewarded with a 🎁. In this case, the instructor or TA will give a suggestion, with the 🎁 emoji in the comment and leave a review as "comment" instead of "changes requested" or "approved". If the student commits the suggestion to acknowledge that they read it, the instructor will then leave an approving review. Free corrections are only available when revisions are otherwise eligible. This means that they cannot extend a deadline and they are not available on the final grading that occurs after our scheduled "exam time".

> **Not**
>
> We d
> assig
> The c
> be th

Skip to main content

These free corrections are used at the instructional team's discretion and are not guaranteed.

This means that, for example, the same mistake the first time, might get a 🎁, a second will probably be a hint, and a third or fourth time might be a regular revision where we ask you to go review prior assignments to figure out what you need to fix with a broad hint instead of the specific suggestion

IDEA

If the course response rate on the IDEA survey is about 75%, 🎁 will be applicable to final grading. **this includes the requirement of the student to reply**

Deadlines

There will be fixed feedback hours each week, if your work is submitted by the start of that time it will get feedback. If not, it will go to the next feedback hours.

We do not have a final exam, but URI assigns an exam time for every class. The date of that assigned exam will be the final due date for all work including all revisions.

Experience badges

Prepare for class tasks must be done before class so that you are prepared. Missing a prepare task could require you to do an experience report to make up what you were not able to do in class.

If you miss class, the experience report should be at least attempted/drafted (though you may not get feedback/confirmation) before the next class that you attend. This is strict, not as punishment, but to ensure that you are able to participate in the next class that you attend. Skipping the experience report for a missed class, may result in needing to do an experience report for the next class you attend to make up what you were not able to complete due to the missing class activities.

If you miss multiple classes, create a catch-up plan to get back on track by contacting Dr. Brown.

Review and Practice Badges

These badges have 5 stages:

- posted: tasks are on the course website
- planned: an issue is created
- started: one task is attempted and a draft PR is open
- completed: all tasks are attempted PR is ready for review, and a review is requested
- earned: PR is approved (by instructor or a TA) and work is merged

Tip

these badges *should* be started before the next class. This will set you up to make the most out of each class session. However, only prepare for class tasks have to be done immediately.

Skip to main content

attempted when you have answered the questions or submitted evidence of doing an activity or asked a sincere clarifying question.

If a badge is planned, but not started within one week it will become expired and ineligble to be earned. You may request extensions to complete a badge by updating the PR message, these will typically be granted. Extensions for starting badges will only be granted in exceptional circumstances.

Expired badges will receive a comment and be closed

Once you have a good-faith attempt at a complete badge, you have until the end of the semester to finish the revisions in order to *earn* the badge.

Tip

Try to complete revisions quickly, it will be easier for you

Explore Badges

Explore badges have 5stages:

- proposed: issue created
- in progress: issue is labeled “proceed” by the instructor
- complete: work is complete, PR created, review requested
- revision: “request changes” review was given
- earned: PR approved

Explore badges are feedback-limited. You will not get feedback on subsequent explore badge proposals until you earn the first one. Once you have one earned, then you can have up to two in progress and two in revision at any given time.

Build Badges

You may earn at most one build badge per month, with final grading in December. To earn three build badges, you must earn the first one by the end of October.

Ungrading Option

At the end of the semester, you have the option of submitting a final reflection that states what grade you think you deserve, and justifies it by summarizing what you have learned and providing evidence of that. Instructions for this option will be provided as we approach the end of the semester. The policy of no submitted content that was not generated by you still applies. If you take this option, you may be required to also take an oral exam by appointment to supplement the evidence provided in your reflection.

This option exists in recognition of the fact that grading schemes are not perfect and I am truly committed to your learning. If you think that the grading scheme described on this page is working out to you earning a different grade than you deserve and you can support that with strong evidence that you have learned, you can have the grade you deserve.

Skip to main content

share your thoughts on this option in the discussions for the class and then

Academic Honesty Violation Penalty

All of your work must reflect your own thinking and understanding. The work that you submit must all be your own work or content that was provided to you in class, it cannot include text that was generated by an AI or plagiarized in any other way.

If you are found to submit prismia responses that do not reflect your own thinking or that of discussion with peers as directed, the experience badge for that class session will be ineligible.

If work is suspected, you will be allowed to take an oral exam in lab time to contest and prove that your work reflects your own understanding.

The first time you will be allowed to appeal through an oral exam. If your appeal is successful, your counter resets. If you are found to have violated the policy then no further work will be graded for the remainder of the semester

If you are found to submit work that is not your own for a review or prepare badge, the review and prepare badges for that date will be ineligible and the penalty free zone terms will no longer apply to the first six badges.

If you are found to submit work that is not your own for an explore or build badge, that badge will not be awarded and your maximum badges at the level possible will drop to 2/3 of the maximum possible.

Badge Procedures

This page includes more visual versions of the information on the badge page. You should read both, but this one is often more helpful, because some of the processes take a lot of words to explain and make more sense with a diagram for a lot of people.

Prepare work and Experience Badges Process

Warning

This was changed substantively on 2023-09-08

This is for a single example with specific dates, but it is similar for all future dates

The columns (and purple boxes) correspond to branches in your KWL repo and the yellow boxes are the things that you have to do. The “critical” box is what you have to wait for us on. The arrows represent PRs (or a local merge for the first one)

sequenceDiagram participant P as prepare Sep 12 participant E as experience Sep 12 participant M as main note over P: complete prepare work<br/> between feb Sep 7 and Sep12 note over E: run experience badge workflow <br/> at the end of class Sep12 P ->> E: local merge or PR you that <br/> does not need approval note over E: fill in experience reflection critical Badge review by instructor or TA E ->> M: Experience badge PR option if edits requested note over E: make requested edits option when approved note over M: merge badge PR end
In the end the commit sequence for this will look like the following:

Skip to main content

"gitunderstanding.md" branch experience-2023-09-12 checkout experience-2023-09-12 commit id: "initexp" merge prepare-2023-09-12 commit id: "fillinexp" commit id: "revisions" tag:"approved" checkout main merge experience-2023-09-12
Where the “approved” tag represents and approving reivew on the PR.

Review and Practice Badge

Legend:

flowchart TD badgestatus[[Badge Status]] passive[/ something that has to occur<br/> not done by student /] student[Something for you to do] style badgestatus fill:#2cf decisionnode{Decision/if} sta[action a] stb[action b] decisionnode --> |condition a|sta decisionnode --> |condition b|stb subgraph phase[Phase] st[step in phase] end
This is the general process for review and practice badges

flowchart TD %% subgraph work[Steps to complete] subgraph posting[Dr Brown will post the Badge] direction TB write[/Dr Brown finalizes tasks after class/] post[/Dr. Brown pushes to github/] link[/notes are posted with badge steps/] posted[[Posted: on badge date]] write -->post post -->link post --o posted end subgraph planning[Plan the badge] direction TB create[/Dr Brown runs your workflow/] decide{Do you need this badge?} close[close the issue] branch[create branch] planned[[Planned: on badge date]] create -->decide decide -->|no| close decide -->|yes| branch create --o planned end subgraph work[Work on the badge] direction TB start[do one task] commit[commit work to the branch] moretasks[complete the other tasks] ccommit[commit them to the branch] reqreview[request a review] started[[Started <br/> due within one week <br/> of posted date]] completed[[Completed <br/>due within two weeks <br/> of posted date]] wait[/wait for feedback/] start --> commit commit -->moretasks commit --o started moretasks -->ccommit ccommit -->reqreview reqreview --> wait reqreview --o completed end subgraph review[Revise your completed badges] direction TB prreview[Read review feedback] approvedq{what type of review} merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] discuss[reply to comments] prreview -->approvedq approvedq -->|changes requested|edit edit -->|last date to edit: May 1| prreview approvedq -->|comment|discuss discuss -->prreview approvedq -->|approved|merge merge --o earned end posting ==> planning planning ==> work work ==> review %% styling style earned fill:#2cf style completed fill:#2cf style started fill:#2cf style posted fill:#2cf style planned fill:#2cf

Explore Badges

flowchart TD subgraph proposal[Propose the Topic and Product] issue[create an issue] proposed[[Proposed]] reqproposalreview[Assign it to Dr. Brown] waitp[/wait for feedback/] proceedcheck{Did Dr. Brown apply a proceed label?} branch[start a branch] progress[[In Progress]] iterate[reply to comments and revise] issue --> reqproposalreview reqproposalreview --> waitp reqproposalreview --> proposed waitp --> proceedcheck proceedcheck -->|no| iterate proceedcheck -->|yes| branch branch --> progress iterate -->waitp end subgraph work[Work on the badge] direction TB moretasks[complete the work] ccommit[commit work to the branch] reqreview[request a review] wait[/wait for feedback/] complete[[Complete]] moretasks -->ccommit ccommit -->reqreview reqreview --o complete reqreview --> wait end subgraph review[Revise your work] direction TB prreview[Read review feedback] approvedq{what type of review} revision[[In revision]] merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] prreview -->approvedq approvedq -->|changes requested|edit edit --> prreview edit --o revision approvedq -->|approved| merge merge --o earned end proposal ==> work work ==> review %% styling style proposed fill:#2cf style progress fill:#2cf style complete fill:#2cf style revision fill:#2cf style earned fill:#2cf

Build Badges

flowchart TD subgraph proposal[Propose the Topic and Product] issue[create an issue] proposed[[Proposed]] reqproposalreview[Assign it] waitp[/wait for feedback/] proceedcheck{Did Dr. Brown apply a proceed label?} branch[start a branch]

Skip to main content

reqproposalreview --> proposed waitp --> proceedcheck proceedcheck -->|no| iterate proceedcheck -->|yes| branch branch --> progress iterate -->waitp end subgraph work[Work on the badge] direction TB commit[commit work to the branch] moretasks[complete the work] draftpr[Open a draft PR and <br/> request a review] ccommit[incorporate feedback] reqreview[request a review] wait[/wait for feedback/] complete[[Complete]] commit -->moretasks commit -->draftpr draftpr -->ccommit moretasks -->reqreview ccommit -->reqreview reqreview --> complete reqreview --> wait end subgraph review[Revise your work] direction TB prreview[Read review feedback] approvedq{what type of review} revision[[In revision]] merge[Merge the PR] edit[complete requested edits] earned[[Earned <br/> due by final grading]] prreview -->approvedq approvedq -->|changes requested|edit edit --> prreview edit -->revision approvedq -->|approved| merge merge --o earned end proposal ==> work work ==> review %% styling style proposed fill:#2cf style progress fill:#2cf style complete fill:#2cf style revision fill:#2cf style earned fill:#2cf

Community Badges

These are the instructions from your `community_contributions.md` file: For each one:

- In the community_contributions.md file on your kwl repo, add an item in a bulleted list (start the line with -)
- Include a link to your contribution like `[text to display](url/of/contribution)`
- create an individual pull request titled “Community-shortname” where `shortname` is a short name for what you did. approval on this PR by Dr. Brown will constitute credit for your grade
- request a review on that PR from @brownsarahm

> **Important**
>
> You want one contribution per PR for tracking

flowchart TD contribute[Make a contribution <br> *typically not in your KWL*] link[Add a link to your <br> contribution to your<br> communiyt_contribution.md<br> in your KWL repo] pr[create a PR for that link] rev[request a review <br> from @brownsarahm] contribute --> link link --> pr --> rev rev --> approved[Dr. Brown approves] --> merge[Merge the PR] merge --o earned

Detailed Grade Calculations

> **Important**
>
> This page is generated with code and calculations, you can view them for more precise implementations of what the english sentences mean.

▶ Show code cell source

Grade cutoffs for total influence are:

▶ Show code cell source

Skip to main content

letter	threshold
letter	
D	44
D+	62
C-	80
C	98
C+	116
B-	134
B	152
B+	170
A-	188
A	206

The total influence of each badge is as follows:

▶ Show code cell source

	badge_type	badge	complexity	weight	influence
0	learning	experience	2.0	1.000000	2.000000
1	learning	review	3.0	1.000000	3.000000
2	learning	practice	6.0	1.000000	6.000000
3	learning	explore	9.0	1.000000	9.000000
4	learning	build	36.0	1.000000	36.000000
0	community	plus	1.0	1.800000	1.800000
1	community	experience_makeup	1.0	0.666667	0.666667
2	community	review_makeup	1.0	0.750000	0.750000
3	community	review_upgrade	1.0	1.000000	1.000000
4	community	practice_makeup	1.0	0.857143	0.857143

▶ Show code cell source

▶ Show code cell source

Important

the labels on the horizontal axis are just example names, they do not have any meaning, I just have not figured out what I want to replace them with that might have meaning and need some sort of unique identifier there for the plot to work.

Warning

Officially what is on the Grading page is what applies if this page is in conflict with that.

The total influence of a badge on your grade is the product of the badge's weight and its complexity. All learning badges have a weight of 1, but have varying complexity. All community badges have a complexity of 1, but the weight of a community badge can vary depending on what learning badges you earn.

There are also some hard thresholds on learning badges. You must have:

- 22 experience badges to earn a D or above
- at least 18 additional badges total across review and practice to earn above C or above

Only community badges can make exceptions to these thresholds. So if you are missing learning badges required to get to a threshold, your community badges will fill in for those. If you meet all of the thresholds, the community badges will be applied with

Skip to main content

If you are on track for an A, community badges can be used to fill in for learning badges, so for example, at the end of the semester, you might be able to skip some the low complexity learning badges (experience, review, practice) and focus on your high complexity ones to ensure you get an A.

More precisely the order of application for community badges:

- to make up missing experience badges
- to make up for missing review or practice badge to reach a total of 18 between these two types
- to upgrade review to practice to meet a threshold
- to give a step up (highest weight)

Schedule

Overview

The following is a tentative 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 getting started with the programming tools.

What tools do Computer Scientists use?

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. While studying the tools and how they work, we will get to see how some common abstractions are re-used throughout the fields and it gives a window and good motivation to begin considering how the computer actually works.

Topics:

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

Skip to main content

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

Tentative 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.

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, FIXME. 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 FIXME, 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

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

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 you are able to successfully complete the course.

Office Hours & Communication

Announcements

Announcements will be made via GitHub Release. You can view them online in the releases page or you can get notifications by watching the repository, choosing "Releases" under custom see GitHub docs for instructions with screenshots. You can choose GitHub only or e-mail notificaiton from the notification settings page

Warning

For the first few classes they will be made by BrightSpace too, but that will stop

Sign up to watch

Watch the repo and then create a file called `community.md` in your kwl repo and add a link to this section, like:

```
- [watched the repo as per announcements](https://introcompsys.github.io/spring2023/syllabus/communi
```

put this on a branch called `watch_community_badge` and title your PR "Community-Watch"

Help Hours

Skip to main content

Day	Time	Location	Host
Monday	11-1	Zoom	Marcin
Monday	4-5	Zoom	Dr. Brown
Tuesday	11:15-12:30	Tyler 140	Amoy
Wednesday	1-2	Zoom	Marcin
Thursday	11:15-12:30	Tyler 140	Amoy
Friday	10-11	Zoom	Marcin
Friday	3:45-5	Tyler 140	Amoy
Friday	4-5	Tyler 134	Dr. Brown (most weeks)

Online office hours locations are linked on the GitHub Organization Page

Important

You can only see them if you are a “member” to join, use the “Whole Class Discussion” link in prismia.

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 tpo or you find an additional helpful resource related to something)

...

You can submit a pull request for the typo above, but be sure to check the pull request tab of the repo before submitting to see if it has already been submitted.

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

Skip to main content

Should you e-mail your work?

No, request a pull request review or make an issue if you are stuck

1. Welcome, Introduction, and Setup

1.1. Introductions

- Dr. Sarah Brown
- Please address me as Dr. Brown or Professor Brown,

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. Prismia

- instead of slides
- you can message us
- we can see all of your responses
- emoji!

questions can be "graded"

- this is instant feedback
- participation will be checked, not impact your final grade
- this helps both me and you know how you are doing

1.2.1. Some Background

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

This information will help me prepare

1.3. This course will be different

- no Brightspace
- 300 level = more independence
- I will give advice, but only hold you accountable to a minimal set
- High expectations, with a lot of flexibility to work in a way that works for you

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

1.3.2. Learning comes in many forms

- different types of material are best remembered in different ways
- some things are hard to explain, but watching it is very concrete

1.4. Learning is the goal

- producing outputs as fast as possible is not learning
- in a job, you may get paid to do things fast
- your work also needs to be correct, without someone telling you it is
- in a job you are trusted to know your work is correct, your boss does not check your work or grade you
- to get a job, you have to interview, which means explaining, in words, to another person how to do something

1.5. What about AI?

Large Language Models will change what programming looks like, but understanding is always going to be more effective than asking an AI. Large language models actually do not know anything, they just know what languages loook like and generate text.

if you cannot tell it when it's wrong, you can not add value for a company

learning deeply means you can actually use them effectively and add value beyond an AI

1.6. This is a college course

- more than getting you one job, a bootcamp gets you one job
- build a long (or maybe short, but fruitful) career
- build critical thinking skill that makes you adaptable
- have options

1.7. "I never use what I learned in college"

- very common saying
- it's actually a sign of deep learning
- when we have expertise, we do not even notice when we apply it
- college is not about the facts, but the processes

We may not think that we use the fact that we know the order of letters in the English language very often. Most of us learned the alphabet with a song, but we do not sing that on a dialy basis.

However, we do fill out forms where we have to, for example, find the state we live in, in a dropdown and knowing the alphabetical order of the states helps us find ours faster.

When you know things really well, you apply them without noticing.

1.8. How does this work?

1.8.1. In class:

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

1.8.2. Outside of class:

1. Build your cookbook with your team
2. Read notes Notes to refresh the material, check your understanding, and find more details
3. Practice material that has been taught
4. Activate your memory of related things to what we will cover
5. Read articles/ watch videos to either fill in gaps or learn more details
6. Bring questions to class

1.9. Getting started

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. You will also add material to the repository to produce evidence of your learning.

Accept the assignment to create your repo

1. click on .gihub, then workflows, then track.yml
2. click the 3 dots menu and select delete
3. commit directly to main with the default message
4. back to the main code tab
5. Click the pencil to edit the readme
6. Add your name
7. Add a descriptive commit message
8. Choose create a branch and open a pull request
9. name the branch

Skip to main content

1.10. 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).

1.10.1. How it fits into your CS degree

1.11. In your degree

Tip

know
where
unde

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

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.

1.12. Git and GitHub terminology

Fur

GitHu
have

We also discussed some of the terminology for git. We will also come back to these ideas in greater detail later.

1.12.1. Programming is Collaborative

There are two very common types of collaboration

- code review (working 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.

Skip to main content

these collaborative aspects.

Important

Remember to fill out the team formation survey

1.12.2. Class forum

This community repository "assignment" will add you to a "team" with the whole class. It allows us to share things on GitHub, for the whole class, but not the whole internet.

Important

When you click that link join the existing team, do not make a new one

1.12.3. Get Credit for Today's class

1. Run your Experience Reflection (inclass) action on your kwl repo
2. Complete the file.

1.13. Prepare for next class/lab

1. review notes after they are posted, both rendered and the raw markdown include links to each in your badge PR
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 in a file called prior-knowledge-map. (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later

1.14. Review

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart (on a branch for this badge).
3. review git and github vocabulary (include link in your badge PR)
4. Post an introduction to your classmates on our discussion forum

1.15. Practice

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart (on a branch for this badge).
3. review git and github vocabulary be sure to edit a file and make an issue or PR (include link in your badge PR)
4. Post an introduction to your classmates on our discussion forum

2. Course Logistics and Learning

“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.

Most of us have had a bug, where we found a solution to get by, without really understanding **why** the solution fixed it or why that bug happened.

Debugging often requires understanding, in practice, of how the programming language works, how it translates that to hardware, and how the hardware works.

the first “bug” was an **actual** moth

These programmers had to know how to take apart the physical computer in order to find the insect.

our computers are a lot different, but we still need systems understanding to be efficient.

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.

This is a 300 level course, so you are one step closer to being a *professional* instead of **only** a *student*.

This is still a college course, so we will be taking time to understand the theory and the *why* not only the *what* that a bootcamp or on the job training might provide.

However, instead of using a text book that is designed explicitly and primarily for a school context, we will use **primary sources**.

In our context, that means using three main types of sources:

- official reference docs
- direct research results
- first hand accounts by professional developers

Back to what a system is …

From ACM Transactions on 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.

We are going to be studying aspects *of* computer systems, but to really understand them, we also have to think about 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.

Skip to main content

- 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.

Important

We will practice and reinforce things a lot

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.

2.3. Why do we need this for computer systems?

2.3.1. Systems are designed by programmers

Warning

this section is a little different than what I said in class, but it is still important and related.

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
were often
creating a
making a r
creating a
places, CS
or out of E
grew out o

2.3.2. 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.

2.3.3. 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.

For finding and reading this section, add a link to the heading above `Systems are designed by programmers` to your `community_contributions.md` in your KWL repo on a new branch, title the PR “Community Badge- Careful Reading” and request

Skip to main content

this will count for one community badge!

2.4. Let’s get organized

For class you should have a folder on your computer where you will keep all of your materials.

Open a terminal window. I am going to use `bash` commands

- if you are on mac, your default shell is `zsh` which is mostly the same as bash for casual use. you can switch to bash to make your output more like mine using the command `bash` if you want, but it is not required.
- if you are on windows, your **GitBash** terminal will be the least setup work to use `bash`
- if you have WSL (if you do not, no need to worry) you should be able to set your linux shell to `bash`

The first command we will use is `pwd` which stands for **p**rint **w**orking **d**irectory.

```
pwd
```

```
/Users/brownsarahm
```

this is called the path and specifically this is an absolute path

Not

these

We can change into another directory with `cd` for **c**hange **d**irectory

```
cd Documents/
```

To see what changed, we use `pwd` again

```
pwd
```

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

Note that the current path is the same as the old one plus the place we changed to.

I moved one step further into my inclass folder

```
cd inclass/
```

We can **mak** a new **dir**ectory with `mkdir`

```
mkdir systems
```

What you want to have is a folder for class (mine is systems) in a place you can find it. (mine is in my inclass folder)

We can view what is in a folder with `ls` for **list**

Skip to main content

I have a few other folders here

```
fa22            prog4dsfa23     sp23            systems
```

And again check the path

```
pwd
```

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

then I can do into the new folder I just made

```
cd systems/
```

and look at the path one more time

```
pwd
```

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

We can change back to the home directory with `~`

```
cd ~
```

and confirm:

```
pwd
```

```
/Users/brownsarahm
```

just like before

Then we can use the relative path` of where we want to go:

```
cd Documents/inclass/systems/
```

`cd` with no path also works

```
cd
```

and we can use `pwd` to see where we end up

Skip to main content

```
/Users/brownsarahm
```

it’s the home directoy just like `~`

```
cd Documents/inclass/systems/
```

`..` stands for up one level,

```
cd ../
```

```
pwd
```

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

2.5. Prepare for Next Class

1. Find the glossary page for the course website, link it below. Review the terms for the next class: shell, terminal, bash, git, GitHub.
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.
3. Make sure you have a working environment, see the list in the syllabus, including `gh` CLI if you use mac`. Use the discussions to ask for help.
4. Sign up for announcements

2.6. Review today’s class

1. review notes after they are posted, both rendered and the raw markdown include links to each in your badge PR
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 in a file called prior-knowledge-map. (optional) try mapping out using mermaid syntax, we’ll be using other tools that will faciltate rendering later

2.7. More Practice

1. review notes after they are posted, both rendered and the raw markdown versions. Include links to both views in your badge PR comment.
2. 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.
3. map out your computing knowledge and add it to your kwl chart repo in a file called prior-knowledge-map.md. Use mermaid syntax, to draw your map. GitHub can render it for you including while you work using the preview button.
4. Read more about version control in general and add a “version control” row to your KWL chart with all 3 columns filled in.

Skip to main content

In office hours, show us that you have a folder to work in for class.

2.9. Questions After Today’s Class

2.9.1. I wonder why Mac and Linux have built-in terminals but Windows, arguably the most popular OS, does not have a built-in terminal?

It does ahve a built in terminal, it just uses a Windows-only shell, `bash` is the most popular shell to learn, because it is used on unix and linux systems and those are most commonly used for **developers** and code **production** environments where you may only have remote, terminal only access.

2.9.2. No real questions, maybe curious about downloading github files using commands

that is what we will do Thursday

2.9.3. I’ve seen this type of material before so I would want to know about from other things I’ve seen would be how to interact with apps (google chrome) and stuff like that if that’s even possible?

one step of the action that builds the course syllabus pdf launches chromium and prints the syllabus to a pdf.

2.9.4. Will we be piping things in the command line often, or are we mostly getting an introduction to the concept?

we will be workign with the shell A LOT

2.9.5. I want to know more about what reverting does to github

In a few classes, we will get there!

2.9.6. does making directories in a shell put them in the directory automatically?

it puts them where you say, nothing explicit, is an implicit relative path to your current working directory.

2.9.7. Are there any benefits to choosing either zsh or bash? Ease of use, compatability, or anything else

`bash` is more standard `zsh` can do some things a little faster

Learning more about the differences is an option for an explore badge

2.9.8. why does cd and cd~ do the same thing?

`~` is a shortcut for home and so is nothing.

2.9.9. I want to know more about how git is connected and will be applied to github and other tools we use.

we will do this over the next few classes

2.9.10. how do I make up lectures

Read the notes and complete the experience report (makeup) action

2.9.11. Should I merge the add my name to readme pull request?

yes

2.9.12. if my lab was approved can i merge the pull request~

yes

2.9.13. in what cases does the terminal file explorer get used as opposed to just navigating the file explorer/finder?

Once you know the terminal, it becomes faster than the GUI. Also, if you need to programmatically move files, bash allows you to make scripts!

2.9.14. I think I’m still just a little confused on badges, and if they’re basically this class’s version of assignments, or something completely different.

they’re roughly like assignments, it’s how we track what work you have completed. Not *all* assigned badges are required though, no matter what grade you want to earn.

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

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

3. How do I use git offline?

Use the same one we used in the last class

and move to the folder we made to use as a working directory for in class

```
cd Documents/inclass/systems/
```

Note

we did some review, see the last notes instead of recapping that here

`..` is a special file that points to a specific relative path, of one level up.

we can use `cd` and `pwd` to illustrate what “one level up” means

```
pwd
```

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

```
cd ..
```

```
pwd
```

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

we move to make the current working directory the one, one step prior in the hierarchy with `cd ..`

```
cd systems/
```

3.2. A toy repo for in class

Warning

I removed the link from the public notes, but you can get it in prismia

this repo will be for *in class* work, you will not get feedback inside of it, unless you ask, but you will answer questions in your kwl repo about what we do in this repo sometimes

only work in this repo during class time or making up class, unless specifically instructed to (will happen once in a few weeks)

3.3. Connecting with GitHub

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

we want option 1 beacuse we are learning git

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. ssh keys (we will do this later)
2. `gh` CLI / gitscm in GitBash through browser

3.3.2. Windows (gitbash)

- `git clone` and paste your URL from GitHub
- then follow the prompts, choosing to authenticate in Browser.

3.3.3. MacOS X

- GitHub CLI: enter `gh auth login` and follow the prompts.
- then `git clone` and paste your URL from github

3.3.4. If nothing else works

Create a 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).

Then proceed to the clone step. You may need to configure an identity later with **`git config`**

3.3.5. Cloning a repo

```
git clone https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
```

then we get several messages back from git and GitHub (the remote)

```
Cloning into 'github-inclass-fa23-brownsarahm'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 8 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
```

Confirm it worked with:

Skip to main content

```
github-inclass-fa23-brownsarahm
```

We see the new folder that matches our repo name

3.4. What is in a repo?

We can enter that folder

```
cd github-inclass-fa23-brownsarahm/
```

When we compare the local directory to GitHub

```
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.

We can actually see the rest with the `-a` for **all** *option* or *flag*. Options are how we can pass non required parameters to command line programs.

```
ls -a
```

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

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

3.5. How do I know what git knows?

`git status` is your friend.

```
git satus
```

```
git: 'satus' is not a git command. See 'git --help'.

The most similar command is
        status
```

git give you hits, when you try to use a command that doesn’t exist. It is friendly, not scary!

```
git status
```

Skip to main content

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

nothing to commit, working tree clean
```

this command compares your working directory (what you can see with `ls -a` and all subfolders except the `.git` directorty) to the current state of your `.git` directory.

this tells us:

- the branch we are on (`On branch main`)
- that we have incorporated all changes downloaded from GitHub (`up to date with 'origin/main'`)
- that our working directory matches what it was after the repo's last commit (`nothing to commit, working tree clean`)

3.6. Making a branch with GitHub.

Note

Run the one action in the github in class repo one time to get the issues

First on an issue, create a branch using the link in the development section of the right side panel. See the github docs for how to do that.

Then it gives you two steps to do. We are going to do them one at a time so we can see better what they each do.

```
git fetch orign
```

```
fatal: 'orign' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Since the argument after `git fetch` is the variable anme of a remote, when I splled `origin` wrong, it says that it can't find it.

First we will update the `.git` directory without changing the working directory using git fetch. We have to tell git fetch where to get the data from, we do that using a name of a remote.

```
git fetch origin
```

```
From https://github.com/introcompsys/github-inclass-fa23-brownsarahm
 * [new branch]      1-create-an-about-file -> origin/1-create-an-about-file
```

then we check status again

```
git status
```

Skip to main content

```
nothing to commit, working tree clean
```

Looks like nothing so far.

Next, we switch to that branch.

```
git checkout 1-create-an-about-file
```

```
branch '1-create-an-about-file' set up to track 'origin/1-create-an-about-file'.
Switched to a new branch '1-create-an-about-file'
```

and verify what happened

```
git status
```

```
On branch 1-create-an-about-file
Your branch is up to date with 'origin/1-create-an-about-file'.

nothing to commit, working tree clean
```

3.7. Creating a file on the terminal

The `touch` command creates an empty file.

```
touch about.md
```

We can use `ls` to see our working directory now.

```
ls
```

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

```
git status
```

```
On branch 1-create-an-about-file
Your branch is up to date with 'origin/1-create-an-about-file'.

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)
```

Now we see something new. Git tells us that there is a file in the working directory that it has not been told to track the changes in and it knows nothing.

made more than one type of change, there would be multiple subheadings each with their own suggestions.

The very last line is advice of what do to overall.

We're going to do a bit more work first though, by adding conent to the file.

We are going to use the nano text editor to edit the file

```
nano about.md
```

On the nano editor the `^` stands for control.

and we can look at the contents of it.

```
cat about.md
```

```
Sarah Brown
2027
```

Now we will check again

```
git status
```

```
On branch 1-create-an-about-file
Your branch is up to date with 'origin/1-create-an-about-file'.

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)
```

In this case both say to `git add` to track or to include in what will be committed. Under untracked files is it says `git add <file>...`, in our case this would look like `git add about.md`. However, remember we learned that the `.` that is always in every directory is a special "file" that points to the current directory, so we can use that to add **all** files. Since we have only one, the two are equivalent, and the `.` is a common shortcut, because most of the time we want to add everything we have recently worked on in a single commit.

`git add` puts a file in the "staging area" we can use the staging area to group files together and put changes to multiple files in a single commit. This is something we **cannot** do on GitHub in the browser, in order to save changes at all, we have to commit. Offline, we can save changes to our computer without commiting at all, and we can group many changes into a single commit.

```
git add .
```

We will use status to see what has changed.

```
git status
```

Skip to main content

```
Your branch is up to date with 'origin/1-create-an-about-file'.

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

Now that one file is marked as a new file and it is in the group “to be committed”. Git also tells us how to undo the thing we just did.

Try this yourself

Try making a change, adding it, then restoring it. Use git status to see what happens at each point

Next, we will commit the file. We use `git commit` for this. the `-m` option allows us to put our commit message dirctly on the line when we commit. Notice that unlike commiting on GitHub, we do not choose our branch with the `git commit` command. We have to be “on” that branch before the `git commit`.

```
git commit -m 'create and complete about file closes #1'
```

```
[1-create-an-about-file 693a2b5] create and complete about file closes #1
 1 file changed, 2 insertions(+)
 create mode 100644 about.md
```

Warning

At this point you might get an error or warning about your identity. Follow what git says to either set or update your identity using `git config

```
ls
```

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

Remember, the messages that git gives you are designed to try to help you. The developers of git know it’s a complex and powerful tool and that it’s hard to remember every little bit.

We again check in with git:

```
git status
```

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

nothing to commit, working tree clean
```

Now it tells us we have changes that GitHub does not know about.

We can send them to github with `git push`

Imp

If you
into
to ent
`appea`
`termi`
`commi`
escap
`type`
write(

Skip to main content

```
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), 347 bytes | 347.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
   6a12db0..693a2b5  1-create-an-about-file -> 1-create-an-about-file
```

This tells us the steps git took to send:

- counts up what is there
- compresses them
- sends them to GitHub
- moves the `2-create-an-about-file` branch on GitHub from commit `3f54148` to commit `57de0cd`
- links the local `2-create-an-about-file` branch to the GitHub `2-create-an-about-file` branch

3.8. Review today's class

Any steps in a badge marked **lab** are steps that we are going to focus in on during lab time. Remember the goal of lab is to help you complete the work, not add additional work. The lab checkout will include some other tasks and then we will encourage you to work on this badge while we are there to help. Lab checkouts are checked only for completion though, not correctness, so steps of activities that we want you to really think about and revise if incorrect will be in a practice or review badge.

1. Read the notes. If you have any questions, post an issue on the course website repo.
2. Using your terminal, download your KWL repo. Include the command used in your badge PR.
3. Try using setting up git using your favorite IDE or GitHub Desktop. Make a file gitoffline.md and include some notes of how it went. Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent or does it use different terms?
4. **lab** 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 a file called gitcommit.md. Compare what happens based on what you can see on GitHub and what you can see with git status.

3.9. More Practice

Any steps in a badge marked **lab** are steps that we are going to focus in on during lab time. Remember the goal of lab is to help you complete the work, not add additional work. The lab checkout will include some other tasks and then we will encourage you to work on this badge while we are there to help. Lab checkouts are checked only for completion though, not correctness, so steps of activities that we want you to really think about and revise if incorrect will be in a practice or review badge.

1. Read the notes. If you have any questions, post an issue on the course website repo.
2. Using your terminal, download your KWL repo. Include the command used in your badge PR.
3. Try using setting up git using your favorite IDE or GitHub Desktop. Make a file gitoffline.md and include some notes of how it went. Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent or does it use different terms?
4. **lab** 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 a file called gitcommit_tips.md. Compare what happens based on

make mistakes with git add and commit and what to look for to get unstuck.

3.10. Prepare for Next Class

1. Reply below with any questions you have about using terminals so that you can bring it up in class
2. Be prepared to compare and contrast bash, shell, terminal, and git.
3. (optional) If you like to read about things before you do them, read about merge conflicts. If you prefer to see them first, read this after.

3.11. Experience Report Evidence

link to your github inclass repo

3.12. Questions After Today's Class

3.12.1. Are the things we learned in the terminal available to revisit later on?

Yes, in these notest!

3.12.2. what's the difference between git checkout and git pull, i used pull in 305 a lot

`git pull` updates the **same** branch you are on. It calls `git fetch` and then does some extra stuff. `git checkout` switches branches

3.12.3. What situations would you use the terminal over something like VScode for?

really quick edits where the time spent to open and configure VSCode does not pay off

3.12.4. Will we learn how to fix mistakes in the terminal? As in commands we did not mean to run

Yes, for the things that can be fixed, we will also see what commands cannot be undone and what safeties are in built into CLI tools.

3.12.5. Im curious about the benefits to viewing files in the terminal as opposed to quick viewing them in finder, or other things that seem to be easier outside of terminal

Viewing a file in finder when we know it is really small and only text requires **context switching** which makes it more work. Using a GUI can be an easier way to start, it is less to remember, but once you know how things work you cannot make them any faster, you have to move your mouse so far, wait for things to load, etc. The terminal lets you gain speed once you learn.

You can also automate and script things.

Skip to main content

4. How do git branches work?

4.1. Review

Recall, We can move around and examine the computer’s file structure using shell commands.

```
cd Documents/inclass/systems/github-inclass-fa23-brownsarahm/
```

To confirm our current working directory we print it with `pwd`

```
pwd
```

```
/Users/brownsarahm/Documents/inclass/systems/github-inclass-fa23-brownsarahm
```

this confirms what we expect

Now let’s see what is in our folder:

```
ls
```

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

and check in with git

```
git status
```

```
On branch 1-create-an-about-file
Your branch is up to date with 'origin/1-create-an-about-file'.

nothing to commit, working tree clean
```

this is as we left off.

4.2. Branches do not sync automatically

Now we will look in GitHub and see what is there.

Warning

the step below requires that you have the `gh` CLI. On windows with GitBash, you have not needed this, and it can be bit tricky to install. If you get it done and working well, and you submit instructions to this repo about how to get it set up, for either a community badge (if it’s simple) or an explore badge if it takes a lot

```
gh repo view --web
```

```
Opening github.com/introcompsys/github-inclass-fa23-brownsarahm in your browser.
```

On Github, we see that it matches the branch we were on because we had merged in our PR in class on Thursday.

```
git status
```

```
On branch 1-create-an-about-file
Your branch is up to date with 'origin/1-create-an-about-file'.

nothing to commit, working tree clean
```

Back on our local computer, we will go back to the main branch, using `git checkout`

```
git checkout main
```

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

now we look at the status of our repo:

```
ls
```

```
README.md
```

the file is missing. It said it was up to date with origin main, but that is the most recent time we checked github only. It's up to date with our local record of what is on GitHub, not the current GitHub.

Next, we will update locally, with `git fetch`

```
git fetch
```

```
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 646 bytes | 215.00 KiB/s, done.
From https://github.com/introcompsys/github-inclass-fa23-brownsarahm
   6a12db0..caeacb5  main       -> origin/main
```

Here we see 2 sets of messages. Some lines start with "remote" and other lines do not. The "remote" lines are what `git` on the GitHub server said in response to our request and the other lines are what `git` on your local computer said.

So, here, it counted up the content, and then sent it on GitHub's side. On the local side, it unpacked (remember git compressed the content before we sent it). It describes the changes that were made on the GitHub side, the main branch was moved from one commit to another. So it then updates the local main branch accordingly ("Updating 6a12db0…caeacb5").

```
ls
```

```
README.md
```

no changes yet. `fetch` updates the .git directory so that git knows more, but does not update our local file system.

We can use git pull to fully update.

```
git pull
```

```
Updating 6a12db0..caeacb5
Fast-forward
 about.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 about.md
```

```
ls
```

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

```
git status
```

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

nothing to commit, working tree clean
```

Now we will follow the instruction sin the README issue, commit to a new branch with `closes #x` where x is the issue number in the comment and make PR.

Merge the PR and then the issue closes.

4.3. Pull will downlaod and update

We will `git pull` again now all at once, this will do the work that fetch did before and the checkout.

```
git pull
```

```
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 1.32 KiB | 337.00 KiB/s, done.
From https://github.com/introcompsys/github-inclass-fa23-brownsarahm
   caeacb5..5c8aaa9  main       -> origin/main
 * [new branch]      add-name   -> origin/add-name
 1 file changed, 4 insertions(+)
```

We can see commits with `git log`

```
git log
```

```
commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (HEAD -> main, origin/main, origin/HEAD)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:46:00 2023 -0400

    closes #2

commit caeacb503cf4776f075b848f0faff535671f2887
Merge: 6a12db0 693a2b5
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:38:54 2023 -0400

    Merge pull request #4 from introcompsys/1-create-an-about-file

    create and complete about file closes #1

commit 693a2b5b9ad4c27eb3b50571b3c93dde353320a1 (origin/1-create-an-about-file, 1-create-an-about-file)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:36:03 2023 -0400

    create and complete about file closes #1

commit 6a12db0035e7c73772f7b2348b80dd0bfb3a2a2e
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Thu Sep 14 16:43:07 2023 +0000

    Add online IDE url

commit cfe32e5066921ad876d8a2c74b1fcb00c99b1cc7
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Thu Sep 14 16:43:05 2023 +0000

    Initial commit
```

this is a program, we can use enter/down arrow to move through it and then `q` to exit.

```
git status
```

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

nothing to commit, working tree clean
```

```
git pull
```

4.4. making a new branch locally

We've used `git checkout` to switch branchs before. To also create a branch at the same time, we use the `-b` option.

```
git checkout -b fun_fact
```

```
Switched to a new branch 'fun_fact'
```

```
git log
```

Note

notice where each branch pointer is

```
commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (HEAD -> fun_fact, origin/main, origin/HEAD, main)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:46:00 2023 -0400

    closes #2

commit caeacb503cf4776f075b848f0faff535671f2887
Merge: 6a12db0 693a2b5
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:38:54 2023 -0400

    Merge pull request #4 from introcompsys/1-create-an-about-file

    create and complete about file closes #1

commit 693a2b5b9ad4c27eb3b50571b3c93dde353320a1 (origin/1-create-an-about-file, 1-create-an-about-file)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:36:03 2023 -0400

    create and complete about file closes #1

commit 6a12db0035e7c73772f7b2348b80dd0bfb3a2a2e
```

we used the nano text editor. `nano` is simpler than other text editors that tend to be more popular among experts, `vim` and `emacs`. Getting comfortable with nano will get you used to the ideas, without putting as much burden on your memory. This will set you up to learn those later, if you need a more powerful terminal text editor.

```
nano about.md
```

Skip to main content

type into the file right away.

Add any fun fact on the line below your content… Then, write out (save), it will prompt the file name. Since we opened nano with a file name (`about.md`) specified, you will not need to type a new name, but to confirm it, by pressing enter/return.

My file looks like this.

```
Sarah Brown
2027
- i skied competitively in high school
```

```
git status
```

```
On branch fun_fact
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")
```

This is very similar to when we checked after creating the file before, but, notice a few things are different.

- the first line tells us the branch but does not compare to origin. (this branch does not have a linked branch on GitHub)
- thefile is listed as “not staged” instead of untracked
- it gives us the choice to add it to then commit OR to restore it to undo the changes

We will add it to stage

```
git add about.md
```

```
git status
```

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

and commit:

Warning

here i gave an example of what to do if you commit without a message. This is not recommended, but important to know how to fix.

```
git commit
```

Skip to main content

message and/or uncomment the template.

When you are done use `escape` to go back to command mode, the ~insert~ at the bottom of the screen will go away. Then type `:wq` and press enter/return.

What this is doing is adding a temporary file with the commit message that git can use to compelte your commit.

```
[fun_fact 6d4dbd3] add fun fact
 1 file changed, 1 insertion(+)
```

Now we can look again at where our branch pointers are.

```
git log
```

fun_fact moved up to the new commit, but the other branches stayed behind.

```
commit 6d4dbd33860fceb9c87bd3c4509deff8cecb3f45 (HEAD -> fun_fact)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:06:54 2023 -0400

    add fun fact

commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (origin/main, origin/HEAD, main)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:46:00 2023 -0400

    closes #2

commit caeacb503cf4776f075b848f0faff535671f2887
Merge: 6a12db0 693a2b5
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:38:54 2023 -0400

    Merge pull request #4 from introcompsys/1-create-an-about-file

    create and complete about file closes #1

commit 693a2b5b9ad4c27eb3b50571b3c93dde353320a1 (origin/1-create-an-about-file, 1-create-an-about-file)
```

Now we will push to github.

```
git push
```

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

    git push --set-upstream origin fun_fact

To have this happen automatically for branches without a tracking
```

Skip to main content

It cannot push, because it does not know where to push, like we noted above that it did not comapre to origin, that was beecause it does not have an “upstream branch” or a corresponding branch on a remote server. It does not work at first because this branch does not have a remote.

git stores its list of remotes in a `.git/config` file

```
cat .git/config
```

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
        precomposeunicode = true
[remote "origin"]
        url = https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
[branch "1-create-an-about-file"]
        remote = origin
        merge = refs/heads/1-create-an-about-file
```

we can see our other branches have `remote = origin` but our new branch is not there.

To fix it, we do as git said.

```
git push --set-upstream origin fun_fact
```

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 311 bytes | 311.00 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.
remote: 
remote: Create a pull request for 'fun_fact' on GitHub by visiting:
remote:      https://github.com/introcompsys/github-inclass-fa23-brownsarahm/pull/new/fun_fact
remote: 
To https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
 * [new branch]      fun_fact -> fun_fact
branch 'fun_fact' set up to track 'origin/fun_fact'.
```

This time the returned message from the remote includes the link to the page to make a PR. We are not goign to make the PR though

4.5. Merge conflicts

First, in your browser edit the about.md file to have a second fun fact.

```
nano about.md
```

```
cat about.md
```

```
Sarah Brown
2027
- i skied competitively in high school
- I went to Northeastern
```

now that we have edited in both places let’s pull.

```
git pull
```

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 697 bytes | 174.00 KiB/s, done.
From https://github.com/introcompsys/github-inclass-fa23-brownsarahm
   6d4dbd3..768dec8  fun_fact   -> origin/fun_fact
Updating 6d4dbd3..768dec8
error: Your local changes to the following files would be overwritten by merge:
        about.md
Please commit your changes or stash them before you merge.
Aborting
```

First it does not work because we have not committed.

This is helpful beacause it prevents us from losing any work.

```
git status
```

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

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")
```

we can add and commit at the same time using the `-a` option from the `git commit``

```
git commit -a -m 'second fun fact'
```

```
[fun_fact b378bd1] second fun fact
 1 file changed, 1 insertion(+)
```

Now we try to pull again

```
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.
```

Now it cannot work because the branches have diverged. This illustrates the fact that our two versions of the branch `fun_fact` and `origin/fun_fact` are two separate things.

gitGraph commit commit branch fun_fact checkout fun_fact commit branch origin/fun_fact checkout origin/fun_fact commit checkout fun_fact commit
the branches have diverged means that they do not agree and that they each have at least one commit that is different from the other.

```
git log
```

```
commit b378bd148e53dfa7195c58123362e40ae12ef3e7 (HEAD -> fun_fact)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:26:20 2023 -0400

    second fun fact

commit 6d4dbd33860fceb9c87bd3c4509deff8cecb3f45
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:06:54 2023 -0400

    add fun fact

commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (origin/main, origin/HEAD, main)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:46:00 2023 -0400

    closes #2

commit caeacb503cf4776f075b848f0faff535671f2887
Merge: 6a12db0 693a2b5
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:38:54 2023 -0400
```

git gave us some options, we will use rebase which will apply our local commits *after* the remote commits.

```
git pull --rebase
```

Skip to main content

```
Auto-merging about.md
CONFLICT (content): Merge conflict in about.md
error: could not apply b378bd1... second fun fact
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 b378bd1... second fun fact
```

Now it tells us there is a conflict.

```
nano about.md
```

We see that git marked up the conflict for us by editing the file:

```
Sarah Brown
2027
- i skied competitively in high school
<<<<<<<<< HEAD
- i started at URI in 2020
=======
- I went to Northeastern
>>>>>>>>> "add fun fact"
```

We manually edit it to be what we want it to be. We can take one change the other or both. We will choose both, so my file looks like this in the end.

```
cat about.md
```

```
Sarah Brown
2027
- i skied competitively in high school
- i started at URI in 2020
- I went to Northeastern
```

Now, we do git add, because above it said that was the next step.

```
git add about.md
```

and then this was the next thing after that.

Warning

If you got somethign very different at this point, run `git --version` mine is 2.41. If yours is different, let me know as an issue.

```
git rebase --continue
```

```
[detached HEAD 756c487] second fun fact
 1 file changed, 2 insertions(+)
```

Skip to main content

Once we rebase and everything is done, we can push.

```
git push
```

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
   768dec8..756c487  fun_fact -> fun_fact
```

4.6. Prepare for Next Class

1. Read the notes from 2023-09-19 carefully
2. Examine an open source project on GitHub. Answer the reflection questions below in `software.md` in your kwl repo on a branch that is linked to this issue. You do not need to make the PR, we will work with this in class on 2023-09-21.

```
## Software Reflection

1. Link and title of the 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 languages?
1. Are there files that are configurations or settings?
1. Is there help for developers? users? which support is better?
1. What type of things are in the hidden files? who would need to see those files vs not?
```

Some open source projects if you do not have one in mind:

- pandas
- numpy
- GitHub CLI
- Rust language
- vs code
- Typescript
- Swift
- Jupyter book

4.7. Review today's class

1. Create a merge conflict in your github in class repo and resolve it using your favorite IDE,. Describe how you created it, show the files, and describe how your IDE helps or does not help in ide_merge_conflict.md. Give advice for when you think someone should resolve a merge conflict manually vs using an IDE. (if you do not regulary use an, IDE, try VSCode)
2. Read more details about git branches(you can also use other resources) add `branches.md` to your KWL repo and describe how branches work, in your own words. Include one question you have about branches or one scenario you think they could

Skip to main content

1. Create a merge conflict in your github in class repo and resolve it using your favorite IDE, then create one and resolve it on GitHub in browser(this requires the merge conflict to occur on a PR). Describe how you created it, show the files, and describe how your IDE helps or does not help in ide_merge_conflict.md. Give advice for when you think someone should resolve a merge conflict in GitHub vs using an IDE. (if you do not regulary use an, IDE, try VSCode)
2. Learn about GitHub forks and more about git branches(you can also use other resources)
3. add `branches-forks.md` to your KWL repo and describe how branches work, what a fork is, and how they relate to one another. If you use other resources, include them in your file as markdown links.

4.9. Experience Report Evidence

Note

this is for if you miss class and need to make it up

run `git log >> inclass2023-09-19.md` and then copy that new file into your KWL repo on the branch for your experience report

4.10. Questions After Today's Class

4.10.1. How would you fix a merge conflict involving other people trying to commit on other systems?

The same way we did in class.

4.10.2. are there situations where if you accidently make another branch, that you can merge it can into the one you wanted to be working on?

yes, exactly!

4.10.3. could you fix a merge conflict in github rather than in the local terminal in some cases?

Yes, when the merge conflict arises as a result of a pull request and it is simple you can resolve it in github.com

4.10.4. What exactly does the rebase command do?

I described it in more detail above with links to git docs.

4.10.5. What point does using git locally give?

it's the only way to keep your work in git and also run code locally

Not sure, but it might be the version of git.

4.10.7. What website do you find the most useful to resolve issues relating to git within the terminal?

I tend to rely on using `git status` to see where it is at, thinking about where I want to get and then searching and using eg stackoverflow for things I do not know how to do.

that said git gud can help you visualize and we will have some other visualizations soon.

4.10.8. Would it be reasonable to never use the local Terminal and just always use github?

No, that means you would not be able to run your code to work. Even if you work on most clous services, you still need to use the “local” terminal, on that cloud instance.

5. When do I get an advantage from git and bash?

so far we have used git and bash to accomplish familiar goals, and git and bash feel like just extra work for familiar goals.

Today, we will start to see why git and bash are essential skills: they give you efficiency gains and time traveling super powers (within your work only, sorry)

Further Reading

Use these for checking facts and resources.

- bash
- git

5.1. Setup

First, we’ll go back to our github inclass folder

```
cd Documents/inclass/systems/github-inclass-fa23-brownsarahm/
```

and we will make sure we are in sync with GitHub. We know we *can* fix it, but it is best to not break it.

```
git status
```

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

nothing to commit, working tree clean
```

Skip to main content

```
git pull
```

```
Already up to date.
```

Now let’s go back to the main branch.

```
git checkout main
```

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

It tells us we are there.

Let’s look at what commits are on the main branch.

```
git log
```

```
commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (HEAD -> main, origin/main, origin/HEAD)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:46:00 2023 -0400

    closes #2

commit caeacb503cf4776f075b848f0faff535671f2887
Merge: 6a12db0 693a2b5
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:38:54 2023 -0400

    Merge pull request #4 from introcompsys/1-create-an-about-file

    create and complete about file closes #1

commit 693a2b5b9ad4c27eb3b50571b3c93dde353320a1 (origin/1-create-an-about-file, 1-create-an-about-file)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Thu Sep 14 13:36:03 2023 -0400
```

We remember that we had added fun facts and none of our commits about those are here, so the fun_fact branch is still ahead of the main branch.

It’s something like this:

gitGraph commit commit commit commit branch fun_fact checkout fun_fact commit branch origin/fun_fact commit checkout fun_fact commit merge origin/fun_fact
We can merge branches locally, without making a pull request in GitHub.

Skip to main content

```
Updating 5c8aaa9..756c487
Fast-forward
 about.md | 4 ++++
 1 file changed, 4 insertions(+)
```

It summarizes the changes.

Now we can use `git log` to see where our branches all point.

```
git log
```

```
commit 756c4879c0447db20980f73a26bc2ba072e08a6d (HEAD -> main, origin/fun_fact, fun_fact)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:26:20 2023 -0400

    second fun fact

commit 768dec80c5e0734476d476ae83376c9c786b6450
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:21:31 2023 -0400

    Update about.md

commit 6d4dbd33860fceb9c87bd3c4509deff8cecb3f45
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 13:06:54 2023 -0400

    add fun fact

commit 5c8aaa9f2a129d551b8cb2cb294676f63c4af410 (origin/main, origin/HEAD)
Merge: caeacb5 65e9e39
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Sep 19 12:47:08 2023 -0400

    Merge pull request #5 from introcompsys/add-name

    closes #2

commit 65e9e39935be8400ef12cc9003592f12244b50da (origin/add-name)
```

Visually, it is like this

gitGraph commit id:"0" commit id:"1" commit id:"2" commit id:"3" branch fun_fact checkout fun_fact commit id:"A" branch origin/fun_fact commit id:"B" checkout fun_fact merge origin/fun_fact commit id:"C" checkout main merge fun_fact

```
git status
```

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

nothing to commit, working tree clean
```

It is ahead by 3 commits now because there were 3 commits on the fun_fact branch.

This
merm
often
idea,
every
(files,
can a
merm
them

Skip to main content

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

next we are going to pretend we worked on the project and made a bunch of files

```
touch abstract_base_class.py helper_functions.py important_classes.py alternative_classes.py README.md LICENS
```

New bash lessons:

- `touch` can accept a list of files to create
- a list in bash is space separated with no brackets

```
ls
```

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

```
git status
```

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
	API.md
	CONTRIBUTING.md
	LICENSE.md
	_config.yml
	_toc.yml
	abstract_base_class.py
	alternative_classes.py
	helper_functions.py
	important_classes.py
	overview.md
	setup.py
	test_alt.py
	test_help.py
	test_imp.py
	tests_abc.py

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

5.3. Echo and redirects

`cat` concatenates the contents of a file to stdout, which is a special file that our terminal reads

```
cat README.md
```

```
# GitHub Practice

Name: Sarah Brown

[![Open in Codespaces](https://classroom.github.com/assets/launch-codespace-7f7980b617ed060a017424585567c406b
```

`echo` allows us to send a message to stdout.

```
ehco "age=35"
```

```
-bash: ehco: command not found
```

For example, we can echo our age

```
echo "age=35"
```

```
age=35
```

We can also **redirect** the contents of a command from stdout to a file in `bash` like file operations while programming there is a similar concept to this mode.

There are two types of redirects, like there are two ways to write to a file, more generally:

- overwrite (`>`)
- append (`>>`)

```
echo "age=35" >> README.md
```

```
cat README.md
```

```
# GitHub Practice

Name: Sarah Brown

[![Open in Codespaces](https://classroom.github.com/assets/launch-codespace-7f7980b617ed060a017424585567c406b
age=35
```

Now we see the text at the bottom of the file.

5.4. commit and add has limitations

Now we have made some changes we want, so let's commit our changes.

Last class we saw we can add and commit at the same time

```
git commit -a -m 'start organizing'
```

Skip to main content

```
 1 file changed, 1 insertion(+)
```

This is way fewer things than we have done recently (remember we also made all of those files with touch)

lets check `git status` again

```
git status
```

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
	API.md
	CONTRIBUTING.md
	LICENSE.md
	_config.yml
	_toc.yml
	abstract_base_class.py
	alternative_classes.py
	helper_functions.py
	important_classes.py
	overview.md
	setup.py
	test_alt.py
	test_help.py
	test_imp.py
	tests_abc.py

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

The `-a` option on a commit does not add untracked files

We have to explicitly add those files.

```
git add .
```

Now we can commit the content.

```
git commit -m 'start organizng for real'
```

```
[main d76bc52] start organizng for real
 15 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 API.md
 create mode 100644 CONTRIBUTING.md
 create mode 100644 LICENSE.md
 create mode 100644 _config.yml
 create mode 100644 _toc.yml
 create mode 100644 abstract_base_class.py
 create mode 100644 alternative_classes.py
 create mode 100644 helper_functions.py
 create mode 100644 important_classes.py
 create mode 100644 overview.md
 create mode 100644 setup.py
 create mode 100644 test_alt.py
 create mode 100644 test_help.py
 create mode 100644 test_imp.py
 create mode 100644 tests_abc.py
```

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

5.5. Redirect file modes

Now, let's go back to thinking about redirects. We saw that with two `>>` we appended to the file. With just *one* what happens?

```
echo "age=35" > README.md
```

We check the file now

```
cat README.md
```

```
age=35
```

It wrote over. One `>` writes to the file in overwrite mode.

5.6. Recovering from mistakes with git restore

This would be bad, we lost content, but this is what git is for!

It is *very very* easy to undo work since our last commit.

This is good for times when you have something you have an idea and you do not know if it is going to work, so you make a commit before you try it. Then you can try it out. If it doesn't work you can undo and go back to the place where you made the commit.

To do this, we will first check in with git

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)

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

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

Notice that it tells us what to do `(use "git restore <file>..." to discard changes in working directory)`. The version of README.md that we broke is in the working directory but not commited to git, so git refers to them as "changes" in the workign

Tip

Conc

Wher
use fu

•
•

Ne

mo
that
glos

both requir
and typical
to open the

read

write

append

Reference

C language

Skip to main content

sense so t

```
git restore README.md
```

Now lets check with git.

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Our working tree is clean now.

and it looks like it did before the `>` line. and we can check the file too

```
cat README.md
```

```
# GitHub Practice

Name: Sarah Brown

[![Open in Codespaces](https://classroom.github.com/assets/launch-codespace-7f7980b617ed060a017424585567c406b
age=35
```

Back how we wanted it!

5.7. What are these files?

We can use a redirect to add a bunch of text to the README file.

```
 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:

Skip to main content

- code that defines a python module
- test code
- documentation
- extra files that “we know” we can delete.

We can see what it does with `cat`

```
cat README.md
```

```
# GitHub Practice

Name: Sarah Brown

[![Open in Codespaces](https://classroom.github.com/assets/launch-codespace-7f7980b617ed060a017424585567c406b
age=35
|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

We can see how the `"` allows us to go on multiple lines this way:

```
echo "sldkjfds

>
>
>
>
>
>
>
>
>
> "
```

and it will output all of the blank lines

Skip to main content

5.8. Getting organized

Recall, we have just created a bunch of files.

```
ls
```

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

First, we'll make a directory with `mkdir`

```
mkdir docs
```

we can use `ls` to see the files that exist

```
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
_config.yml             helper_functions.py     test_imp.py
_toc.yml                important_classes.py    tests_abc.py
```

5.9. Moving Files

next we will move a file there with `mv`

We can see how to use this by getting the error when we do not pass any arguments.

```
mv
```

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

Skip to main content

5.9.1. Help in GitBash

```
mv --help
```

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

This only give a basic version on mac, but is more on Git Bash

5.9.2. Getting help on *nix systems

`man` shows the manfile for help for a specific command

```
man mv
```

It opens in a program, then we need to use `q` to exit.

5.9.3. Back to moving.

```
mv overview.md docs/
```

what this does is change the path of the file from `.../github-inclass-brownsarahm-1/overview.md` to `.../github-inclass-brownsarahm-1/docs/overview.md`

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

```
ls
```

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

We can also use `ls` with a relative or absolute path of a directory to list tht location instead of our current working directory.

```
ls docs/
```

```
overview.md
```

5.9.4. Moving multiple files with patterns

```
ls
```

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

We have lots with similar names.

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

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

Again, we confirm it worked by seeing that they are no longer in the working directory.

```
ls
```

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

and that they *are* in `docs`

```
ls docs/
```

```
_config.yml     _toc.yml        overview.md
```

Next we will work on the test files

```
mkdir tests
```

5.10. Renaming Files

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
```

Skip to main content

```
tests/        tests_abc.py
```

Then I added `_` and pressed tab again to get it full file name.

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

This changes the path from `.../tests_abc.py` to `.../test_abc.py` to. It is doing the same thing as when we use it to move a file from one folder to another folder, but changing a differnt part of the path.

Not

this is
impor
orgar
can u
yours

```
ls
```

```
API.md                    alternative_classes.py  test_alt.py
CONTRIBUTING.md           docs                    test_help.py
LICENSE.md                helper_functions.py     test_imp.py
README.md                 important_classes.py    tests
about.md                  setup.py
abstract_base_class.py  test_abc.py
```

Then we can use a similar pattern to move the files.

```
mv test_*.py tests/
```

Now we can confirm that it is how we expect.

```
ls
```

```
API.md                    about.md                  helper_functions.py
CONTRIBUTING.md           abstract_base_class.py  important_classes.py
LICENSE.md                alternative_classes.py  setup.py
README.md                 docs                      tests
```

```
ls tests/
```

```
test_abc.py     test_alt.py     test_help.py     test_imp.py
```

5.11. Commiting Changes

we check where we are

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
        deleted:    _config.yml
        deleted:    _toc.yml
        deleted:    overview.md
        deleted:    test_alt.py
        deleted:    test_help.py
        deleted:    test_imp.py
        deleted:    tests_abc.py

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

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

```
git add docs/
```

```
git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   docs/_config.yml
        new file:   docs/_toc.yml
        new file:   docs/overview.md

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
        deleted:    _config.yml
        deleted:    _toc.yml
        deleted:    overview.md
        deleted:    test_alt.py
        deleted:    test_help.py
        deleted:    test_imp.py
        deleted:    tests_abc.py

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

```
git add .
```

```
git commit -m 'begin organizing
>
>
> '
```

```
[main 042a42e] begin organizing
 8 files changed, 19 insertions(+)
 rename _config.yml => docs/_config.yml (100%)
 rename test_alt.py => tests/test_abc.py (100%)
 rename test_help.py => tests/test_alt.py (100%)
 rename test_imp.py => tests/test_help.py (100%)
 rename tests_abc.py => tests/test_imp.py (100%)
```

ANd finally push

```
git push
```

```
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 1.72 KiB | 1.72 MiB/s, done.
Total 11 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/introcompsys/github-inclass-fa23-brownsarahm.git
   5c8aaa9..042a42e  main -> main
```

5.12. experience badge with a file

1. Run the action to create the branch for today's experience badge
2. Open a PR to propose changes from your prepare work brach *to* the branch for today's experience badge
3. merge that PR
4. See that in your experience badge, the prepare work is now visible
5. fill in your experience badge
6. request a review from the TA that is in your group

5.13. Prepare for Next Class

1. Read the notes from 2023-09-19 carefully
2. Examine an open source project on GitHub. Answer the reflection questions below in `software.md` in your kwl repo on a branch that is linked to this issue. You do not need to make the PR, we will work with this in class on 2023-09-21.

```
## Software Reflection

1. Link and title of the 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 languages?
1. Are there files that are configurations or settings?
1. Is there help for developers? users? which support is better?
1. What type of things are in the hidden files? who would need to see those files vs not?
```

Some open source projects if you do not have one in mind:

- pandas
- numpy
- GitHub CLI
- Rust language

Skip to main content

- Typescript
- Swift
- Jupyter book

5.14. Review today's class

badge steps marked **lab** are steps that you will be encouraged to use lab time to work on.

1. Update your KWL chart with the new items and any learned items.
2. Clone the course website. Append the commands used and the contents of your `fall2023/.git/config` to a terminal_review.md (hint: history outputs recent commands and redirects can work with any command, not only echo). Edit the README.md, commit, and try to push the changes. What happens and what GitHub concept that we have not used yet might fix it? see your `vocab-` repo for a list of key github concepts. (answer in the `terminal_review.md`)
3. **lab** Organize the provided messy folder in a Codepsace (details will be provided in lab time). Commit and push the changes. Answer the questions below in your kwl repo in a file called `terminal_organization.md`
4. clone your `messy_repo` locally and append the `history.md` file to your `terminal_organization.md`
5. Find your team's repository. It will have a name like `fa23-team#` where `#` is a number 1-4. Join the discussion on that repo about naming your team. Link to your comment directly in your PR for this badge (use the 3 dots menu to get the comment specific URL).

```
# Terminal File moving reflection
1. How was this activity overall? Did this get easier toward the end?
2. When do you think that using the terminal will be better than using your GUI file explorer?
3. What questions/challenges/ reflections do you have after this exercise?

## Commands used
```

5.15. More Practice

badge steps marked **lab** are steps that you will be encouraged to use lab time to work on.

1. Update your KWL chart with the new items and any learned items.
2. Get set up so that you can pull from the introcompsyss/fall2023 repo and push to your own fork of the class website by cloning the main repo, then forking it and adding your fork as an additional remote. Append the commands used and the contents of your `fall2023/.git/config` to a terminalpractice.md (hint: `history` outputs recent commands and redirects can work with any command, not only echo). Based on what you know so far about forks and branches, what advantage does this setup provide? (answer in the `terminal_practice.md`)
3. **lab** Organize the provided messy folder (details will be provided in lab time). Commit and push the changes. Clone that repo locally.
4. For extra practice, re/organize 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 in a new file, terminal_organization_adv.md in your kwl repo. Tip: Start with a file explorer open, but then try to close it, and use only command line tools to explore and make your choices. If you get stuck, look up additional commands to do acomplish your goals.
5. Find your team's repository. It will have a name like `fa23-team#` where `#` is a number 1-4. Join the discussion on that repo about naming your team. Link to your comment directly in your PR for this badge (use the 3 dots menu to get the comment

Skip to main content

```
# Terminal File moving reflection
1. How was this activity overall Did this get easier toward the end?
2. How was it different working on your own computer compared to the Codespace from?
3. Did you have to look up how to do anything we had not done in class?
4. When do you think that using the terminal will be better than using your GUI file explorer?
5. What questions/challenges/ reflections do you have after this?
6. Append all of the commands you used in lab below. (not from your local computer's history, from the codesp
```

5.16. Experience Report Evidence

redirect the history to a file

```
history >> makeup_2023-09-21.md
```

then move the file created to your KWL repo on your experience report branch.

5.17. Questions After Today's Class

5.17.1. How can the "*" be used consistently to move a bunch of file? because we used it with the text infront of the similarity and behind it. i'm just wondering how it's used for multiple files, because it was used both before the similarity and after the similarity.

It fills in for any number of characters.

5.17.2. I have some badges from 2 weeks ago that are awaiting a second review after changes had been fixed.

Important

Do not merge an unapproved badge

Re-request a review

5.17.3. can you close pull requests on the terminal?

Not with git because pull requests are not a git feature, but it is a feature of github. The `gh` CLI can do this.

5.17.4. If you do mv * will it move all files?

In the current working directory.

5.17.5. Are there any other ways of using mv that haven't been covered yet?

Skip to main content

5.17.6. how often should we be practicing with the terminal for git?

Ideally, you work on badges on at least several of the days we do not have class so that you are working with it close to every day.

You could also start trying to use them for your other classes.

5.17.7. Would it be beneficial to organize files with github rather than bash?

GitHub cannot organize files and doing do in broswer would be slow and difficult. We will see that GitHub code spaces give us a virtual machine that we can work with.

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 c
on yo

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 | _ | _ | _ |
| ssh keys | _ | _ | _ |
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
|logic gates | _ | _ | _ |
| ALU | _ | _ | _ |
| binary operations | _ | _ | _ |
| memory | _ | _ | _ |
| cache | _ | _ | _ |
| register | _ | _ | _ |
| clock | _ | _ | _ |
| Concurrency | _ | _ | _ |
```

Required Files

This lists the files for reference, but mostly you can keep track by badge issue checklists.

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.

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.

Review Badges

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. Most days there will be specific additional activities and questions to answer. These should be in your KWL repo. Review activities will help you to reinforce what we do in class and guide you to practice with the most essential skills of this class.

2023-09-07

related notes

Activities:

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart (on a branch for this badge).
3. review git and github vocabulary (include link in your badge PR)
4. Post an introduction to your classmates on our discussion forum

2023-09-12

related notes

Activities:

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 in a file called prior-knowledge-map. (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later

2023-09-14

related notes

Activities:

Any steps in a badge marked **lab** are steps that we are going to focus in on during lab time. Remember the goal of lab is to help you complete the work, not add additional work. The lab checkout will include some other tasks and then we will encourage you to work on this badge while we are there to help. Lab checkouts are checked only for completion though, not correctness, so steps of activities that we want you to really think about and revise if incorrect will be in a practice or review badge.

1. Read the notes. If you have any questions, post an issue on the course website repo.
2. Using your terminal, download your KWL repo. Include the command used in your badge PR.
3. Try using setting up git using your favorite IDE or GitHub Desktop. Make a file gitoffline.md and include some notes of how it went. Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent or does it use different terms?
4. **lab** 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 a file called gitcommit.md. Compare what happens based on what you can see on GitHub and what you can see with git status.

2023-09-19

related notes

Activities:

1. Create a merge conflict in your github in class repo and resolve it using your favorite IDE,. Describe how you created it, show the files, and describe how your IDE helps or does not help in ide_merge_conflict.md. Give advice for when you think someone should resolve a merge conflict manually vs using an IDE. (if you do not regulary use an, IDE, try VSCode)
2. Read more details about git branches(you can also use other resources) add `branches.md` to your KWL repo and describe how branches work, in your own words. Include one question you have about branches or one scenario you think they could help you with.

2023-09-21

related notes

Activities:

badge steps marked **lab** are steps that you will be encouraged to use lab time to work on.

1. Update your KWL chart with the new items and any learned items.
2. Clone the course website. Append the commands used and the contents of your `fall2023/.git/config` to a terminal_review.md (hint: history outputs recent commands and redirects can work with any command, not only echo). Edit the

Skip to main content

fix it? see your `vocab-` repo for a list of key github concepts. (answer in the `terminal_review.md`)

3. **lab** Organize the provided messy folder in a Codepsace (details will be provided in lab time). Commit and push the changes. Answer the questions below in your kwl repo in a file called `terminal_organization.md`
4. clone your `messy_repo` locally and append the `history.md` file to your `terminal_organization.md`
5. Find your team's repository. It will have a name like `fa23-team#` where `#` is a number 1-4. Join the discussion on that repo about naming your team. Link to your comment directly in your PR for this badge (use the 3 dots menu to get the comment specific URL).

```
# Terminal File moving reflection
1. How was this activity overall? Did this get easier toward the end?
2. When do you think that using the terminal will be better than using your GUI file explorer?
3. What questions/challenges/ reflections do you have after this exercise?

## Commands used
```

2023-09-26

related notes

Activities:

1. create an issue on your group repo for a tip or cheatsheet item you want to contribute. Make sure that your contribution does not overlap with one that amemb
2. clone your group repo.
3. work offline and add your contribution and then open a PR
4. review a class mate's PR.
5. Export your git log for your KWL main branch to a file called `gitlog.txt` and commit that as exported to the branch for this issue. **note that you will need to work between two branchse to make this happen**. Append a blank line, `## Commands`, and another blank line to the file, then the command history used for this exercise to the end of the file.

2023-09-28

related notes

Activities:

1. Review the notes, jupyterbook docs, and experiment with the `jupyter-book` CLI to determine what files are required to make `jupyter-book build` run. Make your kwl repo into a jupyter book. Set it so that the `_build` directory is not under version control.
2. Add `docs.md` to your KWL repo and explain the most important things to know about documentation in your own words using other programming concepts you have learned so far. Include in a markdown (same as HTML `<!-- comment -->`) comment the list of CSC courses you have taken for context while we give you feedback.

Prepare for the next class

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.

The correct answer is not as important for these activities as it is to do them before class. We will build on these in class. These are evaluated on completion only, but we may ask you questions or leave comments if appropriate, in that event you should reply and then we'll approve.

2023-09-12

related notes

Activities:

1. (for lab) Read the syllabus section of the course website carefully and explore the whole course website
2. Bring questions about the course to lab
3. (for class) Think about one thing you've learned really well (computing or not). Be prepared to discuss the following: How do you know that you know it? What was it llike to first learn it?

2023-09-14

related notes

Activities:

1. Find the glossary page for the course website, link it below. Review the terms for the next class: shell, terminal, bash, git, GitHub.
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.
3. Make sure you have a working environment, see the list in the syllabus, including `gh` CLI if you use mac`. Use the discussions to ask for help.
4. Sign up for announcements

2023-09-19

related notes

Activities:

1. Reply below with any questions you have about using terminals so that you can bring it up in class
2. Be prepared to compare and contrast bash, shell, terminal, and git.
3. (optional) If you like to read about things before you do them, read about merge conflicts. If you prefer to see them first, read this after.

2023-09-21

related notes

Skip to main content

1. Read the notes from 2023-09-19 carefully
2. Examine an open source project on GitHub. Answer the reflection questions below in `software.md` in your kwl repo on a branch that is linked to this issue. You do not need to make the PR, we will work with this in class on 2023-09-21.

```
## Software Reflection

1. Link and title of the 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 languages?
1. Are there files that are configurations or settings?
1. Is there help for developers? users? which support is better?
1. What type of things are in the hidden files? who would need to see those files vs not?
```

Some open source projects if you do not have one in mind:

- pandas
- numpy
- GitHub CLI
- Rust language
- vs code
- Typescript
- Swift
- Jupyter book

2023-09-26

related notes

Activities:

1. Review your software.md file from last prepare
2. Review the notes from 2023-09-21
3. Bring git questions or scenarios you want to be able to solve to class on Thursday (in your mind or comment here if that helps you remember)
4. Update your `.github/workflows/experienceinclass.yml` file as to add another paramter to the last step (`Create Pull request`) `reviewers: <ta-gh-name>` where `<ta-gh-name>` is the github user name of the TA is in your group. You can see your group on the organiation teams page named like “Fall 2023 Group X”. Do this on a branch for this issue.

2023-09-28

related notes

Activities:

1. Read the 2023-09-26 notes when posted
2. 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`

Skip to main content

of how you think about the problem, how do you decide what to check? how do you decide what to try? what information do you look for?

More Practice Badges

Note

these are listed by the date they were *posted*

More practice exercises are a chance to try new dimensions of the concepts that we cover in class.

Note

Activities will appear here once the semester begins

2023-09-07

related notes

Activities:

1. Review the notes after I post them.
2. Fill in the first two columns of your KWL chart (on a branch for this badge).
3. review git and github vocabulary be sure to edit a file and make an issue or PR (include link in your badge PR)
4. Post an introduction to your classmates on our discussion forum

2023-09-12

related notes

Activities:

1. review notes after they are posted, both rendered and the raw markdown versions. Include links to both views in your badge PR comment.
2. 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.
3. map out your computing knowledge and add it to your kwl chart repo in a file called prior-knowledge-map.md. Use mermaid syntax, to draw your map. GitHub can render it for you including while you work using the preview button.
4. Read more about version control in general and add a “version control” row to your KWL chart with all 3 columns filled in.

2023-09-14

related notes

Any steps in a badge marked **lab** are steps that we are going to focus in on during lab time. Remember the goal of lab is to help you complete the work, not add additional work. The lab checkout will include some other tasks and then we will encourage you to work on this badge while we are there to help. Lab checkouts are checked only for completion though, not correctness, so steps of activities that we want you to really think about and revise if incorrect will be in a practice or review badge.

1. Read the notes. If you have any questions, post an issue on the course website repo.
2. Using your terminal, download your KWL repo. Include the command used in your badge PR.
3. Try using setting up git using your favorite IDE or GitHub Desktop. Make a file gitoffline.md and include some notes of how it went. Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent or does it use different terms?
4. **lab** 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 a file called gitcommit_tips.md. Compare what happens based on what you can see on GitHub and what you can see with git status. Write a scenario with examples of how a person might make mistakes with git add and commit and what to look for to get unstuck.

2023-09-19

related notes

Activities:

1. Create a merge conflict in your github in class repo and resolve it using your favorite IDE, then create one and resolve it on GitHub in browser(this requires the merge conflict to occur on a PR). Describe how you created it, show the files, and describe how your IDE helps or does not help in ide_merge_conflict.md. Give advice for when you think someone should resolve a merge conflict in GitHub vs using an IDE. (if you do not regulary use an, IDE, try VSCode)
2. Learn about GitHub forks and more about git branches(you can also use other resources)
3. add `branches-forks.md` to your KWL repo and describe how branches work, what a fork is, and how they relate to one another. If you use other resources, include them in your file as markdown links.

2023-09-21

related notes

Activities:

badge steps marked **lab** are steps that you will be encouraged to use lab time to work on.

1. Update your KWL chart with the new items and any learned items.
2. Get set up so that you can pull from the introcompsyss/fall2023 repo and push to your own fork of the class website by cloning the main repo, then forking it and adding your fork as an additional remote. Append the commands used and the contents of your `fall2023/.git/config` to a terminalpractice.md (hint: `history` outputs recent commands and redirects can work with any command, not only echo). Based on what you know so far about forks and branches, what advantage does this setup provide? (answer in the `terminal_practice.md`)
3. **lab** Organize the provided messy folder (details will be provided in lab time). Commit and push the changes. Clone that repo locally.
4. For extra practice, re/organize a folder on your computer (good candidate may be desktop or downloads folder), using only a

command line tools to explore and make your choices. If you get stuck, look up additional commands to do acomplish your goals.

5. Find your team's repository. It will have a name like `fa23-team#` where `#` is a number 1-4. Join the discussion on that repo about naming your team. Link to your comment directly in your PR for this badge (use the 3 dots menu to get the comment specific URL).

```
# Terminal File moving reflection
1. How was this activity overall Did this get easier toward the end?
2. How was it different working on your own computer compared to the Codespace from?
3. Did you have to look up how to do anything we had not done in class?
4. When do you think that using the terminal will be better than using your GUI file explorer?
5. What questions/challenges/ reflections do you have after this?
6. Append all of the commands you used in lab below. (not from your local computer's history, from the codes
```

2023-09-26

related notes

Activities:

1. Find a resource/reference that helps explain a topic related to the course that you want to review. Make sure that your contribution does not overlap with one that another member is going to post by viewing other issues before you post your issue. Create an issue for your planned item.
2. Clone your group repo.
3. Work offline to add your contribution and then open a PR. Your reference review should help a classmate decide if that reference material will help them understand better or not. It should summarize the material and it's strengths/weaknesses.
4. Complete a peer review of a class mate's PR. Use inline comments for any minor corrections, provide a summary, and either approve or request changes.
5. learn about options for how git can display commit history. Try out a few different options. Choose two, write them both to a file, `gitlog-compare.md`. Using a text editor, wrap each log with three backticks to make them "code blocks" and then add text to the file describing a use case where that format in particular would be helpful.

2023-09-28

related notes

Activities:

1. Review the notes, jupyterbook docs, and experiment with the `jupyter-book` CLI to determine what files are required to make `jupyter-book build` run. Make your kwl repo into a jupyter book. Set it so that the `_build` directory is not under version control.
2. Learn about the documentation ecosystem in another language that you know using at least one official source and additional sources as you find helpful. In `docs.md` include a summary of your findings and compare and contrast it to jupyter book/sphinx. Include a bibtex based bibliography of the sources you used. You can use this generator for informal sources and google scholar for formal sources.

Skip to main content

Explore Badges

Warning

Explore Badges are not required, but an option for higher grades. The logistics of this could be streamlined or the instructions may become more detialed during the penalty free zone.

Explore Badges can take different forms so the sections below outline some options. This page is not a cumulative list of requirements or an exhaustive list of options.

Tip

You might get a lot of suggestions for improvement on your first one, but if you apply that advice to future ones, they will get approved faster.

How do I propose?

Create an issue on your kwl repo, label it explore, and “assign” @brownsarahm.

In your issue, describe the question you want to answer or topic to explore and the format you want to use.

If you propose something too big, you might be advised to consider a build badge instead. If you propose something too small, you will get ideas as options for how to expand it and you pick which ones.

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.

Important

Either way, there must be a separate issue for this work that is also linked to your PR

What should the work look like?

It should look like a blog post, written tutorial, graphic novel, or visual aid with caption. It will likely contain some code excerpts the way the class 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.

The exact length can vary, but these must go beyond what we do in class in scope

Explore Badge Ideas:

Skip to main content

- for a more practice that asks you to describe potential uses for a tool, try it out, find or write code excerpts and examine them
    - for a more practice that asks you to try something, try some other options and compare and contrast them. eg “try git in your favorite IDE” -> “try git in three different IDEs, compare and contrast, and make recommendations for novice developers”
- For a topic that left you still a little confused or their was one part that you wanted to know more about. 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. eg:
    - Describe how cryptography evolved and what caused it to evolve (i.e. SHA-1 being decrypted)
    - Learn a lot more about a specific number system
    - compare another git host
    - try a different type of version control
- Create a visual aid/memory aid to help remember a topic. Draw inspriation from Wizard Zines or
- Review a reference or resource for a topic

Examples from past students:

- Scripts/story boards for tiktoks that break down course topics
- Visual aid drawings to help remember key facts

For special formatting, use jupyter book’s documentation.

Not

These
these
struct
the sa

Build Badges

Warning

This page is subject to change until the end of the penalty free zone

Proposal Template

If you have selected to do a project, please use the following template to propose a bulid

```
## < Project Tite >

<!-- insert a 1 sentence summary  -->

### Objectives

<!-- in this section describe the overall goals in terms of what you will learn and the problem you will solv

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi

### Deliverables

<!-- list what your project will produce with target deadlines for each-->

### Milestones
```

Skip to main content

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, plus unlimited references 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 your kwl repo as a new file `build_report_title.md` where `title` is the (title or a shortened version) from the 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.

Build Ideas

- make a vs code extension for this class or another URI CS course
- port the courseutils to rust. crate clap is like the python click package I used to develop the course utils
- buld a polished documentation website for your CSC212 project with sphinx or another static site generator
-

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 earning the badges. Everything at a level must be complete and correct.

How do I keep track of my earned badges?

with in one place. This is quite different than checking your grade in BrightSpace, but using tools like this represents the real tools used by developers.

You will be able to use provided command line tools and github actions to produce a report of your status at any time from your PR list, starting in the third week. Additionally, at particular points in the course, an in class or class preparation activity will be for you to review a “progress report” that we help you create and update your success plan for the course.

Also, when are each badge due, time wise?

Review and practice must start within a week, but I recommend starting before the next class. Must be a good faith completion within 2 weeks, but again recommend finishing sooner.

Experience reports for missing class is on a case by case basis depending on why you missed class. You must have a plan by the next class.

Explore and build, we’ll agree to a deadline when you propose.

Will everything done in the penalty free zone be approved even if there are mistakes?

No. In the penalty-free zone I still want you to learn things, but we will do extra work to make sure that you get credit for all of your effort even if you make mistakes in how to use GitHub. We will ask you to fix things that we have taught you to fix, but not things that we will not cover until later.

The goal is to make things more fair while you get used to GitHub. It’s a nontrivial thing to learn, but getting used to it is worth it.

I want this class to be a safe place for you to try things, make mistakes and learn from them without penalty. A job is a much higher stakes place to learn a tool as hard as GitHub, so I want this to be lower stakes, even though I cannot promise it will be easy.

Once we make revisions on a pull request, how do we notify you that we have done them?

You do not have to do anything, GitHub will automatically notify which ever one of us who reviewed it initially when you make changes.

What should work for an explore badge 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.

I can’t push to my repository, I get an error that updates were rejected

If your error looks like this…

```
! [rejected] main -> main (fetch first)
error: failed to push some refs to <repository name>
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Your local version and github version are out of sync, you need to pull the changes from github to your local computer before you can push new changes there.

After you run

```
git pull
```

You’ll probably have to resolve a merge conflict

My command line says I cannot use a password

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

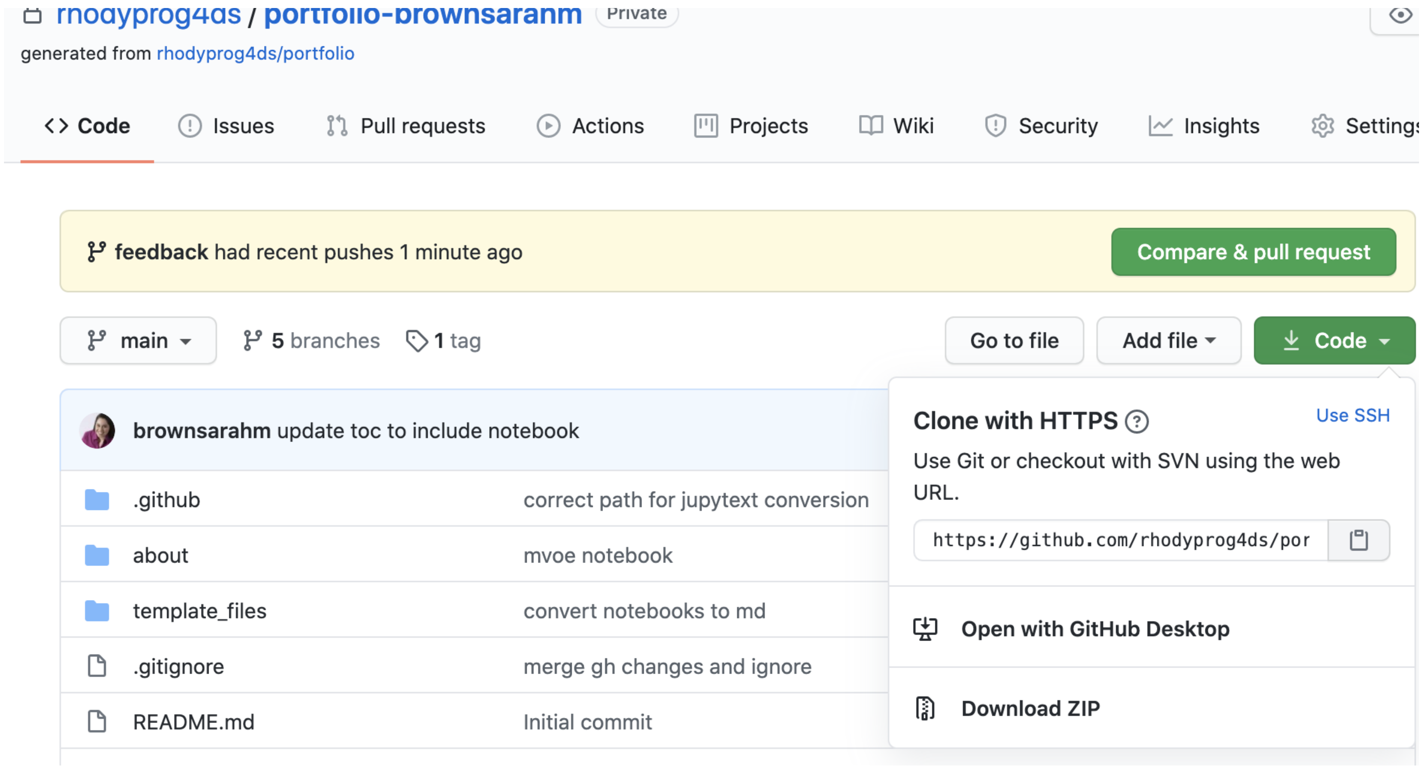
Help! I accidentally merged the Badge Pull Request before my assignment was graded

That’s ok. You can fix it.

note: these instructions use the main branch the way we use the badge branches and the feedback branch the way we use the main branch in this course

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


rhodyprog4ds / portfolio-brownsarahm Private

generated from rhodyprog4ds/portfolio

Use Git or checkout with SVN using the web URL.

```
https://github.com/rhodyprog4ds/por
```

brownsarahm update toc to include notebook

.github	correct path for jupytext conversion
about	mvoe notebook
template_files	convert notebooks to md
.gitignore	merge gh changes and ignore
README.md	Initial commit

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.

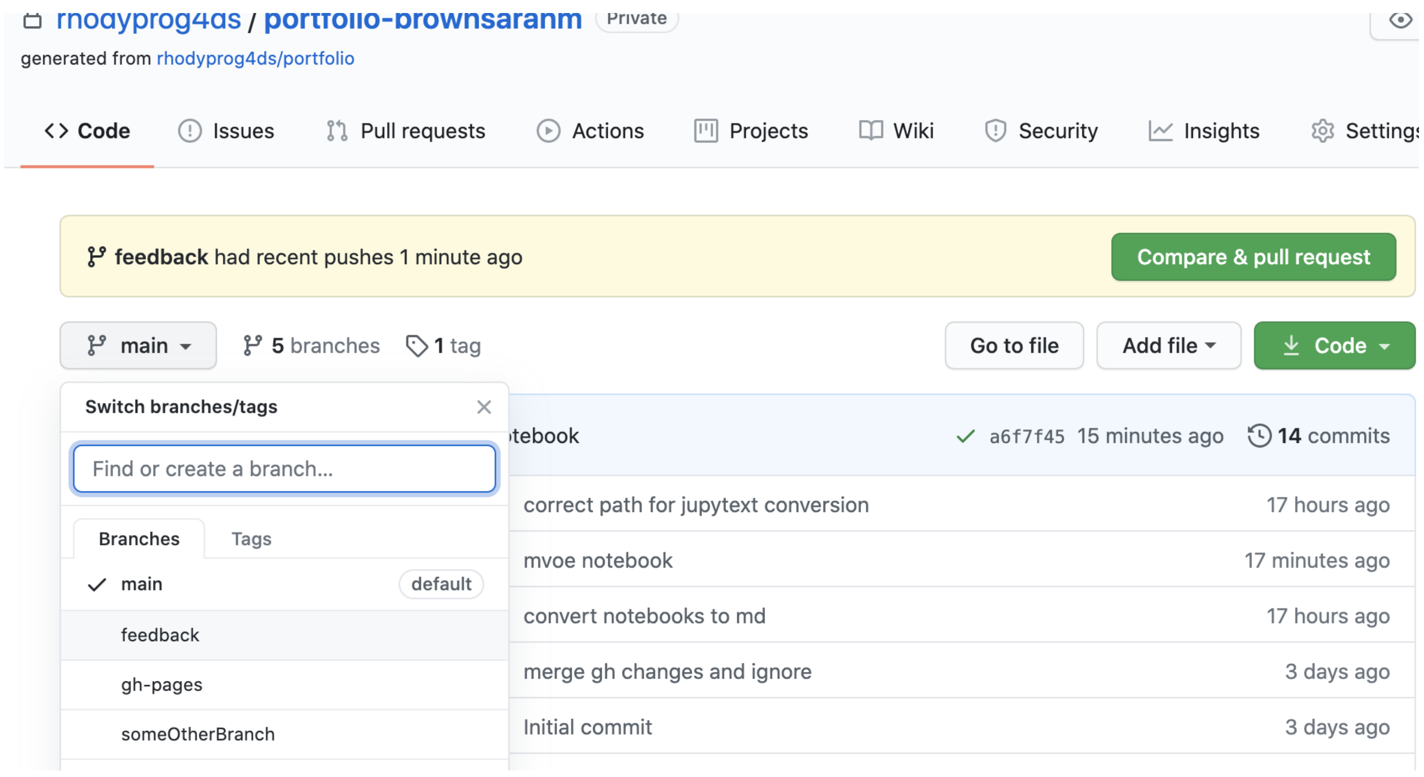
```
git clone https://github.com/rhodyprog4ds/portfolio-brownsarahm.git
```

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.

Skip to main content


rhodyprog4ds / portfolio-brownsarahm Private
generated from rhodyprog4ds/portfolio

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

feedback had recent pushes 1 minute ago — Compare & pull request

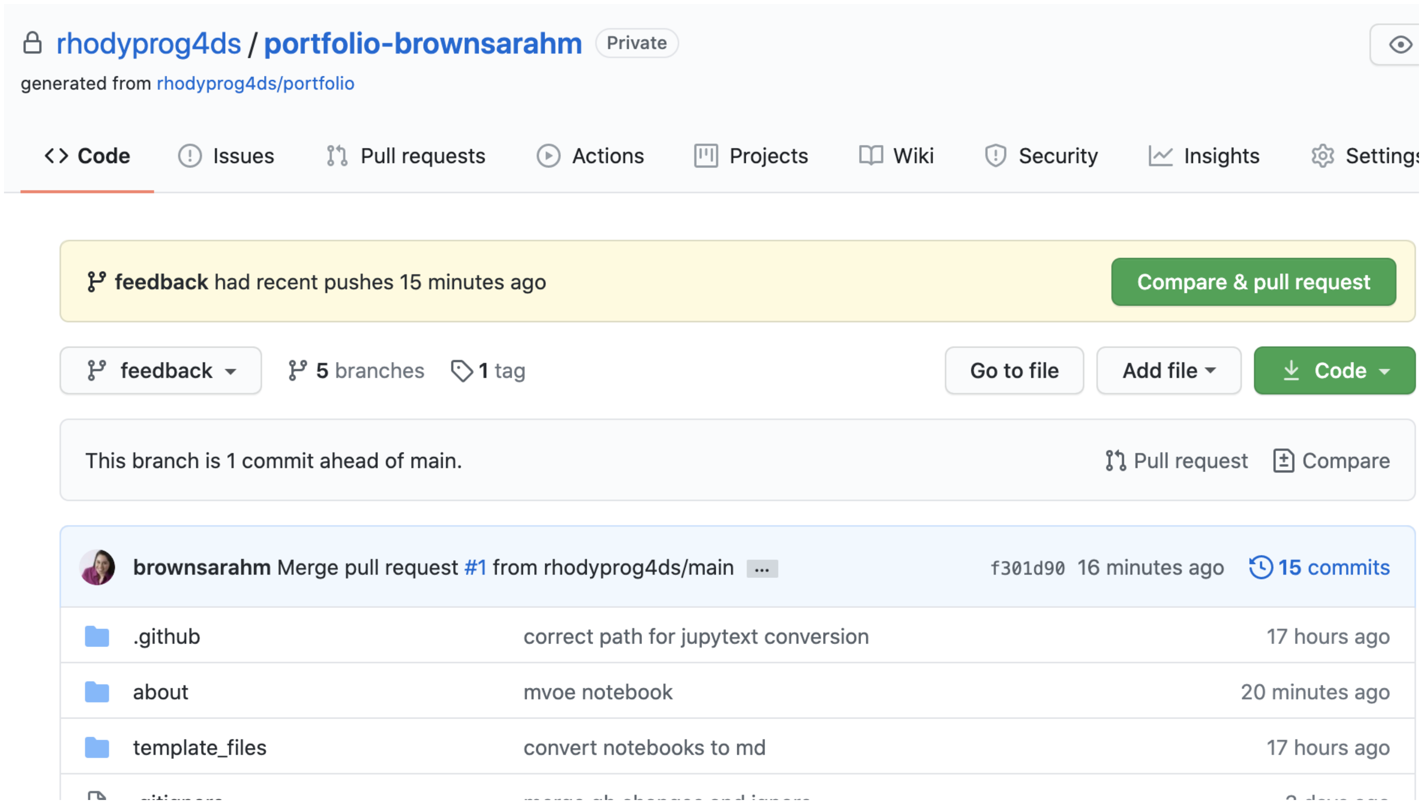
main · 5 branches · 1 tag — Go to file · Add file · Code

Switch branches/tags
Find or create a branch...
Branches | Tags
✓ main (default)
feedback
gh-pages
someOtherBranch

tebook ✓ a6f7f45 15 minutes ago 14 commits

correct path for jupytext conversion	17 hours ago
mvoe notebook	17 minutes ago
convert notebooks to md	17 hours ago
merge gh changes and ignore	3 days ago
Initial commit	3 days ago

Now view the list of all of the commits to this branch, by clicking on the clock icon with a number of commits


rhodyprog4ds / portfolio-brownsarahm Private
generated from rhodyprog4ds/portfolio

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

feedback had recent pushes 15 minutes ago — Compare & pull request

feedback · 5 branches · 1 tag — Go to file · Add file · Code

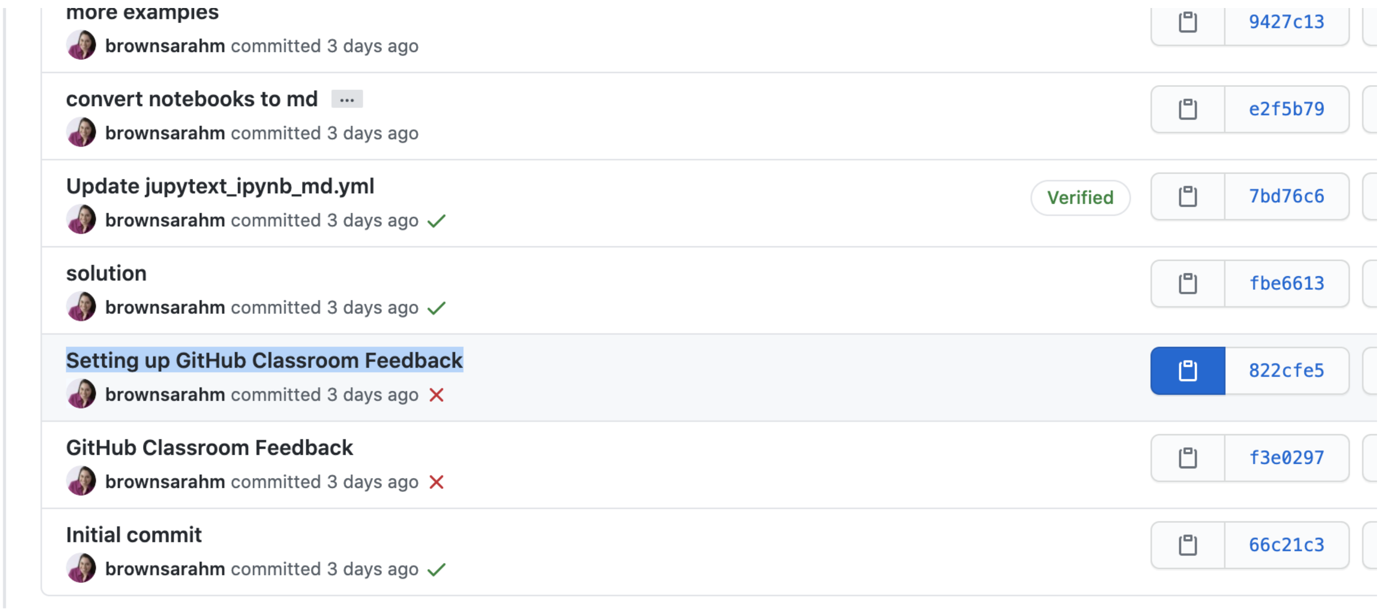
This branch is 1 commit ahead of main. — Pull request · Compare

brownsarahm Merge pull request #1 from rhodyprog4ds/main ... f301d90 16 minutes ago 15 commits

.github	correct path for jupytext conversion	17 hours ago
about	mvoe notebook	20 minutes ago
template_files	convert notebooks to md	17 hours ago

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.

Skip to main content


more examples
brownsarahm committed 3 days ago 9427c13
convert notebooks to md ...
brownsarahm committed 3 days ago e2f5b79
Update jupytext_ipynb_md.yml Verified
brownsarahm committed 3 days ago ✓ 7bd76c6
solution
brownsarahm committed 3 days ago ✓ fbe6613
Setting up GitHub Classroom Feedback
brownsarahm committed 3 days ago ✗ 822cfe5
GitHub Classroom Feedback
brownsarahm committed 3 days ago ✗ f3e0297
Initial commit
brownsarahm committed 3 days ago ✓ 66c21c3

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)
```

Skip to main content

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.

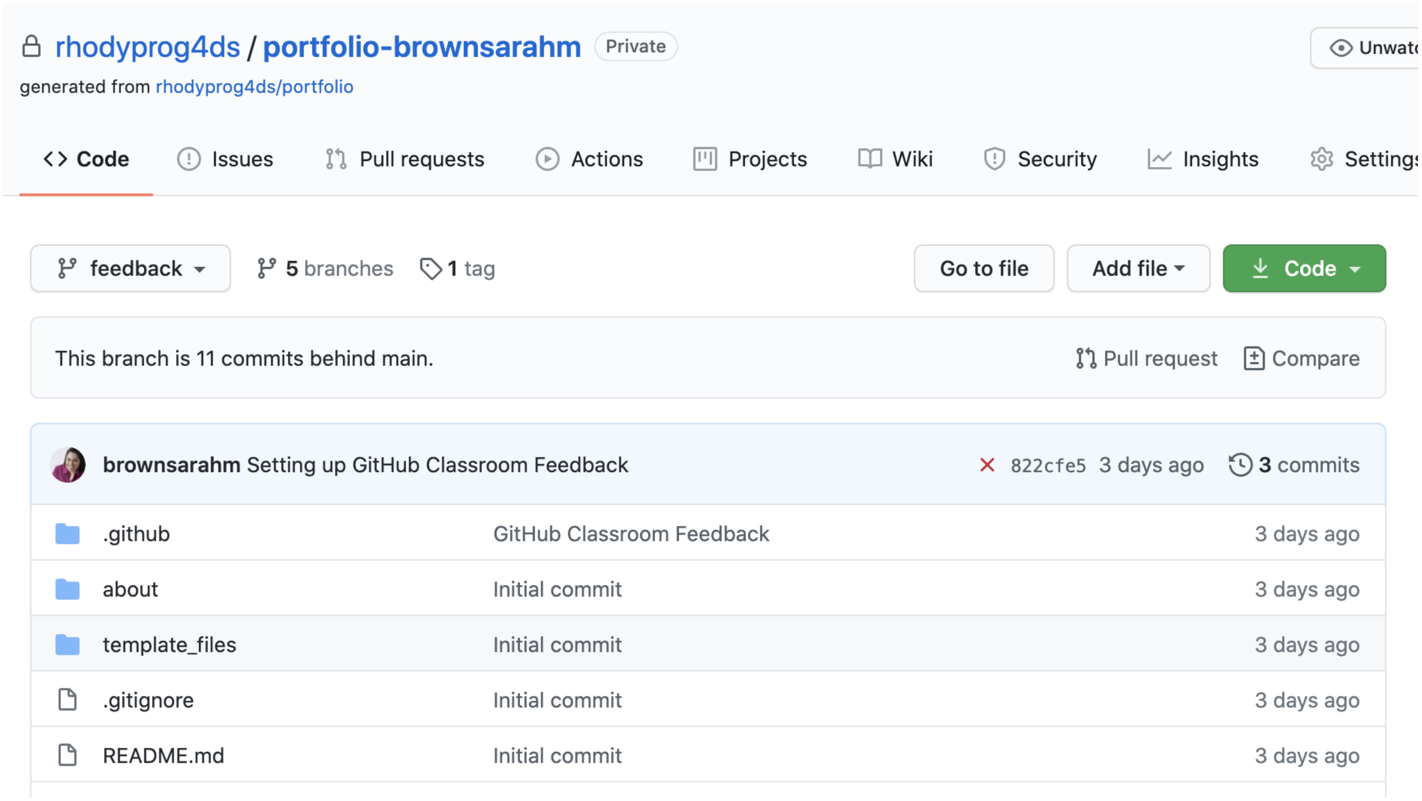
```
git push origin -f
```

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").



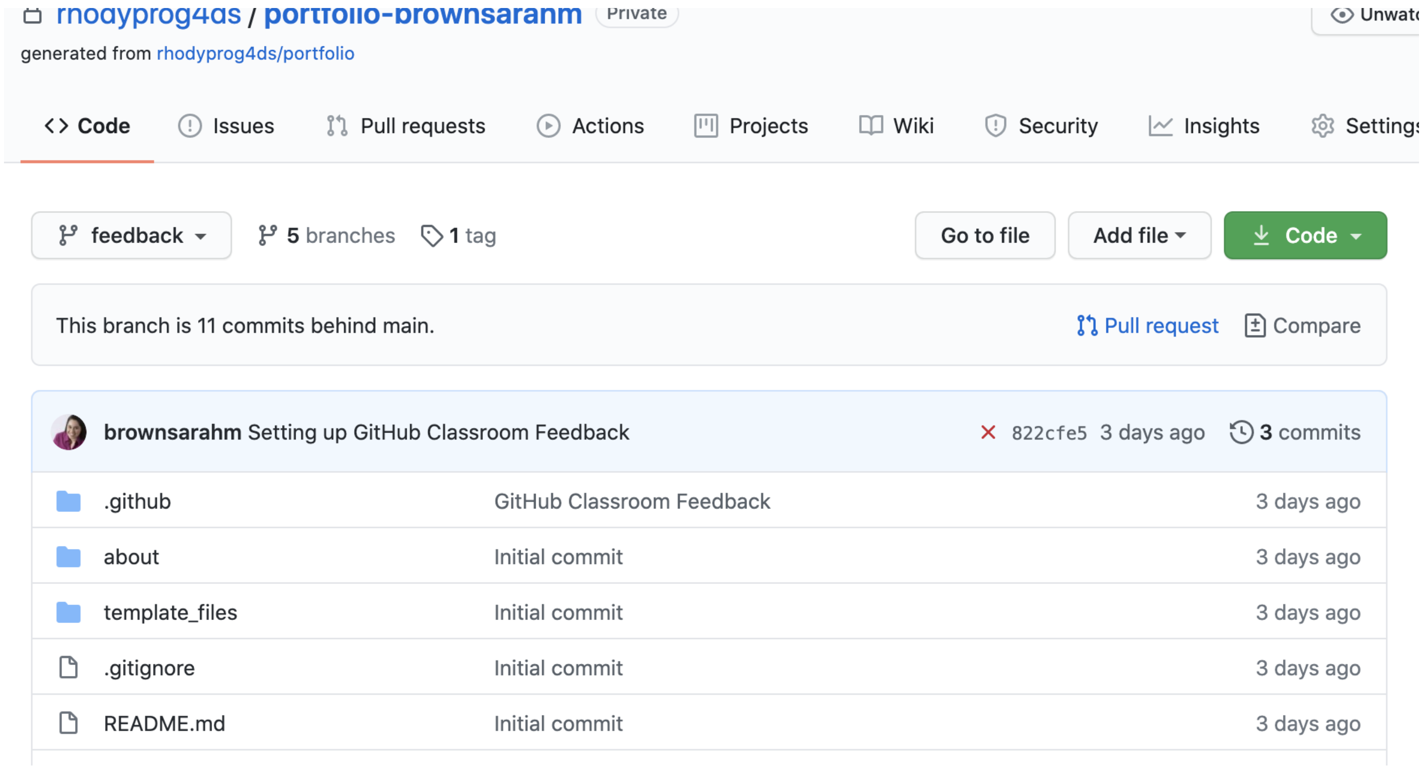
This branch is 11 commits behind main. Pull request Compare

brownsarahm Setting up GitHub Classroom Feedback ✕ 822cfe5 3 days ago 3 commits

.github	GitHub Classroom Feedback	3 days ago
about	Initial commit	3 days ago
template_files	Initial commit	3 days ago
.gitignore	Initial commit	3 days ago
README.md	Initial commit	3 days ago

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

Skip to main content


rhodyprog4ds / portfolio-brownsarahm Private

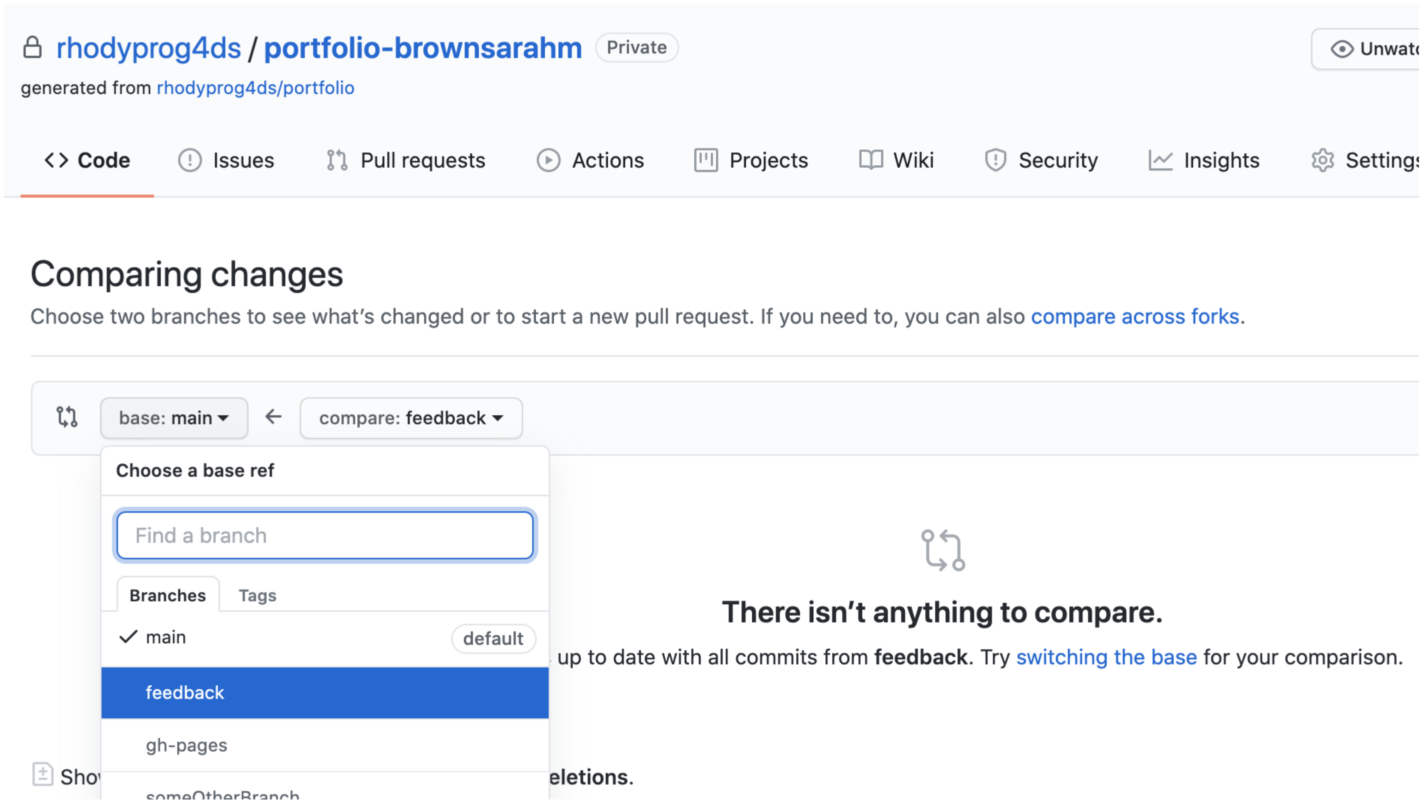
generated from rhodyprog4ds/portfolio

This branch is 11 commits behind main. Pull request Compare

brownsarahm Setting up GitHub Classroom Feedback ✗ 822cfe5 3 days ago 3 commits

.github	GitHub Classroom Feedback	3 days ago
about	Initial commit	3 days ago
template_files	Initial commit	3 days ago
.gitignore	Initial commit	3 days ago
README.md	Initial commit	3 days ago

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.


rhodyprog4ds / portfolio-brownsarahm Private

generated from rhodyprog4ds/portfolio

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

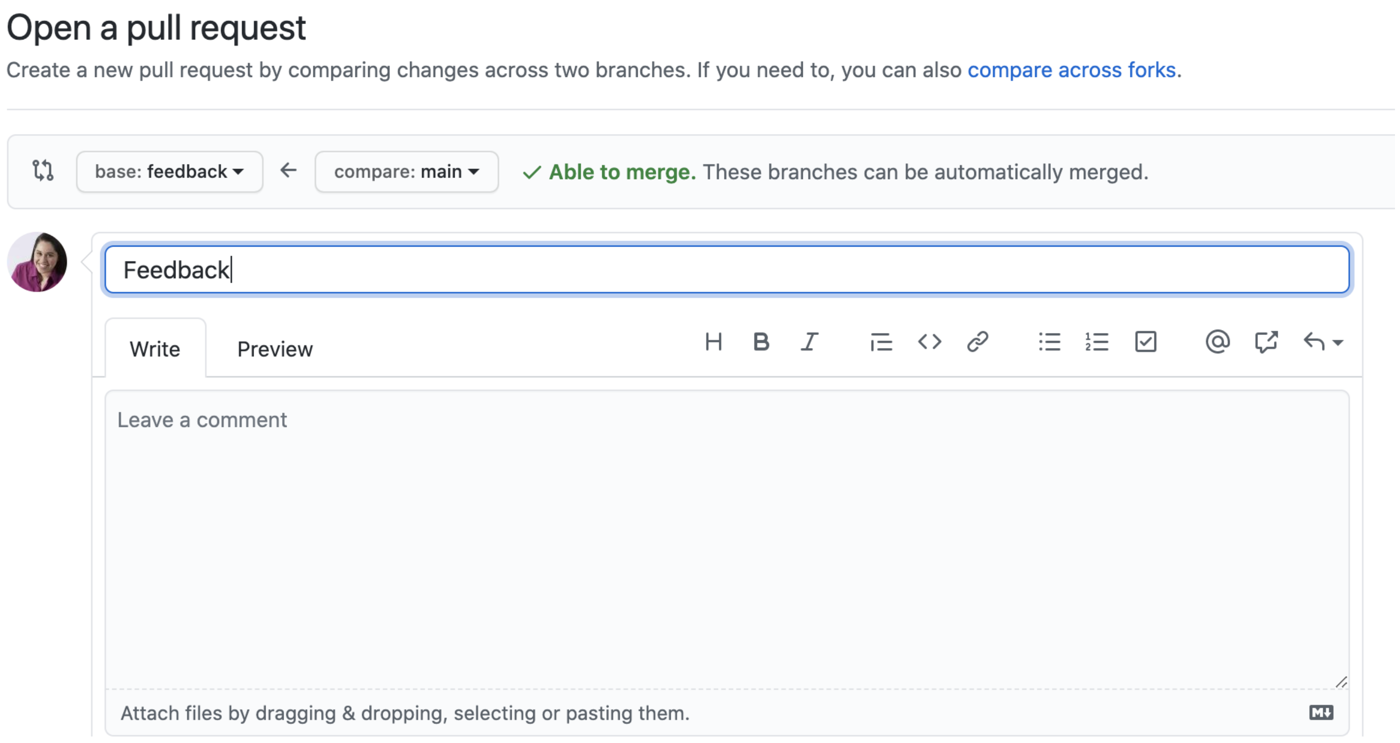
There isn't anything to compare.

up to date with all commits from **feedback**. Try switching the base for your comparison.

Sho eletions.

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.

Skip to main content


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: feedback ← compare: main ✓ Able to merge. These branches can be automatically merged.

Feedback

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.

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
```

Glossary

Tip

We will build a glossary as the semester goes on. When you encounter a term you do not know, create an issue to ask for help, or contribute a PR after you find the answer.

absolute path

the path defined from the root of the system

add (new files in a repository)

the step that stages/prepares files to be committed to a repository from a local branch

bitwise operator

Skip to main content

bitwise operator

an operation that happens on a bit string (sequence of 1s and 0s). They are typically faster than operations on whole integers.

Compiled Code

code that is put through a compiler to turn it into lower level assemlby language before it is executed. must be compiled and re-executed everytime you make a change.

directory

a collection of files typically created for organizational purposes

floating point number

the concept that the decimal can move within the number (ex. scientific notation; you move the decimal based on the exponent on the 10). can represent more numbers than a fixed point number.

fixed point number

the concept that the decimal point does not move in the number (the example in the notes where if we split up a bit in the middle and one half was for the decimal and the other half was for the whole number. Cannot represent as many numbers as a floating point number.

.gitignore

a file in a git repo that will not add the files that are included in this .gitignore file. Used to prevent files from being unnecessarily committed.

git

a version control tool; it’s a fully open source and always free tool, that can be hosted by anyone or used without a host, locally only.

git objects

something (a file, directory) that is used in git; has a hash associated with it

GitHub

a hosting service for git repositories

Git Plumbing commands

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

Git Workflow

a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner

HEAD

the branch that is currently being checked out (think of the current branch)

merge

putting two branches together so that you can access files in another branch that are not available in yours

hash function

the actual function that does the hashing of the input (a key, an object, etc.)

putting an input through a function and getting a different output for every input (the output is called a hash; used in hash tables and when git hashes commits).

interpreted code

code that is directly executed from a high level language. more expensive computationally because it cannot be optimized and therefore can be slower.

integreated development environment

also known as an IDE, puts together all of the tools a developer would need to produce code (source code editor, debugger, ability to run code) into one application so that everything can be done in one place. can also have extra features such as showing your file tree and connecting to git and/or github.

Linker

a program that links together the object files and libraries to output an executable file.

path

the “location” of a file or folder(directory) in a computer

pull (changes from a repository)

download changes from a remote repository and update the local repository with these changes.

push (changes to a repository)

to put whatever you were working on from your local machine onto a remote copy of the repository in a version control system.

relative path

the path defined **relative** to another file or the current working directory

repository

a project folder with tracking information in it in the form of a .git file

ROM (Read-Only Memory)

Memory that only gets read by the CPU and is used for instructions

SHA 1

the hashing function that git uses to hash its functions (found to have very serious collisions (two different inputs have same hashes), so a lot of software is switching to SHA 256)

shell

a command line interface; allows for access to an operating system

ssh

allows computers to safely connect to networks (such as when we used an ssh key to clone our github repos)

templating

templating is the idea of changing the input or output of a system. For instance, the Jupyter book, instead of outputting the markdown files as markdown files, displays them as HTML pages (with the contents of the markdown file).

terminal

Skip to main content

tree objects
: type of git object in git that helps store multiple files with their hashes (similar to directories in a file system)

yml
: see YAML

YAML
: a file specification that stores key-value pairs. It is commonly used for configurations and settings.

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.

Begin a great programmer does not require memorizing all of the specific commands, but instead knowing the common patterns and how to use them to interpret others' code and write your own. Being efficient requires knowing how to use tools and how to let the computer do tedious tasks for you. This is how this course is designed to help you, but you have to get practice with these things.

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. These tools can help you when you are writing cod eand forget a specific bit of syntax, but these tools will not help you *read* code or debug environment issues. You also have to know how to effectively use these tools.
Knowing the common abstractions we use in computing and recognizing them when they look a little bit differently will help you with these more complex tasks. Understanding what is common when you move from one environment to another or to 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.

Why this way?

Learning requires iterative practice. In this class, you will first get ready to learn by preparing for class. Then, in class, you will get a first experience with the material. The goal is that each class is a chance to learn by engaging with the ideas, it is to be a guided inquiry. Some classes will have a bit more lecture and others will be all hands on with explanation, but the goal is that you *experience* the topics in a way that helps you remember, because being immersed in an activity helps brains remember more than passively watching something. Then you have to practice with the material

A new boo
programmi
Brain As o
by clicking
contents s

Preparing for class will be activities that helpy ou bring your prior knowledge to class in the most helpful way, help me mee

Skip to main content

on a recommended practices from working devs to [keep a notebook]](https://blog.nelhage.com/2010/05/software-and-lab-notebooks/) or keep a blog and notebook.

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.

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.

GitHub Interface reference

This is an overview of the parts of GitHubt from the view on a repository page. It has links to the relevant GitHub documentation for more detail.

Top of page

The very top menu with the logo in it has GitHub level menus that are not related to the current repository.

Skip to main content

Code Issues Pull Requests Actions Projects Security Insights Settings

This is the main view of the project

Branch menu & info, file action buttons, download options (green code button)

File panel

the header in this area lists who made the last commit, the message of that commit, the short hash, date of that commit and the total number of commits to the project.

If there are actions on the repo, there will be a red x or a green check to indicate that if it failed or succeeded on that commit.

the header in this area lists who made the last commit, the message of that commit, the short hash, date of that commit and the total number of commits to the project.

If there are actions on the repo, there will be a red x or a green check to indicate that if it failed or succeeded on that commit. ^^^ file list: a table where the first column is the name, the second column is the message of the last commit to change that file (or folder) and the third column is when is how long ago/when that commit was made

README file

About has basic facts about the repo, often including a link to a documentation page

Releases, Packages, and Environments are optional sections that the repo owner can toggle on and off.

Releases mark certain commits as important and give easy access to that version. They are related to git tags

Packages are out of scope for this course. GitHub helps you manage distributing your code to make it easier for users.

Environments are a tool for dependency management. We will cover thigns that help you know how to use this feature indirectly, but probably will not use it directly in class. This would be eligible for a build badge.

The bottom of the right panel has information about the languages in the project

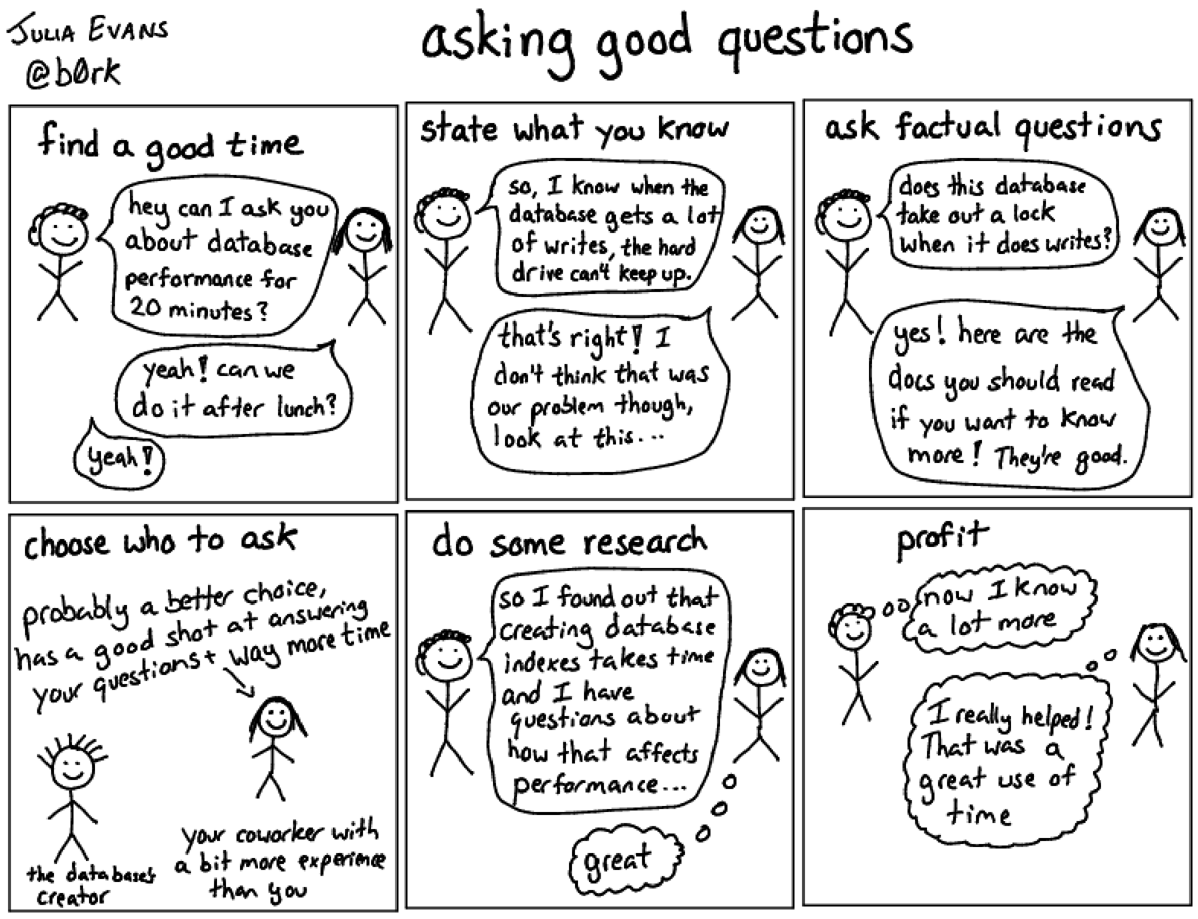
Language/Shell Specific References

- bash
- C
- Python

Skip to main content

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


Julia Evans
@b0rk
asking good questions
find a good time
hey can I ask you about database performance for 20 minutes?
yeah! can we do it after lunch?
yeah!
state what you know
so, I know when the database gets a lot of writes, the hard drive can't keep up.
that's right! I don't think that was our problem though, look at this...
ask factual questions
does this database take out a lock when it does writes?
yes! here are the docs you should read if you want to know more! They're good.
choose who to ask
probably a better choice, has a good shot at answering your questions + way more time
the database's creator
your coworker with a bit more experience than you
do some research
so I found out that creating database indexes takes time and I have questions about how that affects performance...
great
profit
now I know a lot more
I really helped! That was a great use of time

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.

Not

A fun
debu

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.

Skip to main content

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.

Windows Help & Notes

CRLF Warning

This is GitBash telling you that git is helping. Windows uses two characters for a new line `CR` (cariage return) and `LF` (line feed). Classic Mac Operating system used the `CR` character. Unix-like systems (including MacOS X) use only the `LF` character. If you try to open a file on Windows that has only `LF` characters, Windows will think it’s all one line. To help you, since git knows people collaborate across file systems, when you check out files from the git database (`.git/` directory) git replaces `LF` characters with `CRLF` before updating your working directory.

When working on Windows, when you make a file locally, each new line will have `CRLF` in it. If your collaborator (or server, eg GitHub) runs not a unix or linux based operating system (it almost certainly does) these extra characters will make a mess and make the system interpret your code wrong. To help you out, git will automatically, for Windows users, convert `CRLF` to `LF` when it adds your work to the index (staging area). Then when you push, it’s the compatible version.

git documentation of the feature