About this Site

Contents

Skip to main content

Welcome to the course website for Computer Systems and Programming Tools in Spring 2024 with Professor Brown.

This class meets TuTh 12:30-1:45 in Ranger 302 and lab on Monday 3-4:45 in Ranger 202.

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

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.

```
# this is a comment in a clode block
command argument --option -a
```

```
command output
important line, emphasized
```

Try it Yourself

Notes will have exercises marked like this

Skip to main content

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

Further reading

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

Hint

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

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

Que

Ques
unans
answ
with a
will be

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.

Skip to main content

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 fellowship at University of California Berkeley, and worked as a postdoctoral research associate at Brown University before joining URI. At Brown University, Dr. Brown taught the Data and Society course for the Master's in Data Science Program. You can learn more about me at my website or my research on my lab site.

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

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

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.

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

You can get a transcript from class from Prismia.chat using the menu in the top right.

Seein
loggi
being

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:

- 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

Recommendation

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:

Skip to main content

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.
7. Differentiate between social conventions and technical requirements in programming contexts.

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 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.
- 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 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.

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

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 submitted.

What happens if you merged a PR without feedback?

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. There will also be a lab 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 form of engagment is tracked with a particular type of badge that you can earn as you accumulate evidence of your learning and engagment.

- experience: guided in class activities, with reflection
- lab: accountability to basics through 1:1 conversation with a member of the instructional team
- review: just the basics
- practice: a little bit more indepdendent
- explore: posing your own directions of inquiry
- build: in depth application of course topics

Below are the recommended paths to different grade targets.

To earn a D, complete:

- 22 experience badges
- 13 lab check outs

Skip to main content

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

To earn a B, complete:

- 22 experience badges
- 13 lab check outs
- one of the following:
    - 18 practice badges
    - 12 review + 12 practice

For an A you must complete:

- 22 experience badges
- 13 lab check outs
- one of the following:
    - 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.

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.

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.

These
detail
calcu
cutoff
calcu

Skip to main content

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

In class

You earn an experience badge in class by:

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

Makeup

You can make up an experience badge by:

- preparing for class
- reading the posted notes
- completing the activity from the notes
- completeing an "experience report"
- attaching evidence as indiated in notes OR attending office hours to show the evidence

Tip

On prismia questions, I will generally give a "Last chance to get an answer in" warning before I resume instruction. If you do not respond at all too many times, we will ask you to follow the makeup procedure instead of the In Class proccedure for your experience badge.

To be sure that your response rate is good, if you are paying attention, but do not have an answer you can use one of the following special commands in prismia:

- `\idk`: "I am paying attention, but do not know how to answer this"
- `\dgt`: "I am paying attention, not really confused, but ran out of time trying to figure out the answer"

you can send these as plain text by pressing `enter` (not Mac) or `return` (on Mac) to send right away or have them render to emoji by pressing `tab` 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 advised 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.

Lab Checkouts

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

Where do you find assignments?

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.

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

Build badges are for when you have an idea of something you want to do. There are also some ideas on the build page.

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

Skip to main content

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 impactful**

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
- contributing annotated resources the course website

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.

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".

Important

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

Not

We d
assig
The c
be th

Skip to main content

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**

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.

What do you think?

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

Badge Deadlines and 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.

Skip to main content

```
%matplotlib inline
import os
from datetime import date,timedelta
import calendar
import pandas as pd
import numpy as np
import seaborn as sns
from myst_nb import glue
# style note: when I wrote this code, it was not all one cell. I merged the cells
# for display on the course website, since Python is not the main outcome of this course

# semester settings
first_day = date(2024,1,22)
last_day = date(2024,4,29)


# no_class_ranges = [(date(2023,11,23),date(2023,11,26)),
#                      (date(2023,11,13)),
#                    (date(2023,10,10))]
no_class_ranges = [(date(2024,3,10),date(2024,3,16)),
                    (date(2024,2,19))]


meeting_days =[1,3] # datetime has 0=Monday

penalty_free_end = date(2024, 2, 9)


def day_off(cur_date,skip_range_list):
    '''
    is the current date a day off?

    Parameters
    ----------
    cur_date : datetime.date
        date to check
    skip_range_list : list of datetime.date objects or 2-tuples of datetime.date
        dates where there is no class, either single dates or ranges specified by a tuple

    Returns
    -------
    day_is_off : bool
        True if the day is off, False if the day has class
    '''
    # default to not a day off
    day_is_off=False
    #
    for skip_range in skip_range_list:
        if type(skip_range) == tuple:
            # if any of the conditions are true that increments and it will never go down, flase=0, true=1
            day_is_off +=  skip_range[0]<=cur_date<=skip_range[1]
        else:
            day_is_off += skip_range == cur_date
    #
    return day_is_off


# enumerate weeks

mtg_delta = timedelta(meeting_days[1]-meeting_days[0])
week_delta = timedelta(7)



possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
weekly_meetings = [[c1,c2] for c1,c2 in possible if not(day_off(c1,no_class_ranges))]
meetings = [m for w in weekly_meetings for m in w]
meetings_string = [m.isoformat() for m in meetings]
weekly_meetings


# possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
```

Skip to main content

```
# build a table for the dates
badge_types = ['experience', 'review', 'practice']
target_cols = ['review_target','practice_target']
df_cols = badge_types + target_cols
badge_target_df = pd.DataFrame(index=meetings, data=[['future']*len(df_cols)]*len(meetings),
                                columns=df_cols).reset_index().rename(
                                   columns={'index': 'date'})
# set relative dates
today = date.today()
start_deadline = date.today() - timedelta(7)
complete_deadline = date.today() - timedelta(14)

# mark eligible experience badges
badge_target_df['experience'][badge_target_df['date'] <= today] = 'eligible'
# mark targets, cascading from most recent to oldest to not have to check < and >
badge_target_df['review_target'][badge_target_df['date'] <= today] = 'active'
badge_target_df['practice_target'][badge_target_df['date'] <= today] = 'active'
badge_target_df['review_target'][badge_target_df['date']
                          <= start_deadline] = 'started'
badge_target_df['practice_target'][badge_target_df['date']
                            <= start_deadline] = 'started'
badge_target_df['review_target'][badge_target_df['date']
                          <= complete_deadline] = 'completed'
badge_target_df['practice_target'][badge_target_df['date']
                            <= complete_deadline] = 'completed'
# mark enforced deadlines
badge_target_df['review'][badge_target_df['date'] <= today] = 'active'
badge_target_df['practice'][badge_target_df['date'] <= today] = 'active'
badge_target_df['review'][badge_target_df['date']
                          <= start_deadline] = 'started'
badge_target_df['practice'][badge_target_df['date']
                            <= start_deadline] = 'started'
badge_target_df['review'][badge_target_df['date']
                          <= complete_deadline] = 'completed'
badge_target_df['practice'][badge_target_df['date']
                            <= complete_deadline] = 'completed'
badge_target_df['review'][badge_target_df['date']
                          <= penalty_free_end] = 'penalty free'
badge_target_df['practice'][badge_target_df['date']
                          <= penalty_free_end] = 'penalty free'

# convert to numbers and set dates as index for heatmap compatibility
status_numbers_hm = {status:i+1 for i,status in enumerate(['future','eligible','active','penalty free','star
badge_target_df_hm = badge_target_df.replace(status_numbers_hm).set_index('date')

# set column names to shorter ones to fit better
badge_target_df_hm = badge_target_df_hm.rename(columns={'review':'review(e)','practice':'practice(e)',
                                                       'review_target':'review(t)','practice_target':'practi
# build a custom color bar
n_statuses = len(status_numbers_hm.keys())
manual_palette = [sns.color_palette("pastel", 10)[7],
                 sns.color_palette("colorblind", 10)[2],
                 sns.color_palette("muted", 10)[2],
                 sns.color_palette("colorblind", 10)[9],
                 sns.color_palette("colorblind", 10)[8],
                 sns.color_palette("colorblind", 10)[3]]
# generate the figure, with the colorbar and spacing
ax = sns.heatmap(badge_target_df_hm,cmap=manual_palette,linewidths=1)
# mote titles from bottom tot op
ax.xaxis.tick_top()
# pull the colorbar object for handling
colorbar = ax.collections[0].colorbar
# fix the location fo the labels on the colorbar
r = colorbar.vmax - colorbar.vmin
colorbar.set_ticks([colorbar.vmin + r / n_statuses * (0.5 + i) for i in range(n_statuses)])
colorbar.set_ticklabels(list(status_numbers_hm.keys()))
# add a title
today_string = today.isoformat()
glue('today',today_string,display=False)
glue('today_notdisplayed',"not today",display=False)
```

Skip to main content

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 65
     60 mtg_delta = timedelta(meeting_days[1]-meeting_days[0])
     61 week_delta = timedelta(7)
---> 65 possible = [(first_day+week_delta*w, first_day+mtg_delta+week_delta*w) for w in range(weeks)]
     66 weekly_meetings = [[c1,c2] for c1,c2 in possible if not(day_off(c1,no_class_ranges))]
     67 meetings = [m for w in weekly_meetings for m in w]

NameError: name 'weeks' is not defined
```

Deadlines

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.

These badges badges must be *started* within one week of when the are posted (2pm) and *completed* within two weeks. A task is 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

Skip to main content

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 5 stages:

- 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. At most, you will receive feedback for one explore badge per week, so in order to earn six, your first one must be complete by March 18.

Build Badges

At most one build badge will be evaluated every 4 weeks. This means that if you want to earn 3 build badges, the first one must be in 8 weeks before the end of the semester, March 4. The second would be due April 1st, and the third submitted by the end of classes, April 29th.

Prepare work and Experience Badges Process

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:

gitGraph commit commit checkout main branch prepare-2023-09-12 checkout prepare-2023-09-12 commit id: "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

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] 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 commit[commit work to the branch]

Skip to main content

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 in your KWL repo: 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.

Warning

Some phrasing on this may change, but the core of what is required will not change

▶ Show code cell source

Tip

You c
see n
are c

Skip to main content

```
 'D+': 124,
 'C-': 142,
 'C ': 192,
 'C+': 210,
 'B-': 228,
 'B ': 246,
 'B+': 264,
 'A-': 282,
 'A ': 300}
```

▶ Show code cell source

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 23
     21 learning_df = pd.Series(learning_weights,name ='complexity').reset_index()
     22 learning_df['badge_type'] = 'learning'
---> 23 comm_df  = pd.Series(community_weights,name='weight').reset_index()
     24 comm_df['badge_type'] = 'community'
     25 # nans are for learning badges which all ahve weight 1

NameError: name 'community_weights' is not defined
```

The total influence of each badge on the grade is as follows:

▶ Show code cell source

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 # display
----> 2 influence_df[['badge_type','badge','complexity','weight','influence']]

NameError: name 'influence_df' is not defined
```

The total influence of a badge on your grade is the product of the badge’s complexity. All learning badges have a weight of 1, but have varying complexity.

Bonuses

In addition to the weights for each badge, there also bonuses that will automatically applied to your grade at the end of the semester. These are for longer term patterns, not specific assignments. You earn these while workng on other assignments, not separately.

Important

the grade plans on the grading page and the thresholds above assume you earn the Participation and Lab bonuses for all grades a D or above and the Breadth bonus for all grades above a C.

Name	Definition	Influence	type
Participation	22 experience badges	18	auto
Lab	13 lab badges	18	auto
Breadth	If review + practice badges >-18:	32	auto
Git-ing unstuck	fix large mistakes your repo using advanced git operations and submit a short reflection (allowable twice; Dr. Brown must approve)	9	event
Early bird	6 review + practice submitted by 2/15	9	event
Descriptive commits	all commits in KWL repo and build repos after penalty free zone have descriptive commit messages (not GitHub default or nonsense)	9	event
Community Star	10 community badges	18	auto

Auto bonuses will be calculated from your other list of badges. Event bonuses will be logged in your KWL repo, where you get instructions when you meet the criteria.

Note

These bonuses are not pro-rated, you must fulfill the whole requirement to get the bonus. Except where noted, each bonus may only be earned once

Note

You cannot guarantee you will earn the Git-ing unstuck bonus, if you want to intentionally explore advanced operations, you can propose an explore badge, which is also worth 9.

Grade thresholds

Grade cutoffs for total influence are:

▶ Show code cell source

Skip to main content

letter	threshold
D	106
D+	124
C-	142
C	192
C+	210
B-	228
B	246
B+	264
A-	282
A	300

Bonus Implications

Attendance and participation is *very* important:

- 14 experience, 6 labs, and 9 practice is an F
- 22 experience, 13 labs, and 9 practice is a C-
- 14 experience, 6 labs, 9 practice and one build is a C-
- 22 experience, 13 labs, 9 practice and one build is a C+

Missing one thing can have a nonlinear effect on your grade. Example 1:

- 22 experience, 13 labs, and 18 review is a C
- 21 experience, 13 labs, and 18 review is a C-
- 21 experience, 13 labs, and 17 review is a D+
- 21 experience, 12 labs, and 17 review is a D

Example 2:

- 22 experience, 13 labs, and 17 practice is a C
- 22 experience, 13 labs, 17 practice, and 1 review is a B-
- 22 experience, 13 labs, and 18 practice is a B

The Early Bird and Descriptive Commits bonuses are straight forward and set you up for success. Combined, they are also the same amount as the participation and lab bonuses, so getting a strong start and being detail oriented all semester can give you flexibility on attendance or labs.

Early Bird, Descriptive commits, Community Star, and Git-ing Unstuck are all equal to the half differnce between steps at a C or above. So earning any two can add a + to a C or a B for example:

- 22 experience, 13 labs, 18 practice, Descriptive Commits, and Early Bird is a B+
- 22 experience, 13 labs, 18 review, Descriptive Commits, and Early Bird is a C+

Skip to main content

without any extra work!

If you are missing learning badges required to get to a bonus, community badges will fill in for those first. If you earn the Participation, Lab, and Breadth bonuses, then remaining community badges will count toward the community bonus.

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.

The order of application for community badges:

- to make up missing experience badges
- to make up for missing review or practice badges to earn the breadth bonus
- to upgrade review to practice to meet a threshold
- toward the community badge bonus

To calculate your final grade at the end of the semester, a script will count your badges and logged event bonuses. The script will output a dictionary with keys for each type of learning badge and event bonus with a count for the value of each.

```
example_student = {'experience' :21, 'lab': 13, 'review': 6,'practice': 12,
                   'explore': 2,
                   'build' :1,
                 'community': 3,
                 'unstuck': 0,
                 'descriptive': 1,
                 'early': 1 }
```

Then these counts will go into the following function to calculate the final grade

Warning

This is not complete, but will be before the end of the penalty free zone.

```
# set up remaining constants (some are above)
bonus_criteria = {'participation_bonus': lambda r: int(r['experience'] >=22),
                  'lab_bonus':  lambda r: int(r['lab'] >=13),
                   'breadth_bonus': lambda r: int(r['review'] + r['practice']>=18),
                 'community_bonus': lambda r: int(r['community']>=10),
                 'unstuck_bonus': lambda r: r['unstuck'],
                 'descriptive_bonus': lambda r: r['descriptive'],
                 'early_bonus': lambda r: r['early'] }
bonus_values = {'participation_bonus': bonus_participation,
                  'lab_bonus':  bonus_lab,
                   'breadth_bonus': bonus_breadth,
                 'community_bonus': 18,
                 'unstuck_bonus': 9,
                 'descriptive_bonus': 9,
                 'early_bonus': 9 }
weights = learning_weights.copy()
weights.update(bonus_values)
community_thresh = {'experience':22,
                 'review':4,
                 'practice':7,
                 'review_upgrade':3}
community_cost = {'experience':3,
                 'review':4,
                 'practice':7,
                 'review_upgrade':3}

# compute grade

# def calculate_grade(badge_dict):
#     if badge_dict['community']>0:
        # apply community badges

    # add bonuses


    # final sum
    # sum([weights[cat]*count_dict[cat] for cat in count_dict.keys()])
```

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

Skip to main content

actually works.

Topics:

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

What Happens When I run code?

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

Recommended workload distribution

Note

General badge deadlines are on the detailed badge procedures page.

To plan your time, I recommend expecting the following:

- 30 minutes, twice per week for prepare work (typically not this much).
- 1.5(review)-3(practice) hours, twice per week for the dated badges (including revisions).

For each explore:

- 30 min for proposal
- 7 hours for the project

Not

the fi
take l
will b
ones

For each build:

- 1.5 hour for the proposal (including revisions)
- 22 hours for the project

Skip to main content

This is a four credit course, meaning we have approximately 4 hours of class + lab time per week($75 \times 2 + 105 = 255$ minutes or 4.25 hours). By the accredidation standards, students should spend a minimum of 2 hours per credit of work outside of class over 14 weeks. For a 4 credit class, then, the expected minimum number of hours of work outside of class you should be spending is 112 hours(2 * 4 * 14). With these calculations, given that there are 26 class sessions and only 18 review or practice are required, it is possible to earn an A with approximately 112 hours of work outside of class and lab time.

Tentative Timeline

Warning

This section is not yet updated for spring 2024.

This is a rough example.

This is the planned schedule, but is subject to change in order to adapt to how things go in class or additional questions that come up.

```python
import pandas as pd
pd.read_csv('schedule.csv',index_col='date').sort_index()
```

Skip to main content

date	question	keyword	conceptual	practical	social	activity
2023-09-07	Welcome, Introduction, and Setup	intro	what is a system, why study tools	GitHub basics	class intros	create kwl repo in github, navigate github.com...
2023-09-12	Course Logistics and Learning	logistics	github flow with issues	syllabus	working together and building common vocab	set up to work offline together, create a folder
2023-09-14	Bash intro & git offline	terminal start	git structure, paths and file system	bash path navigation, git terminal authentication	why developers work differently than casual users	navigate files and clone a repo locally
2023-09-19	How can I work with branches offline?	gitoffline	git branches	github flow offline, resolving merge conflicts	commuication is important, git can help fix mi...	clone a repo and make a branch locally
2023-09-21	When do I get an advantage from git and bash?	why terminal	computing mental model, paths and file structure	bash navigation, tab completion	collaboration requires shared language, shared...	work with bash and recover from a mistake with...
2023-09-26	What *is* a commit?	merge conflicts	versions, git vlaues	merge conflicts in github, merge conflicts wit...	human and machine readable, commit messages ar...	examine commit objects, introduce plumbing com...
2023-09-28	How do programmers communicate about code?	documentation	build, automation, modularity, pattern matching,	generate documentation with jupyterbook, gitig...	main vs master, documentation community	make a jupyterbook
2023-10-03	What *is* git?	git structure	what is a file system, how does git keep track...	find in bash, seeing git config, plumbing/porc...	git workflows are conventions, git can be used...	examine git from multiple definitions and insp...
2023-10-05	Why are these tools like this?	unix philosophy	unix philosophy, debugging strategies	decision making for branches	social advantages of shared mental model, diff...	discussion with minor code examples
2023-10-12	How does git make a commit?	git internals	pointers, design and abstraction, intermediate...	inspecting git objects, when hashes are unique...	conventions vs requirements	create a commit using plumbing commands
2023-10-17	What is a commit number?	numbers	hashes, number systems	git commit numbers, manual hashing with git	number systems are derived in culture	discussion and use hashing algorithm
2023-10-19	How can I release and share my code?	git references	pointers, git branches and tags	git branches, advanced fixing, semver and conv...	advantages of data that is both human and mach...	make a tag and release
2023-10-24	How can I automate things with bash?	bash scripting	bash is a programming language, official docs,...	script files, man pages, bash variables, bash ...	using automation to make collaboration easier	build a bash script that calculates a grade
2023-10-26	How can I work on a remote server?	server	server, hpc, large files	ssh, large files, bash head, grep, etc	hidden impacts of remote computation	log into a remote server and work with large f...

date

Skip to main content

date	question	keyword	conceptual	practical	social	activity
2023-10-31	What is an IDE?	IDE	IDE parts	compare and contrast IDEs	collaboraiton features, developer communities	discussions and sharing IDE tips
2023-11-02	How do I choose a Programming Language for a p...	programming languages	types of PLs, what is PL studying	choosing a language for a project	usability depends on prior experience	discussion or independent research
2023-11-07	How can I authenitcate more securely from a te...	server use	ssh keys, hpc system strucutre	ssh keys, interactive, slurm	social aspects of passwords and security	configure and use ssh keys on a hpc
2023-11-09	What Happens when we build code?	building	building C code	ssh keys, gcc compiler	file extensions are for people, when vocabular...	build code in C and examine intermediate outputs
2023-11-14	What happens when we run code?	hardwar	von neuman architecture	reading a basic assembly language	historical context of computer architecures	use a hardware simulator to see step by step o...
2023-11-16	How does a computer represent non integer quan...	floats	float representation	floats do not equal themselves	social processes around standard developents, ...	work with float representation through fractio...
2023-11-21	How can we use logical operations?	bitwise operation	what is a bit, what is a register, how to brea...	how an ALU works	tech interviews look for obscure details somet...	derive addition from basic logic operations
2023-11-28	What *is* a computer?	architecture	physical gates, history	interpretting specs	social context influences technology	discussion
2023-11-30	How does timing work in a computer?	timing	timing, control unit, threading	threaded program with a race condition	different times matter in different cases	write a threaded program and fix a race condition
2023-12-05	How do different types of storage work together?	memory	different type of memory, different abstractions	working with large data	privacy/respect for data	large data that has to be read in batches
2023-12-07	How does this all work together	review	all	end of semester logistics	group work final	review quiz, integration/reflection questions
2023-12-12	How did this semester go?	feedback	all	grading	how to learn better together	discussion

Tentative Lab schedule

```
pd.read_csv('labschedule.csv',index_col='date').sort_index()
```

Skip to main content

date	topic	activity
2023-09-08	GitHub Basics	syllabus quiz, setup
2023-09-15	working at the terminal	organization, setup kwl locally, manage issues
2023-09-22	offline branches	plan for success, clean a messy repo
2023-09-29	tool familiarity	work on badges, self progress report
2023-10-06	unix philosophy	design a command line tool that would enable a...
2023-10-13	git plumbing	git plumbing experiment
2023-10-20	git plumbing	grade calculation script, self reflection
2023-10-27	scripting	releases and packaging
2023-11-03	remote, hpc	server work, batch scripts
2023-11-10	Compiling	C compiling experiments
2023-11-17	Machine representation	bits and floats and number libraries
2023-12-01	hardware	self-reflection, work, project consultations
2023-12-08	os	hardware simulation

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

Skip to main content

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 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, Access, and Inclusion Services for Students Statement

Your access in this course is important. Please send me your Disability, Access, and Inclusion (DAI) 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 DAI, please contact them to engage in a confidential conversation about the process for requesting reasonable accommodations in the classroom. DAI can be reached by calling: 401-874-2098, visiting: web.uri.edu/disability, or emailing: dai@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 or email address associated with a commit 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

Skip to main content

> Assignments are tested against LLMs and designed so that they require you to learn and that an LLM answer will be low quality and not earn credit.

All of your work must reflect your own thinking and understanding. The written work in English 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. You may use auto-complete in all tools incuding, IDE-integrated GitHub co-pilot (or similar, IDE embedded tool) for any code that is required for this course because the code is necessary to demonstrate examples, but language syntax is not the core learning outcome.

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 to be the result of inappropriate collaboration or AI use, 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 the badge in question will be ineligible and your maximum number of badges possible to be earned will be limited according to the guidelines below per badge type (you cannot treat the plagiarized badge as skipped). If you are found to have violated the policy a second time, 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 practice* badge, the review and practice 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.

Viral Illness Precautions

The University is committed to delivering its educational mission while protecting the health and safety of our community. Students who are experiencing symptoms of viral illness should NOT go to class/work. Those who test positive for COVID-19 should follow the isolation guidelines from the Rhode Island Department of Health and CDC.

If you miss class, you do not need to notify me in advance. You can follow the makeup procedures.

Excused Absences

Absences due to serious illness or traumatic loss, religious observances, or participation in a university sanctioned event are considered excused absences.

You do not need to notify me in advance.

For short absences (1-2 classes) can follow the makeup procedures.

For this, email Dr. Brown when you are ready to get caught up and she will help you make a plan for the best order to complete missed work so that you are able to participate in subsequent activities. Extensions on badges will be provided if needed. In your plan, include what class sessions you missed by date.

Skip to main content

We understand that college comes with challenges and stress associated with your courses, job/family responsibilities and personal life. URI offers students a range of services to support your mental health and wellbeing, including the URI Counseling Center, MySSP (Student Support Program) App, the Wellness Resource Center, and Well-being Coaching.

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 week they will be made by BrightSpace too, but after that, all course activities will be only on GitHub.

Sign up to watch

Watch the repo and then, after the first class, claim a community badge for doing so, using a link to these instructions as the "contribution" like follows.

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

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	1:00 - 3:00	Zoom	Marcin
Monday	4:45-6:30	052 Tyler	Trevor
TBD	TBD	Zoom	Dr. Brown
TBD	TBD	134 Tyler	Dr. Brown
Tuesday	11-12:30	052 Tyler	Trevor
Wednesday	10:00 - 12:00	139 Tyler	Marcin
Wednesday	2:00 - 4:00	049 Tyler	Gyanko
Thursday	10:00 - 11:00	Zoom	Marcin
Thursday	11-12:30	052 Tyler	Trevor
Friday	2:00 - 4:00	049 Tyler	Gyanko
TBD	TBD	Zoom	TA
TBD	TBD	in person TBD	TA

Online office hours locations are linked on the GitHub Organization Page

Important

You can only see them if you are a "member" to join, make sure that you have completed Lab 0.

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 isses

- 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.

Skip to main content

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

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,
- Ms./Mrs. are not acceptable

1.2. Prismia

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

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

1.4. 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.

(see prismia or a member of the instructional team for the link to create your KWL if you are making up class)

1.5. We havea Glossary!!

For example, the term we used above:

Skip to main content

Tip

In class, on prismia, I will sometimes link like above, but you can also keep the page open if that is helpful for you.

In the course site, glossary terms will be linked as in the follwing list.

Key terms for the first class:

- repository
- git
- github
- PR

1.6. GitHub Docs are really helpful and have screenshots

1.7. First KWL setup steps

Imp

GitHu
am n
so I d
of rec

Today we did the following:

1. Accept the assignment to create your repo: KWL Chart
2. Edit the README to add your name by clicking the pencil icon (editing a file step 2)
3. adding a descriptive commit message and commited directly to main (editing a file step 5)
4. Edit the README to fill in one row of the table by clicking the pencil icon (editing a file step 2)
5. created a new branch (named `prior_knowledge`) (editing a file step 7-8)
6. added a message to the Pull Request (pull request step 5)
7. Creating a pull request (pull request step 6)
8. Clicking Merge Pull Request

1.8. GitHub Actions

GitHub allows us to run scripts within our repos, the feature is called GitHub Actions and the individual items are called workflows.

We ran the `Experience Reflection` action at the end of class.

1.9. Questions After Class

1.9.1. is there a limit to how many branches you can have from the main branch?

Not specifically. Technically there are limits, but they're second hand and we will see them later

1.9.2. what should I name the commit for the experience badge?

Skip to main content

1.9.3. When did Git and GitHub get created?

git was created in 2005. GitHub was founded in 2008.

1.9.4. Do I commit directly to the branch when I edit my experience badge?

yes

1.9.5. will these experience files be auto-generated each class?

You will run the action manually, but it will create and name the files for you.

1.9.6. Is making Pull Requests our way of submitting work?

Yes

1.9.7. What is the action doing

Copying a file, naming it with that day’s date, making a branch with today’s date, and opening a PR with a specific body and title.

1.9.8. How does GitHub actions work and what is it useful for?

This is a big picture question for the github docs to answer.

2. How does this course work?

Today we will:

- continue getting familiar with the structure of GitHub
- clarify more how the course will flow
- practice with new vocabulary

Tuesday was a lot of new information, today we will reinforce that mostly, and add only a little

2.1. Warm up

1. Navigate to your KWL repo
2. Find the issues tab
3. Open the prepare-2024-01-25 issue and follow those instructions

*hint: my KWL repo URL is: `https://github.com/compsys-progtools/kwl-sp24-brownsarahm`

Skip to main content

When you know something well:

- it becomes automatic, you can do it without thinking about the details
- you are able to anticipate things
- you use specialized vocabulary with ease

Important

The goal is to get you to that point with alal of the developer tools we will learn about.

If the tools become automatic in this class, you an *use* them with ease in other classes to your advantage.

Note

We did some prismia questions on GitHub things to reinforce that material we had already seen. When we do this, I'll typically skip that section in the posted notes, but you can always view on prismia including getting a transcript from prisma:

1. In the top left corner there is a `>` icon which opens a sidebar popup.
2. In the top left opens a menu
3. "Get transcript from class"

2.3. Pull Requests and Commits

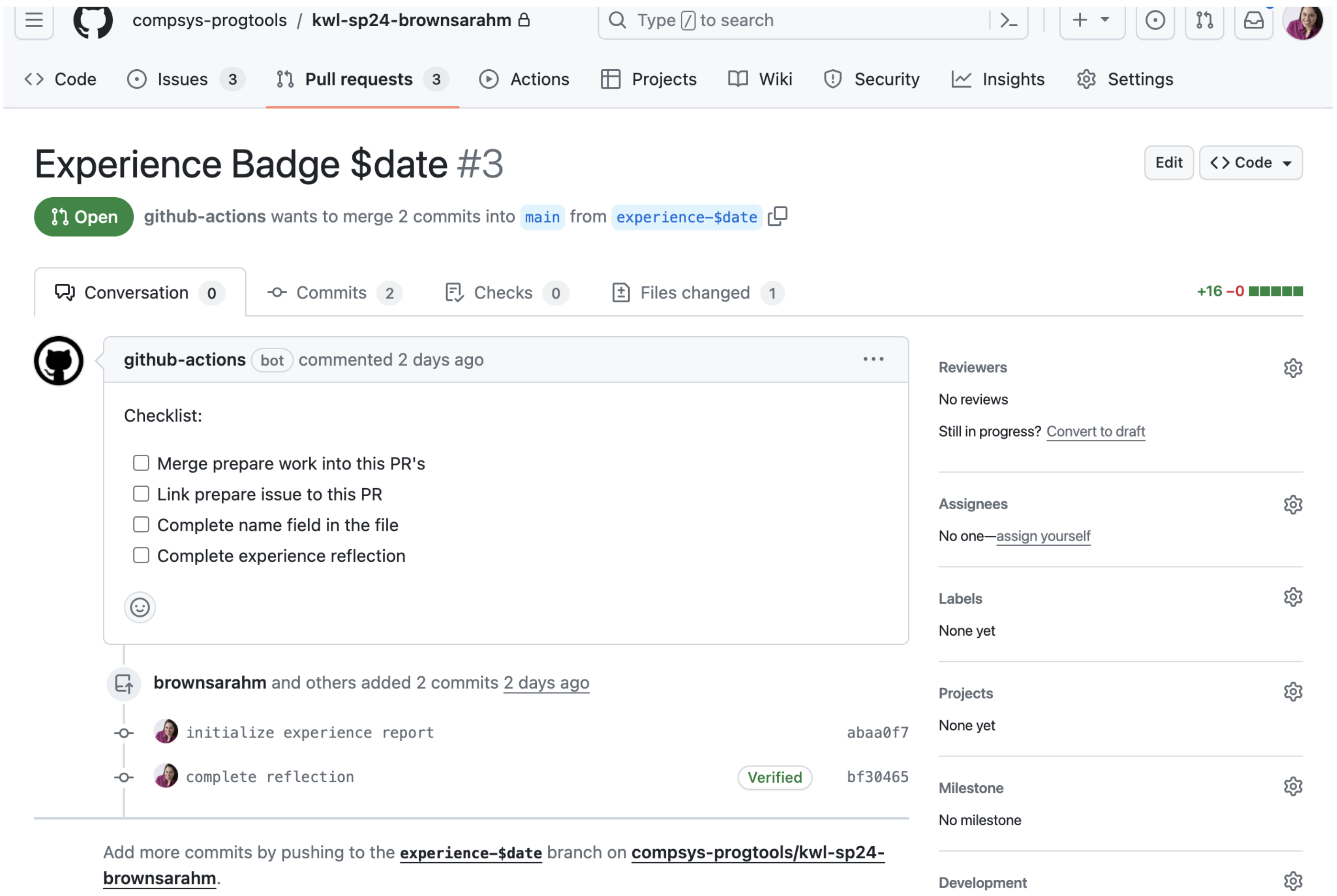
The unit of changes in a histoy is a commit. Each commmit is said to be "on" a branch.

We examined the PRs for your first experience badge.

Not

We w
and c
is a *u*
if it is
imple

Skip to main content



Experience Badge $date #3

Open github-actions wants to merge 2 commits into `main` from `experience-$date`

github-actions bot commented 2 days ago

Checklist:

- [ ] Merge prepare work into this PR's
- [ ] Link prepare issue to this PR
- [ ] Complete name field in the file
- [ ] Complete experience reflection

brownsarahm and others added 2 commits 2 days ago

```
initialize experience report          abaa0f7
complete reflection        Verified   bf30465
```

Add more commits by pushing to the **`experience-$date`** branch on **compsys-progtools/kwl-sp24-brownsarahm**.

Fig. 2.1 A PR that has two commits on it with one file changed. This PR compares the `experience-$date` branch to the `main` branch.

From this page we can learn an important implication of the fact that **a pull request is a comparison between two branches**. Since the relationship is between the branches, it is not related directly to commits. We can *add* more commits to the branch after we make a PR, and they show up on the PR page.

At the bottom of this screenshot which is right above the checks for the PR is a reminder from GitHub:

> Add more commits by pushing to the experience-$date branch on compsys-progtools/kwl-sp24-brownsarahm.

Note

here we are learning *by example* and then *synthesizing* that into concrete facts.

This course will proceed like this a lot; we *do* things to let you experience the important to know things *in context* first, then discuss and label and examine.

Hint

If we ever do something and you are not sure why, please ask!!

Skip to main content

> just play with it

- common attitude in CS
- not optimal for learning

It’s not optimal for learning because it can leave you unguided and not knwo where to start, but it is the common culture in developer environments.

My goal is to teach you to thrive within that culture, because you are likely to encounter it at work.

My goal, however, is not to just drop you in with no preparation. If it feels that way, **please ask questions**.

We’ll practice what it *looks like* to learn by tinkering and then examining.

Imp

Do no
quest
quest
payin

To do this:

- set up opportunities for you to *do* the things that give you the opportunity
- highlight important facts about what just happened
- ask you questions to examine what just happened

This is why attendance/participation is a big part of your grade.

Experience badges are evidence of having learned, you get credit for engaging in the *process* of learning.

2.5. Admin

1. Go to you PR tab
2. Find the PR titled “Template updates” that I created
3. Merge and confirm the PR

2.6. Commit history & PRs

When we merge a PR into `main` all of the commits on the PR compare branch are added to `main` + an additional commit for the merge itself.

For easy merges, that commit does not add more changes to any files, but if the merge is not automatic, it can.

2.7. 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

- **No need to tell me in advance**
- For 1 class, no need to tell me why at all
- For 1 class, make it up and keep moving
- For long absences, I will help you plan how to get caught up, must meet university criteria for excused absence

If you do email me about missing a single clss, I will likely not reply. Not because I do not care about your long term success; I do! I get too many emails. I try to prioritize time on things that has the biggest impact; confirming I saw an email that does not change any other policies is lower impact than, for example giving feedback on student work.

2.7.2. My focus is for you to learn

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

Important

You **will** be asked to revise things at some point, for 2 reasons:

- since I allow revisions, I enforce really high standards on the quality of your work so that you do not hold onto misconceptions
- the process of using GitHub to make revisions is something to get familiar with.

2.7.3. 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

2.8. 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

2.9. 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 do not add value for a company, so why would they pay you?

Skip to main content

- 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

2.11. “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 (just) about the facts, but the processes

2.12. How does this work?

2.12.1. In class:

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

2.12.2. Outside of class:

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

2.13. How to be successful

Important

There are links on the side to advice from previous semesters, details in the grading have changed, but the core is still the same.

I give a time breakdown in the syllabus.

Take a minute to think about how you use your time and what that breakdown means for how you will plan.

Skip to main content

make a plan that will work for you. I want this to be flexible enough that you can adapt it to your liking, but that mean you have to make choices.

2.14. 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?

We will:

- 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).

2.14.1. How it fits into your CS degree

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

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

(for BS) 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.

2.14.2. Programming is Collaborative

There are two very common types of collaboration

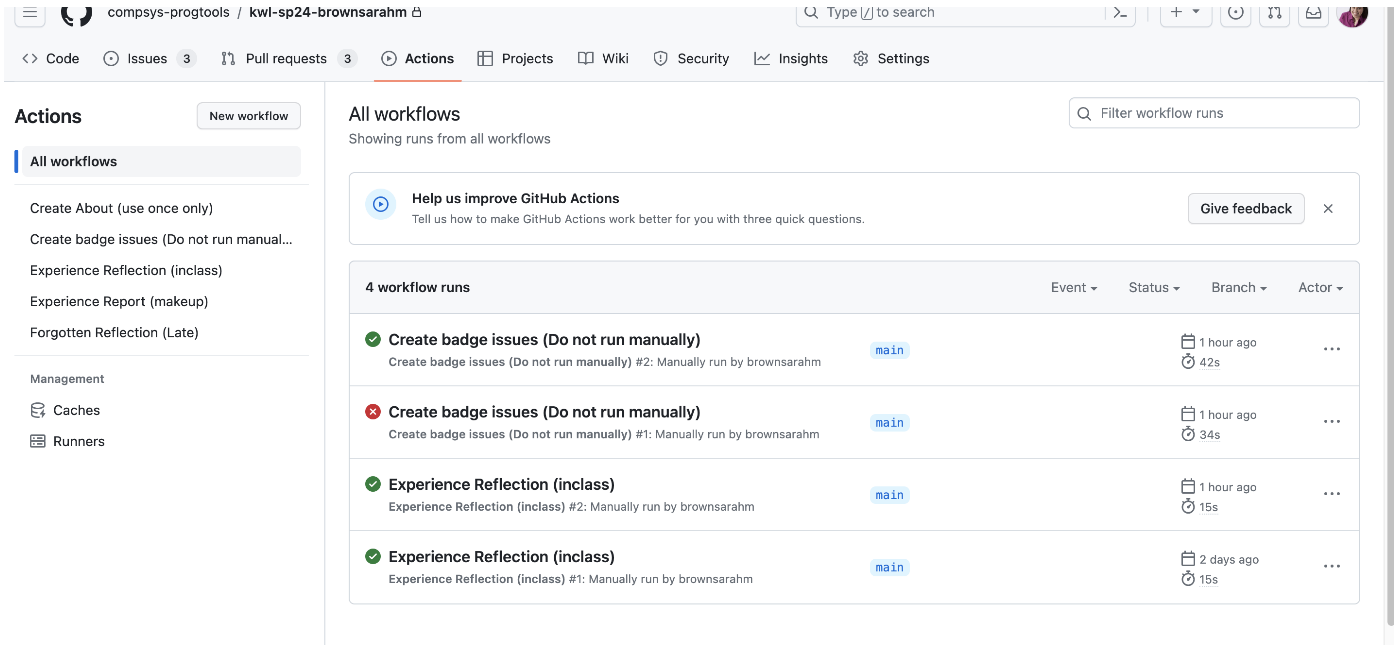
- code review (working independently and then reviewing)
- pair programming (sitting together and discussing while writing)

We are going to build your skill in the *code review* model, especially.

You can do build badges collaboratively, for a closer collaboration, but those are your choice.

2.15. GitHub Actions Tab Review

GitHub allows us to run scripts within our repos, the feature is called GitHub Actions and the individual items are called workflows.


compsys-progtools / kwl-sp24-brownsarahm

Code | Issues 3 | Pull requests 3 | Actions | Projects | Wiki | Security | Insights | Settings

Actions — New workflow

All workflows

- Create About (use once only)
- Create badge issues (Do not run manual...
- Experience Reflection (inclass)
- Experience Report (makeup)
- Forgotten Reflection (Late)

Management

- Caches
- Runners

All workflows
Showing runs from all workflows

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.
Give feedback

4 workflow runs — Event, Status, Branch, Actor

Create badge issues (Do not run manually)
Create badge issues (Do not run manually) #2: Manually run by brownsarahm — main — 1 hour ago — 42s

Create badge issues (Do not run manually)
Create badge issues (Do not run manually) #1: Manually run by brownsarahm — main — 1 hour ago — 34s

Experience Reflection (inclass)
Experience Reflection (inclass) #2: Manually run by brownsarahm — main — 1 hour ago — 15s

Experience Reflection (inclass)
Experience Reflection (inclass) #1: Manually run by brownsarahm — main — 2 days ago — 15s

Fig. 2.2 Screen shot of the actions tab of my repo showing 4 total runs in the center panel. Three of the runs succeeded and have a green check mark. One failed and has a red x. On the left hand side there is a list of 5 possible actions to run. Notice the list of workflows on the left each has a unique name (because we generally choose to name things so that names can be used as identifies) the list of workflow runs includes the same name multiple times because we can *run* each workflow multiple times.

this should be different from yours, because I tested things in mine before making your PRs

2.15.1. Get Credit for Today's class

**Run your Experience Reflection (inclass) action on your kwl repo **

talk with peers to make sure you remember what the right way to click on it is

2.16. Reminder about class structure

flowchart TD exp[Experience badges] dated[Dated Badges] rev[Review badges] pra[Practice badge] inclass[Class Session <br> 26 total] exp --> | tracks participation for each| inclass rev --> |is one type of| dated pra --> |is one type of| dated breadth[Breadth bonus] breadth --> |awarded for 18 total | dated participation[Participation bonus] participation --> |awarded for 22 total| exp dated --> |posted for each| inclass
This is called a concept map, you would read it along the arrows, so this corresponds to the following bullets:

- review badges are one type of dated badge
- practice badges
- dated badges are posted for each class session
- experience badges track participation for each class session

Hin

This

that y

badg

Skip to main content

Remember, this website is generated from a GitHub repo, you can find it from the course organization page on GitHub. The organization is named `compsys-progtools` so the org page is at `https://github.com/compsys-progtools` and it is the owner of your KWL repo so there is a link to it in the top left corner.

2.17. Prepare for lab

this is really a tip/hint about what we will do in the next lab

Next lab we will give you help with sorting out the procedures of PRs and issues and submitting work.

If you work on the badges, but you are not sure about things, log your work as comments on the issue or even in a separate file somewhere, and we will help you get it submitted in lab.

You can also bring questions about anything in the syllabus to get help (or post them for written answers).

2.18. 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, zsh, powershell, GitHub. Make a diagram using mermaid to highlight how these terms relate to one another
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.

Example "venn diagram " with mermaid subgraphs

flowchart subgraph Browsers subgraph Safari end subgraph Chromium based gg[Google Chrome] me[Microsoft Edge] end end

2.19. Badges

Review Practice

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 or what you know about git/GitHub so far and add it to your kwl chart repo in a file called prior-knowledge-map.md. Use mermaid

2.20. Questions After Class

2.20.1. In experience badge, after I commit, do I do anything after?

Yes, request a review from `@instructors`

2.20.2. Are we adding a reviewer to our experience badge?

Yes

Skip to main content

that?

In the future, you will, but we'll work through what that looks like in more detail next week.

2.20.4. do we do both the review and prepare?

Only one per date. If you look, they're very similar. For the first class, they're just about the same. As we go on for the first few weeks the difference between them will increase (which is mostly complexity or depth of understanding that I am looking to check)

2.20.5. How do I submit work?

Create a PR and request a review.

Two things to note:

- *in the penalty free zone, there is no penalty for doing it wrong*
- checking this is the main topic of the next lab, so we will one on one check in with each of you that you know how to do this

2.20.6. If we wanted to ping an instructor/the instructor that got assigned to us, would we @ them in a comment?

Yes! you can @ mention any of us or the team whenever you need help.

2.21. A final note

Tip

Reading to the end is always valued, you can claim a community badge for finding this by linking to the heading above and requesting a review from @brownsarahm, title your PR `community-reading-notes`.

Extra bonus, there are tips like this throughout the website that let you get community badges for setting things up, or reading carefully.

3. Working offline

3.1. Why do we need this for computer systems?

3.1.1. Computer Systems are designed by people

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, most of it is essentially an imaginary world that is 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.

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.

3.1.2. 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.

3.1.3. 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.

3.2. Let’s get organized

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

We will start using the terminal today, by getting all set up.

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`

On mac you can optionally use the `bash` command to switch to bash.

```
bash
```

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
```

Then it tells you Apple’s default is somethign else and how to switch back.

We can use `pwd` you can see your current “location”. It stands for **p**rint **w**orking **d**irectory.

```
pwd
```

this is called the path and specifically this is an absolute path, we can tell because it starts with `/` that is the “root” or “top” of the directory tree structure.

Skip to main content

Tip

You can think of absolute paths like a street address. It describes the location in a general (absolute) way; not relative to another location.

I can say Tyler hall is at 9 Greenhouse Road and you can get there from anywhere.

In contrast if I say it is that building in the distance, across the quad past the large glass building, that works from our classroom in Ranger, but not from the library, or anywhere off campus. This would be a relative set of directions to get to Tyler Hall.

these

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

```
cd Documents/
```

We can use a the relative path to describe where we want to change to. The `Documents/` is a relative path because it does not start with `/`

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)

3.3. Your Home directory is easy to get to

Next we will use `cd` without path at all.

```
cd
```

Here we noted that the prompt changed back to what we started with, the `~`. The home directory.

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

Remember, you can use `tab` to completed. So I typed: `cd Doc` `tab` `in` `tab` `sys` `tab`

This saves time **and** reduces typos, the terminal is filling in based on what the available options are, what exists in your filesystem, it will not fill in with something that is not there.

3.4. Relative paths

Let's look at our current location again:

```
pwd
```

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

`..` is a special file that points to a specific relative path, of one level up.
We can use `..` to go up one level from wherever we are.

```
cd ..
```

we can see the impact using `pwd`

```
pwd
```

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

Notice that before the last part was systems and the new path is missing that last part.

We will go back to the folder we made, since that is where we want to work for class.

```
cd systems/
```

3.5. 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)

Then find your readme issue and note its number, and copy the template for the README.

From the code tab, create a new file called README.md and paste in the template.

Commit the changes directly to main with `start readme closes #` and the number of your issue as the message, mine was `start readme closes #3`.

This will close the issue.

3.6. Connecting with GitHub

We have two choices to Download a repository:

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

we want option 1 beacuse we are learning git

3.6.1. 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

3.6.2. Windows (gitbash)

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

3.6.3. MacOS X

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

3.6.4. 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`

3.6.5. Cloning a repo

Skip to main content

```
git clone https://github.com/compsys-progtools/gh-inclass-sp24-brownsarahm.git
```

then we get several messages back from git and GitHub (the remote, it *could* be a different host and a repo can have multiple remotes)

```
Cloning into 'gh-inclass-sp24-brownsarahm'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 8 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
```

You c
becau
my re
URL

You g
buttor
repo.
Local

We can see that it made a new folder by looking at what is in our folder.

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

```
ls
```

```
gh-inclass-sp24-brownsarahm
```

It was empty and now it has your repo!

3.7. What is in a repo?

We can enter that folder

```
cd gh-inclass-sp24-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. Hidden files are not protected, they are just lightly hidden to protect a casual user from accidentally editing them. As a developer, you will likely edit hidden files a lot.

In this case, the hidden files are **for** GitHub's server, not for local use.

We can actually see the rest with the `-a` for **all** option or *flag*. Options are how we can modify how a command line program works, mostly they are optional or nonrequired.

```
ls -a
```

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

Skip to main content

There is also the `.git` directory. This folder is for use by the git program. Most simple programs we write in school, run, store all of their values in variables in memory and thn when you restart you make new values. git, as a program uses that `.git` directory to store all of the information it needs in files.

We also see some special “files” that we will always see in every location:

- `.` the current location
- `..` up one directory

3.8. How do I know what git knows?

`git status` is your friend.

Let’s see how it works:

```
git status
```

```
On branch main
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`)

3.9. Making a branch with GitHub.

On your about issue on Github.com, 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.

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/compsys-progtools/gh-inclass-sp24-brownsarahm
 * [new branch]      2-create-an-about-file -> origin/2-create-an-about-file
```

Skip to main content

```
git status
```

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

nothing to commit, working tree clean
```

Looks like nothing so far.

Next, we switch to that branch.

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

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

and verify what happened

```
git status
```

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

nothing to commit, working tree clean
```

Now it shows us the new branch!

3.10. Creating a file on the terminal

The `touch` command creates an empty file

```
touch about.md
```

```
git status
```

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

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

We put some content in the file, any content then saved and exit.

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

and we can look at the contents of it.

Now we will check again with git.

```
git status
```

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

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

We will use `.` as our "file" to stage everythign in the current working directory.

```
git add .
```

And again, we will check in with git

```
git status
```

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

Skip to main content

```
        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 about file close s #2'
```

```
[2-create-an-about-file 81c6f18] create about file close s #2
 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

Imp

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

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

nothing to commit, working tree clean
```

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

We can send them to github with `git push`

```
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), 337 bytes | 337.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
   1aef0a1..81c0f18  2-create-an-about-file -> 2-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

3.11. Concept Overview

Note

In the notes from last class I described what this type of diagram is called and how to read it.

Important

This is not the only thing that would be correct for the prep work, in the next prepare, you can see if yours *conceptually* matches this even if visualized differntly.

flowchart TD shell --> |is used through| terminal zsh --> |is an example of a | shell bash --> |is an example of a | shell powershell --> |is an example of a | shell shell --> |interface to | os[Operating system] git --> |is a program used through | terminal
Another way to think abut things (and adds some additional examples to help you differentiate between categories and examples of categories)

flowchart LR subgraph programs subgraph shell bash zsh powershell end subgraph vcs[version control systems] git svn mercurial end end subgraph gh[git hosts] GitHub BitBucket GitLab end
Today's bash commands:

command	explanation
`pwd`	print working directory
`cd <path>`	change directory to path
`mkdir <name>`	make a directory called name
`ls`	list, show the files
`touch`	create an empty file

We also learned some git commands

Not

It is s
alt tex
not w

they p
reade
most
intere
the is

Skip to main content

command	explanation
`status`	describe what relationship between the working directory and git
`clone <url>`	make a new folder locally and download the repo into it from url, set up a remote to url
`add  <file>`	add file to staging area
`commit -m 'message'`	commit using the message in quotes
`push`	send to the remote

3.12. 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, zsh, powershell, GitHub. Make a diagram using mermaid to highlight how these terms relate to one another
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.

Example "venn diagram " with mermaid subgraphs

flowchart subgraph Browsers subgraph Safari end subgraph Chromium based gg[Google Chrome] me[Microsoft Edge] end end

3.13. Badges

Review Practice

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 comment. Be sure it is not inside another repository.
3. Try using setting up git using your favorite IDE's git integration (not its terminal) or GitHub Desktop. Make a file gitgui.md and include some notes of how it went. Give the file a heading like `# Setting up <tool name>`, with the actual tool name you setup in the title (eg Github Desktop or VSCode source control panel or …) Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent to the terminal or does it use different terms?

3.14. Experience Report Evidence

If you missed class today, link to your gh-inclass repo in the Experience report PR.

3.15. Questions After Today's Class

3.15.1. can you link an issue to both commits and PRs, or only PRs

Skip to main content

the about.md file.

3.15.2. if we want to stage multiple files, can we do that in one command, like git add about.md README.md

Yes! or you can use `.`

3.15.3. do we do anything with the pull request we just made in the in-class repo?

Leave the about PR open for class on Thursday

3.15.4. Is mermaid something we will be using just for now, or should I become more familiar with it?

It is useful and we will use it on an off throughout class. Since GitHub supports it in all markdown, including comments, it is handy to be able to put in a quick diagram to explain something.

If you like it, it is something you can use even more, for a possible explore badge.

3.15.5. What are the benefits of using GitHub locally

Using git and GitHub locally means you can use your regular devtools and use GitHub as a could copy of your work.

4. How do branches work?

4.1. Review

Note

This part of the notes is the key facts of the questions we did, not the questions, you can reivew the actual questions on Prismia

- `cd` with no path changes to home (`~`)
- absolute paths start with `/` or `C:\` or similar on Windows outside of bash
- absolute paths always work, like an address; relative paths are starting point specific, like directions
- `.` is a special file that references the current location, for example, we can stage all of the files in the working directory with `git add .`

4.2. Back to the gh-inclass repo

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

First we will navigate to our working directory for class that we made in the last class

```
pwd
```

```
/Users/brownsarahm
```

Now that we know where we are, we can move to the target location

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

and look at the location again to confirm it is what we expected

```
pwd
```

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

Not

we us
home
us the

Know
the ro
error

Using
increa
other

Tip

In class, I demod the fact about `cd` without a path going to home and then used my built in history to repeat a command by using the `&#8593;` key

Then we move into the gh inclass repo

```
cd gh-inclass-sp24-brownsarahm/
```

and confirm the path

```
pwd
```

```
/Users/brownsarahm/Documents/inclass/systems/gh-inclass-sp24-brownsarahm
```

4.3. Branches do not sync automatically

First, we will look at where we left off to refresh, then we will make a change in GitHub and see how that impacts our local copy.

We start examining our local copy by looking at the working directory

```
ls
```

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

we see both files like we expected.

Skip to main content

```
git status
```

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

nothing to commit, working tree clean
```

and everything is logged in git.

```
Opening github.com/compsys-progtools/gh-inclass-sp24-brownsarahm in your browser.
```

In the browser, we merged the PR we opened on Tuesday, that updates the main branch on GitHub with 2 more commits:

- the commit that was on the `2-create-an-about-file`
- the merge commit

First we check,

```
git status
```

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

nothing to commit, working tree clean
```

merging on GitHub does not change anythign locally

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

```
git cehkout main
```

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

The most similar command is
        checkout
```

Now, I spell it correctly and actually switch

```
git checkout main
```

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

It says we are up to date with `origin/main` and remember `origin` is the *name* of the remote that is where we cloned from on GitHub.com.

Now, we look at the working directory.

Get
sys

This
CLI w

It is fi
not w
brows

If you
figure
that a
faq pa

Imp

Note
chec
me w
to my

Skip to main content

```
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.

We can also confirm that it is not hidden:

```
ls -a
```

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

4.4. Updating Locally

Updating locally involves 2 steps:

- update the local repo (the .git directory with all of the commits)
- update the working directory

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), 919 bytes | 919.00 KiB/s, done.
From https://github.com/compsys-progtools/gh-inclass-sp24-brownsarahm
   faef6af..1e2a45f  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").

Now, again, we check the working directory

```
ls
```

```
README.md
```

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

```
git status
```

```
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

Now, git knows that our local main is behind `origin/main` and by how much and tells us how to apply the changes to the local branch.

So we do that.

```
git pull
```

```
Updating faef6af..1e2a45f
Fast-forward
 about.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 about.md
```

It applies the changes to the local directory too:

```
ls
```

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

4.5. What other tools do I have to examine things on the terminal

Lets try two more ways of looking at our repo and files. Then we will use those to learn more about working with branches.

4.5.1. Git log

We can see commits with `git log`

```
git log
```

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

```
commit 1e2a45fbca5ce7bf775827f5f4dbe23b6561cff4 (HEAD -> main, origin/main, origin/HEAD)
Merge: faef6af 81c6f18
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 1 12:51:17 2024 -0500

    Merge pull request #4 from compsys-progtools/2-create-an-about-file

    create about file close s #2
```

```
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Jan 30 13:33:54 2024 -0500

    create about file close s #2

commit faef6af98e6bd0951ebcefb809ff4e353a0c7fbc
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Jan 30 13:03:36 2024 -0500

    start readme, closes #3

commit 98cff657d25adf9a0820a04d60f6409445f70e76
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Tue Jan 30 17:53:54 2024 +0000

    Initial commit
```

4.5.2. Concatenate a file

the `cat` command *concatenates a files' contents to the terminal output (which is actually a special file called standard out or stdout)

```
cat about.md
```

```
I like to ski
```

4.6. Making a branch locally

Important

It is totally okay to test ideas out and see what happens as a way to investigate how things work. Here we tested a few suggestions to see what works. This lets us learn 2 things:

- what works
- how to read error messages

4.6.1. Checkout requires the branch to exist

```
git checkout my_branchcehcekd
```

```
error: pathspec 'my_branchcehcekd' did not match any file(s) known to git
```

This error message says that it cannot switch because the name we gave does not already exist. This is a good to recognize error message because this is also what would happen if you tried to switch, but spelled the branch name wrong.

4.6.2. the -b option lets checkout create a branch

Skip to main content

```
git checkout -b my_branch_cehckoutb
```

```
Switched to a new branch 'my_branch_cehckoutb'
```

Success!

the `-b` lets `git checkout` both create and switch to a branch

4.6.3. create is not an option

```
git branch create example
```

```
fatal: not a valid object name: 'example'
```

This error message is tricky; it only complains about `example`. We could try

```
git branch create
```

this gives no response, but if we use `git branch` we would see it created a branch called `create`.

So, back to our original attempt, is that it thinks we are asking to make a branch named `create` but then sees this extra thing `example` that it doesn't know what to do with, so blames that for the error.

4.6.4. Chaining commands together

Finally, we tried:

```
git branch my_branch; git checkout my_branch
```

```
Switched to branch 'my_branch'
```

This worked. From this example, we learn a 2 things:

- `git branch <new name>` creates a new branch
- we can put multiple commands on a "single line" using `;` between them

4.6.5. We can list branches

without any argument, we get a list of the branches that exist`

```
git branch
```

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

note it also indicates which branch you are on

4.7. Branches are pointers

We'll go back to main

```
git checkout main
```

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

and then make a new branch for the next changes we will make.

We've used `git checkout` to switch branches 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'
```

If we use `git log` to see the commits, we can see more about the branches.

```
git log
```

```
commit 1e2a45fbca5ce7bf775827f5f4dbe23b6561cff4 (HEAD -> fun_fact, origin/main, origin/HEAD, my_branch_cehck
Merge: faef6af 81c6f18
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Thu Feb 1 12:51:17 2024 -0500

    Merge pull request #4 from compsys-progtools/2-create-an-about-file

    create about file close s #2

commit 81c6f187f146caaaf43d97bc1bb8ed237142f4c3 (origin/2-create-an-about-file, 2-create-an-about-file)
Author: Sarah M Brown <brownsarahm@uri.edu>
Date:   Tue Jan 30 13:33:54 2024 -0500

    create about file close s #2

commit faef6af98e6bd0951ebcefb809ff4e353a0c7fbc
Author: Sarah Brown <brownsarahm@uri.edu>
Date:   Tue Jan 30 13:03:36 2024 -0500

    start readme, closes #3

commit 98cff657d25adf9a0820a04d60f6409445f70e76
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Tue Jan 30 17:53:54 2024 +0000

    Initial commit
```

branches are pointers a branch points to a commit.

Skip to main content

4.8. Linking a locally created branch to a remote

Next we will edit the file so we can create a commit.

```
nano about.md
```

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.

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.

```
cat about.md
```

```
I like to ski
I came to URI in 2020
```

My file now has 2 lines in it.

Now we check the status

```
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")
```

Now the file is *modified* instead of what we saw before that was *untracked*

Again we stage it first:

```
git add about.md
```

Tip

Git re

then look again

```
git status
```

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

Skip to main content

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

```
[fun_fact 07897fd] add a fun fact
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Then we push

```
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
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

but it fails.

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.

We follow git’s advice

```
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% (2/2), done.
Writing objects: 100% (3/3), 287 bytes | 287.00 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/compsys-progtools/gh-inclass-sp24-brownsarahm/pull/new/fun_fact
remote: 
To https://github.com/compsys-progtools/gh-inclass-sp24-brownsarahm.git
 * [new branch]      fun_fact -> fun_fact
branch 'fun_fact' set up to track 'origin/fun_fact'.
```

and success!

4.9. Merge conflicts

We are going to *intentionally* make a merge conflict here.

This means we are learning two things:

- what *not* to do if you can avoid it
- how to fix it when a merge conflict occurs

people did two types of work that were supposed to be independent, but turned out not to be.

To create the merge conflict we are going to edit the same branch in two different ways locally and in browser. This couls also happen by two different people making edits to the same branch *or* more commonly, by having edits made on both the compare branch and the base branch of a PR to the same part of the same file.

4.9.1. Edit in browser

First, in your browser edit the about.md file to have a second fun fact. Commit directly to the `fun_fact` branch.

4.9.2. Edit locally

Then edit it locally to also have 2 fun facts.

```
nano about.md
```

My local version looks like this:

```
cat about.md
```

```
I like to ski
I came to URI in 2020
I moved to RI in 2018
```

Now we check with git

```
git status
```

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

and stage and commit

```
git add .
git commit -m 'another funfact'
```

```
[fun_fact 7985615] another funfact
 1 file changed, 1 insertion(+)
```

Skip to main content

```
git pull
```

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 938 bytes | 312.00 KiB/s, done.
From https://github.com/compsys-progtools/gh-inclass-sp24-brownsarahm
   07897fd..7e78493  fun_fact   -> origin/fun_fact
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 are divergent. This illustrates the fact that our two versions of the branch `fun_fact` and `origin/fun_fact` are two separate things.

diverged means like this diagram:

```
gitGraph
    commit id:"A"
    commit id:"B"
    branch fun_fact
    checkout fun_fact
    commit id:"C"
    branch origin/fun_fact
    checkout origin/fun_fact
    commit id:"D"
    checkout fun_fact
    commit id:"E"
```

Note that the `origin/fun_fact` and `fun_fact` have different histories.

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.

diverged does not necessarily mean a conflict, sometimes we can simply change what commit has what parent and it will work.

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

Rebase will try to make it like this:

gitGraph commit id:"A" commit id:"B" branch fun_fact checkout fun_fact commit id:"C" commit id:"D'" checkout fun_fact commit id:"E'"
the commits `D` and `E` get modified some, the time and parent info is updated, but the changes to the content are the same.

```
git pull --rebase
```

Skip to main content

```
Auto-merging about.md
CONFLICT (content): Merge conflict in about.md
error: could not apply 7985615... another funfact
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 7985615... another funfact
```

It was not able to do it; beacuse both had edited the same line of the same file. This requires a person to figure out what the content should be.

it tells us what file the issue is in the 2nd line.

4.9.4. Resolving a merge conflict

To resolve manually, we have to edit the file. We are going to use nano again here.

```
nano about.md
```

The file looks like this when we open it up:

```
I like to ski
I came to URI in 2020
<<<<<<< HEAD
I am from NH
=======
I moved to RI in 2018
>>>>>>> 7985615
```

git added markup to the file to show us what was in each of the two versions, labeled by the HEAD pointer and the specific commit for the other. It puts `=` between the two. To resolve, we edit the file to be what we want. We can choose one version, the other, or both. We'll choose both here.

```
cat about.md
```

```
I like to ski
I came to URI in 2020
I am from NH
I moved to RI in 2018
```

Next, we go back to check in with git.

```
git status
```

```
interactive rebase in progress; onto 7e78493
Last command done (1 command done):
   pick 7985615 another funfact
No commands remaining.
You are currently rebasing branch 'fun_fact' on '7e78493'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)
```

Skip to main content

```
  (use "git add <file>..." to mark resolution)
	both modified:   about.md

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

it tells us how to finish up what we were doing.

First we commit:

```
git add .
git commit -m 'resolve conflit'
```

```
[detached HEAD 07959c0] resolve conflit
 1 file changed, 3 insertions(+), 1 deletion(-)
```

then use the `continue` option

```
git rebase --continue
```

```
Successfully rebased and updated refs/heads/fun_fact.
```

Finally we check in:

```
git status
```

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

nothing to commit, working tree clean
```

and 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), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/compsys-progtools/gh-inclass-sp24-brownsarahm.git
   7e78493..07959c0  fun_fact -> fun_fact
```

4.10. Prepare for Next Class

1. Add file called terminal-vocab.md on a branch linked to this issue. Fill in the template below:

```
## Before

<insert your diagram from 2024-01-30 prepare>

## Reflection

<!-- write a few bullets assessing your diagram above based on the notes from 2024-01-30. was anything in it


## Final thoughts

<!-- write a few bullets/sentences on how trying to make the diagram did/not help you think about the terms
```

4.11. Badges

Review Practice

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.

4.12. Summary

- branches do not synch automatically
- branches are pointers to commits
- every commit knows its parents
- if two different commits have the same parent, when we try to merge we will have divergent branches
- divergent branches can be merged by different strategies
- a merge conflict occurs if, when merging branches, a single file has been edited in two different ways

We often visualize git using graphs like subway maps:

gitGraph commit id:"A" commit id:"B" branch fun_fact checkout fun_fact commit id:"C" commit id:"D"' checkout fun_fact commit id:"E"
However you can also think of what we learned today like this:

flowchart LR cA[A <br> commit] cB[B <br> commit] cB --> |knows its parent| cA cC[C <br> commit] --> |knows its parent| cB blocal[fun_fact <br> branch] --> |is a pointer to| cC bremote[origin/fun_fact <br> branch] --> |is a pointer to| cB
Over the next few weeks we will keep refining this understanding.

4.12.1. New bash commands

Skip to main content

command	explanation
`cat`	concatenate a file to standard out (show the file contents)

4.12.2. New git commands

command	explanation
`git log`	show list of commit history
`git branch`	list branches in the repo
`git branch new_name`	create a `new_name` branch
`git checkout -b new_Name`	create a `new_name` branch and switch to it
`git pull`	apply or fetch and apply changes from a remote branch to a local branch

4.13. Experience Report Evidence

run the command:

```
git log >> makeup-2024-02-02.md
```

then `mv makeup-2024-02-02.md <rel>` where `<rel>` is the relative path from your gh incalss repo to your KWL repo.

commit this file to the experience report branch

4.14. Questions After Today’s Class

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

Skip to main content

```
# KWL Chart


<!-- replace the  _ in the table or add new rows as needed -->

| Topic | Know | Want to Know | Learned |
| ------| ------- | ------ | ------- |
| Git | _ | _ | _ |
| GitHub | _ | _ | _ |
| Terminal | _ | _ | _ |
| IDE | _ | _ | _ |
| text editors | _ | _ | _ |
|file system | _ | _ |_ |
|bash | _ | _ | _ |
|abstraction | _ | _ | _ |
|programming languages | _ | _ | _ |
|git workflows | _ | _ | _ |
| git branches | _ | _ | _ |
| bash redirects | _ | _ | _ |
|number systems | _ | _ | _ |
| merge conflicts | _ | _ | _ |
| documentation | _ | _ | _ |
| templating | _ | _ | _ |
|bash scripting | _ | _ | _ |
| developer tools | _ | _ | _ |
| networking | _ | _ | _ |
|ssh | _ | _ | _ |
| ssh keys | _ | _ | _ |
|compiling | _ | _ | _ |
| linking   | _ | _ | _ |
| building | _ | _ | _ |
| machine representation  | _ | _ | _ |
| integers   | _ | _ | _ |
| floating point  | _ | _ | _ |
|logic gates | _ | _ | _ |
| ALU | _ | _ | _ |
| binary operations | _ | _ | _ |
| memory | _ | _ | _ |
| cache | _ | _ | _ |
| register | _ | _ | _ |
| clock | _ | _ | _ |
| Concurrency | _ | _ | _ |
```

Required Files

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

Team Repo

Warning

This will change for spring 2024

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.

Skip to main content

For the cookbook, use standard markdown.

to denote code inline `use single backticks`

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

to make a code block use 3 back ticks

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

To nest blocks use increasing numbers of back ticks.

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

Collaboration

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

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

Peer Review

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

In your summary comments answer the following:

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

Choose an action:

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

Review Badges

Review After Class

Skip to main content

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.

2024-01-23

related notes

Activities:

2024-01-25

related notes

Activities:

2024-01-30

related notes

Activities:

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 comment. Be sure it is not inside another repository.
3. Try using setting up git using your favorite IDE's git integration (not its terminal) or GitHub Desktop. Make a file gitgui.md and include some notes of how it went. Give the file a heading like `# Setting up <tool name>`, with the actual tool name you setup in the title (eg Github Desktop or VSCode source control panel or …) Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent to the terminal or does it use different terms?

2024-02-01

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.

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.

2024-01-25

this will actually be done in the first few minutes of 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?

2024-01-30

1. Find the glossary page for the course website, link it below. Review the terms for the next class: shell, terminal, bash, git, zsh, powershell, GitHub. Make a diagram using mermaid to highlight how these terms relate to one another
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.

Example "venn diagram " with mermaid subgraphs

flowchart subgraph Browsers subgraph Safari end subgraph Chromium based gg[Google Chrome] me[Microsoft Edge] end end

2024-02-01

1. Add file called terminal-vocab.md on a branch linked to this issue. Fill in the template below:

```
# Terminal vocab

## Before

<insert your diagram from 2024-01-30 prepare>

## Reflection

<!-- write a few bullets assessing your diagram above based on the notes from 2024-01-30. was anything in it


## Final thoughts

<!-- write a few bullets/sentences on how trying to make the diagram did/not help you think about the terms
```

2024-02-01

related notes

Activities:

```
# Terminal vocab

## Before

<insert your diagram from 2024-01-30 prepare>

## Reflection

<!-- write a few bullets assessing your diagram above based on the notes from 2024-01-30. was anything in it


## Final thoughts

<!-- write a few bullets/sentences on how trying to make the diagram did/not help you think about the terms
```

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.

2024-01-23

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 comment)

2024-01-25

related notes

Activities:

the text in `()` *below is why each step is assigned*

1. review today's notes after they are posted, both rendered and the raw markdown versions. Include links to both views in your badge PR comment. (to review + there are hints for some of the following there)
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. Do you think this information will help you approach learning more effectively? why or why not? How, if at all, does it changes how you think about debugging and learning to program? Give examples of

Skip to main content

strategies and prime for things we will see in the next few weeks)

3. Make a concept map of your current understanding of git and github git-basics-map.md. Use mermaid syntax, to draw your map. GitHub can render it for you including while you work using the preview button. (review what we have learned so far; think about connections + learn a new tool)
4. Read more about version control in general and add a “version control” row to your KWL chart with all 3 columns filled in. (git is version control, but not the only one)

2024-01-30

related notes

Activities:

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 comment. Be sure it is not inside another repository.
3. Try using setting up git using your favorite IDE’s git integration (not its terminal) or GitHub Desktop. Make a file gitgui.md and include some notes of how it went. Give the file a heading like `# Setting up <tool name>` with the actual tool name you setup in the title (eg Github Desktop or VSCode source control panel or …). Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent to the terminal or does it use different terms?
4. Design a small demo to illustrate the difference between git add and git commit and how they impact `git push`. Write your demo in stage_commit.md. Compare what happens based on what you can see on GitHub and what you can see with git status. Denote what you can tell about each case from the terminal and what you can tell from GitHub.com. For this demo, test things either using the files for this badge in your KWL repo or create a new sandbox repo where your account (not the course) is the owner.

2024-02-01

related notes

Activities:

1. Create a merge conflict in your KWL repo on the branch for this issue 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 merge_conflict_comparison.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) *You can put content in the file `merge_conflict_comparison.md` for the purpose of making the merge conflicts for this exercise.
2. Learn about GitHub forks and more about git branches(you can also use other resources)
3. In branches-forks.md in your KWL repo, compare and contrast branches and forks; be specific about their relationship. You may use mermaid diagrams if that helps you think through or communicate the ideas. If you use other resources, include them in your file as markdown links.

KWL File List

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:

- Extend a more practice:

Skip to main content

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
- Review a reference or resource for a topic
- write a code tour that orients a new contributor to a past project or an open source tool you like.

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

Build may be individual or in pairs.

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
```

The deliverables will depend on what your method is, which depend on your goals. It must be approved and the final submitted will have to meet what is approved. Some guidance:

Skip to main content

- 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.

Collaborative Build rules/procedures

- Each student must submit a proposal PR for tracking purposes. The proposal may be shared text for most sections but the deliverables should indicate what each student will do (or be unique in each proposal).
- the proposal must indicate that it is a pair project, if iteration is required, I will put those comments on both repos but the students should discuss and reply/edit in collaboration
- the project must include code reviews as a part of the workflow links to the PRs on the project repo where the code reviews were completed should be included in the reflection
- each student must complete their own reflection. The abstract can be written together and shared, but the reflection must be unique.

Build Ideas

General ideas to write a proposal for

- 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
- use version control, including releases on any open source side-project and add good contributor guidelines, README, etc

For these build options, you can copy-paste the template below to create your proposal issue and assign it to `@brownsarahm`.

For working alone there are two options, for working with a partner there is one.

212 Project Solo- Docs focus

Use this option if your team for your 212 project is not currently enrolled in this class or does not want to do a collaborative build. This version focuses on the user docs.

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

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

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi
1. ensure there is API level documentation in the code files
1. build a documentation website using [jupyterbook/ sphinx/doxygen/] that includes setup instructions and ex
1. configure the repo to automatically build the documentation website each time the main branch is updated


### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

212 Project Solo- Developer focus

Use this option if your team for your 212 project is not currently enrolled in this class or does not want to do a collaborative build. This version focuses on the contributor experience.

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

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

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

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

1. ensure there is API level documentation in the code files
1. add a license, readme, and contributor file
1. add [code tours](https://marketplace.visualstudio.com/items?itemName=vsls-contrib.codetour) that help some
```

Skip to main content

```
### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

212 Project Pair

Use this option if your teammate for your 212 project is in this class and wants to do a collaborative build.

```
## 212 Project Doc & Developer onboarding

Add documentation website and developer onboarding information to your CSC 212 project.

### Objectives

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

This project will provide information for a user to use the data structure implemented for a CSC 212 project

### Method

 <!-- describe what you will do , will it be research, write & present? will there be something you build? wi
1. ensure there is API level documentation in the code files
1. build a documentation website using [jupyterbook/ sphinx/doxygen/] that includes setup instructions and ex
1. configure the repo to automatically build the documentation website each time the main branch is updated
1. add [code tours](https://marketplace.visualstudio.com/items?itemName=vsls-contrib.codetour) that help some
1. set up a PR template
1. set up 2 issue templates: 1 for feature request and 1 for bug reporting



### Deliverables


- link to repo with the contents listed in method in the reflection file


### Milestones

<!-- give a target timeline -->
```

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.

Who should I request to review my work?

- Experience badge (inclass): TA in your group
- Experience report(makeup) `@brownsarahm``
- Review badge: `@instructors` team (it will convert to one of the three of us)
- Explore Proposal: `@brownsarahm`
- Explore Badge: `@brownsarahm`
- Build Proposal: `@brownsarahm`
- Build Badge: `@brownsarahm`

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.

Skip to main content

It should be a tutorial or blog style piece of writing, likely with code excerpts or screenshots embedded in it.

an example that uses mostly screenshots

an example of heavily annotated code

They should be markdown files in your KWL repo. I recommend myst markdown.

Git and GitHub

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

If your error looks like this…

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

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

After you run

```
git pull
```

You’ll probably have to resolve a merge conflict

My command line says I cannot use a password

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

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

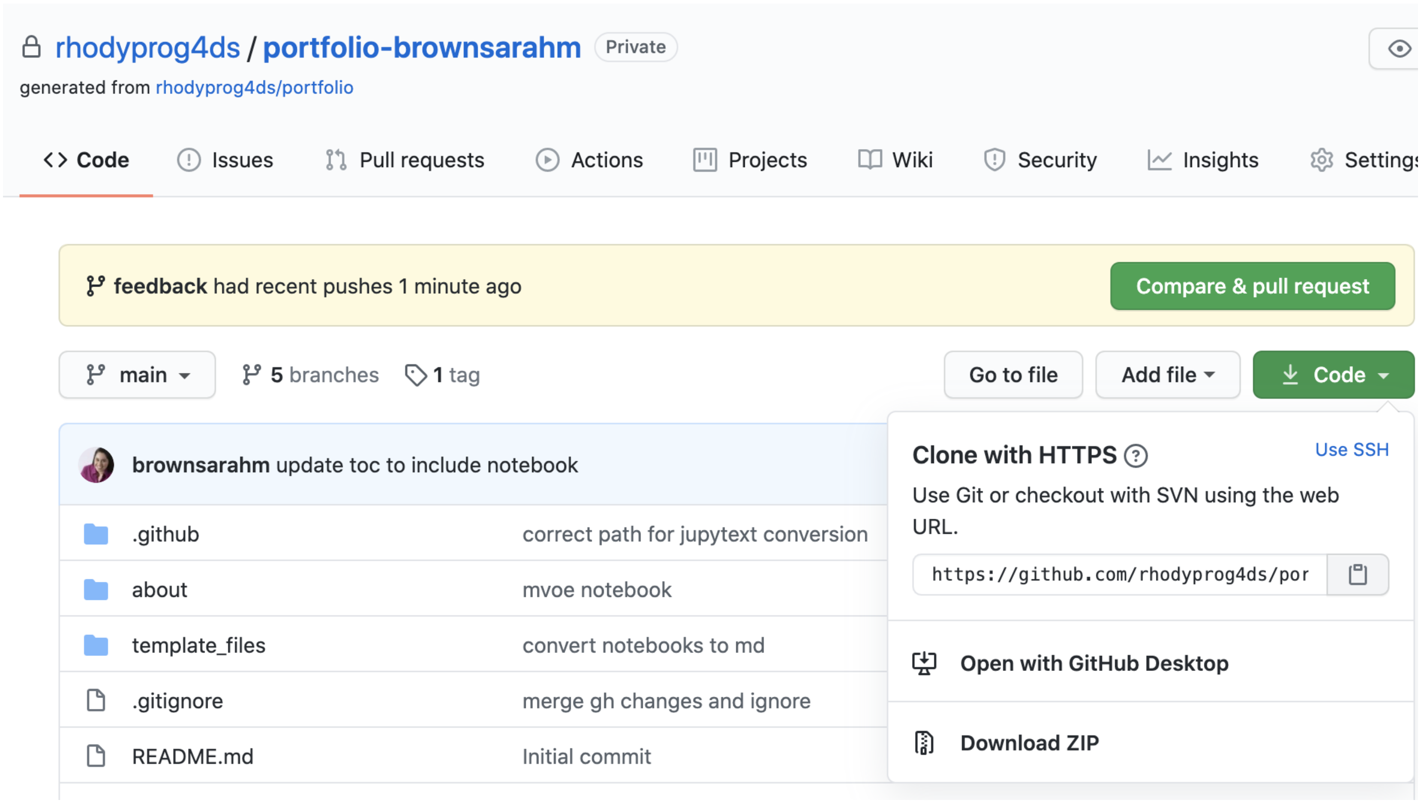
That’s ok. You can fix it.

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)

Skip to main content

click the green “Code” button, then copy the url that’s show


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
Clone with HTTPS
Use SSH
Use Git or checkout with SVN using the web URL.
https://github.com/rhodyprog4ds/por
Open with GitHub Desktop
Download ZIP

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
```

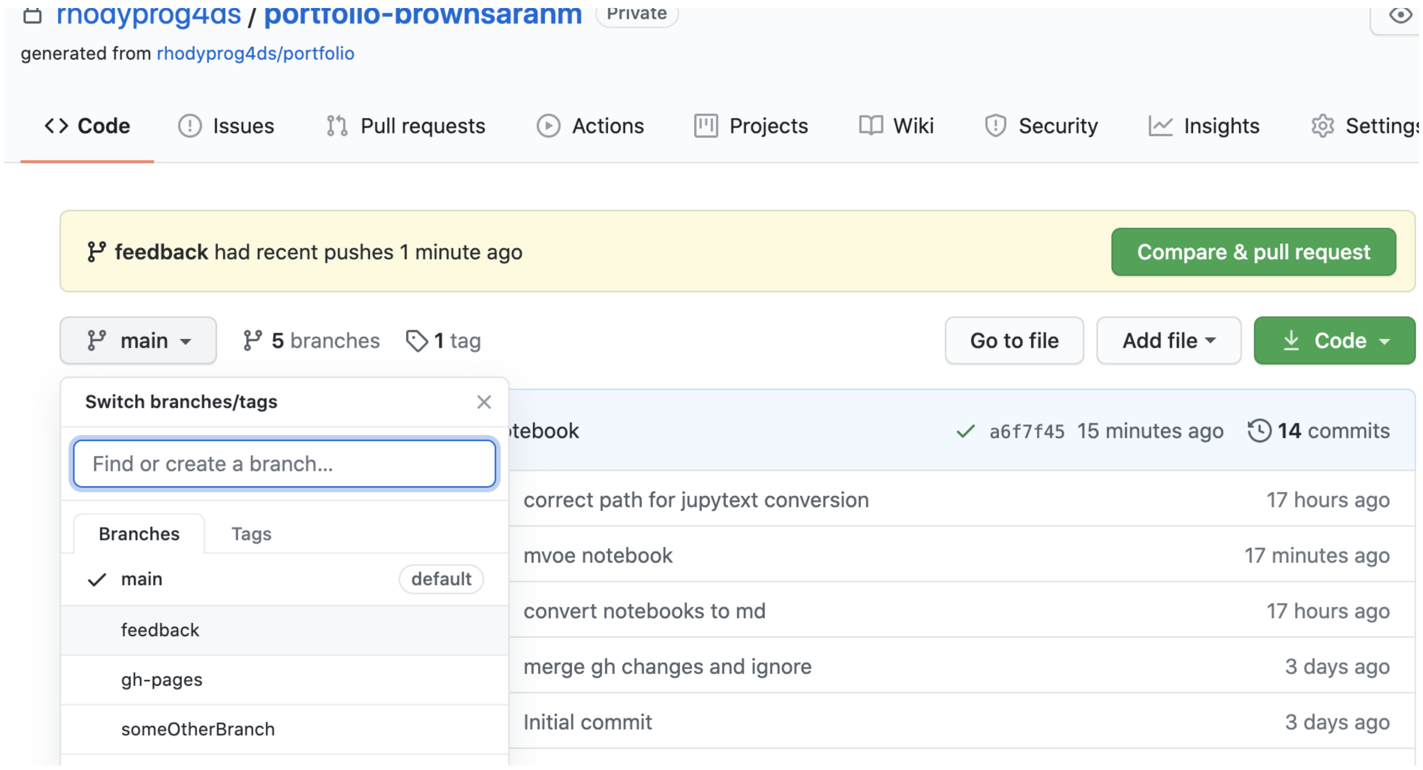
then past your url that you copied. It will look something like this, but the last part will be the current assignment repo and your username.

```
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.


rhodyprog4ds / portfolio-brownsarahm Private
generated from rhodyprog4ds/portfolio

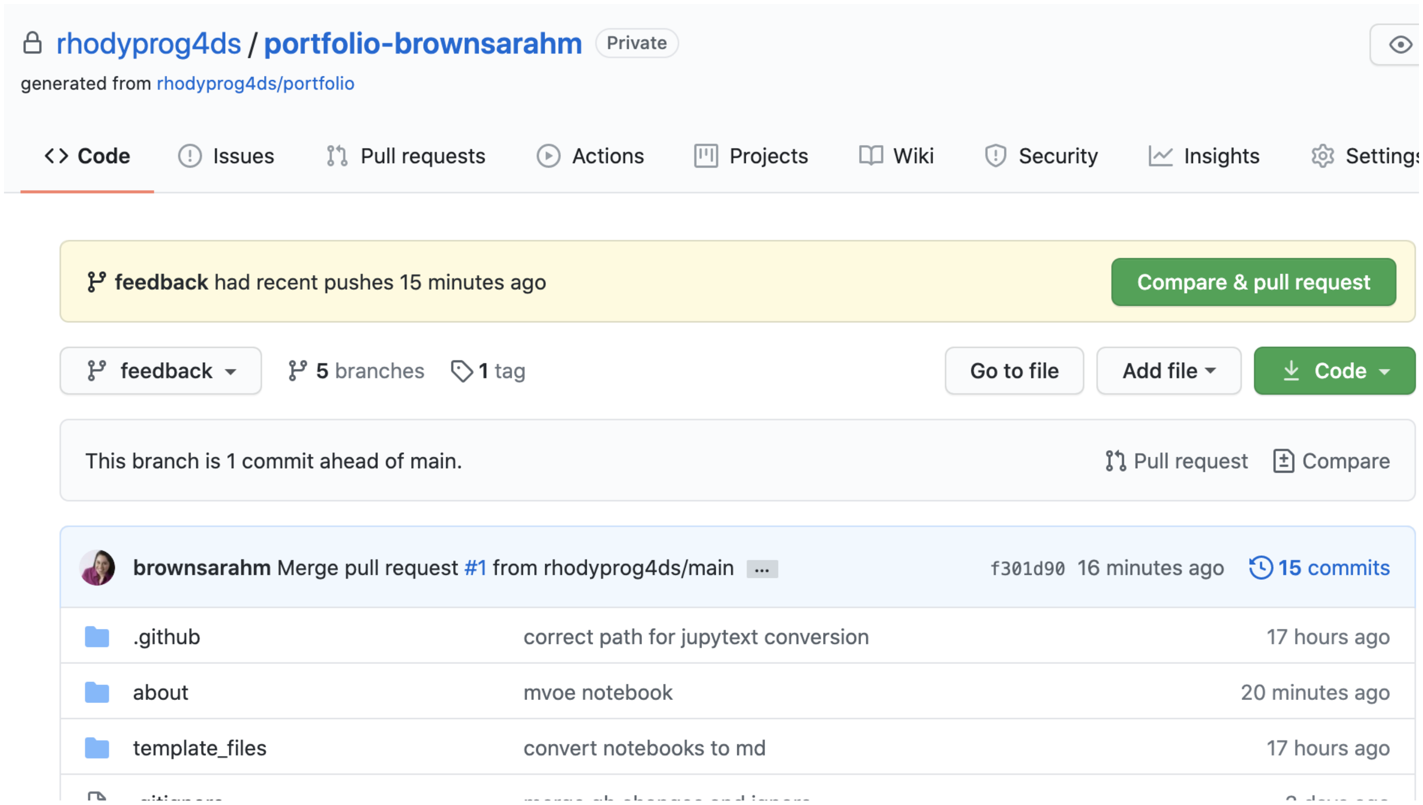
feedback had recent pushes 1 minute ago

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

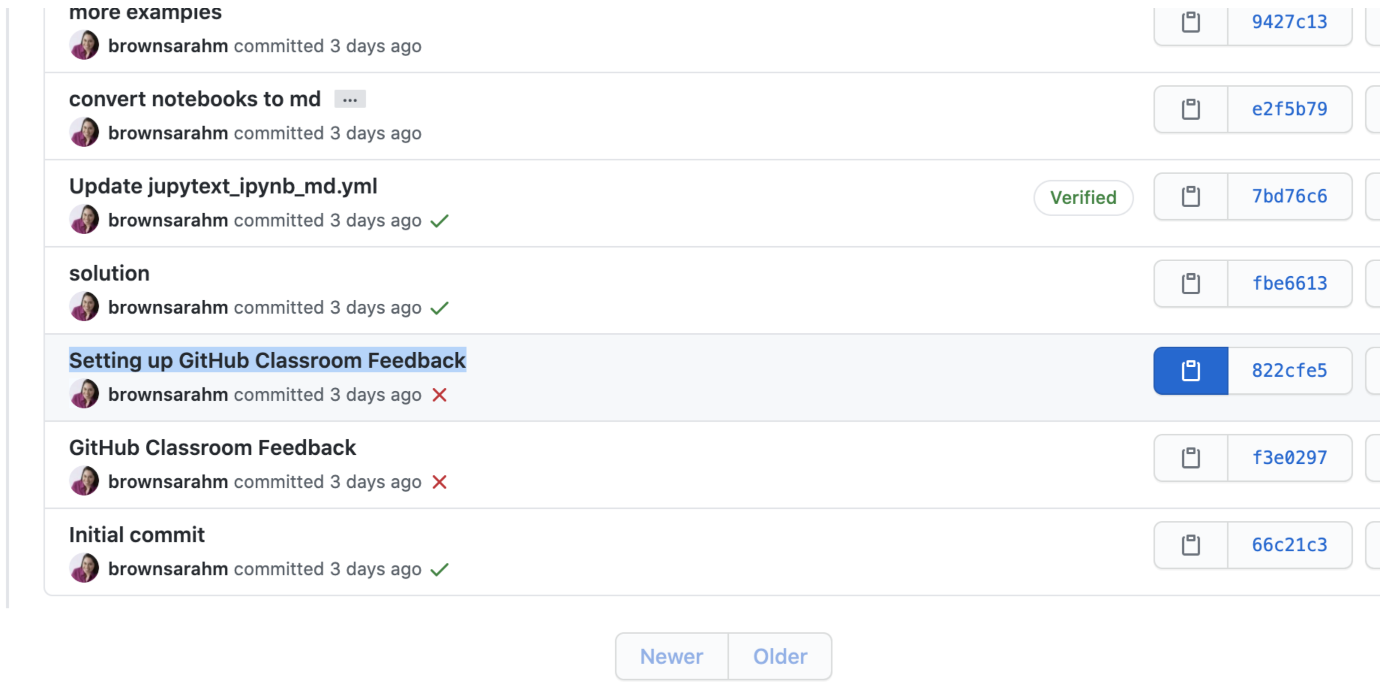
feedback had recent pushes 15 minutes ago

This branch is 1 commit ahead of main.

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.


more examples
brownsarahm committed 3 days ago
9427c13

convert notebooks to md
brownsarahm committed 3 days ago
e2f5b79

Update jupytext_ipynb_md.yml
brownsarahm committed 3 days ago ✓
Verified
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.

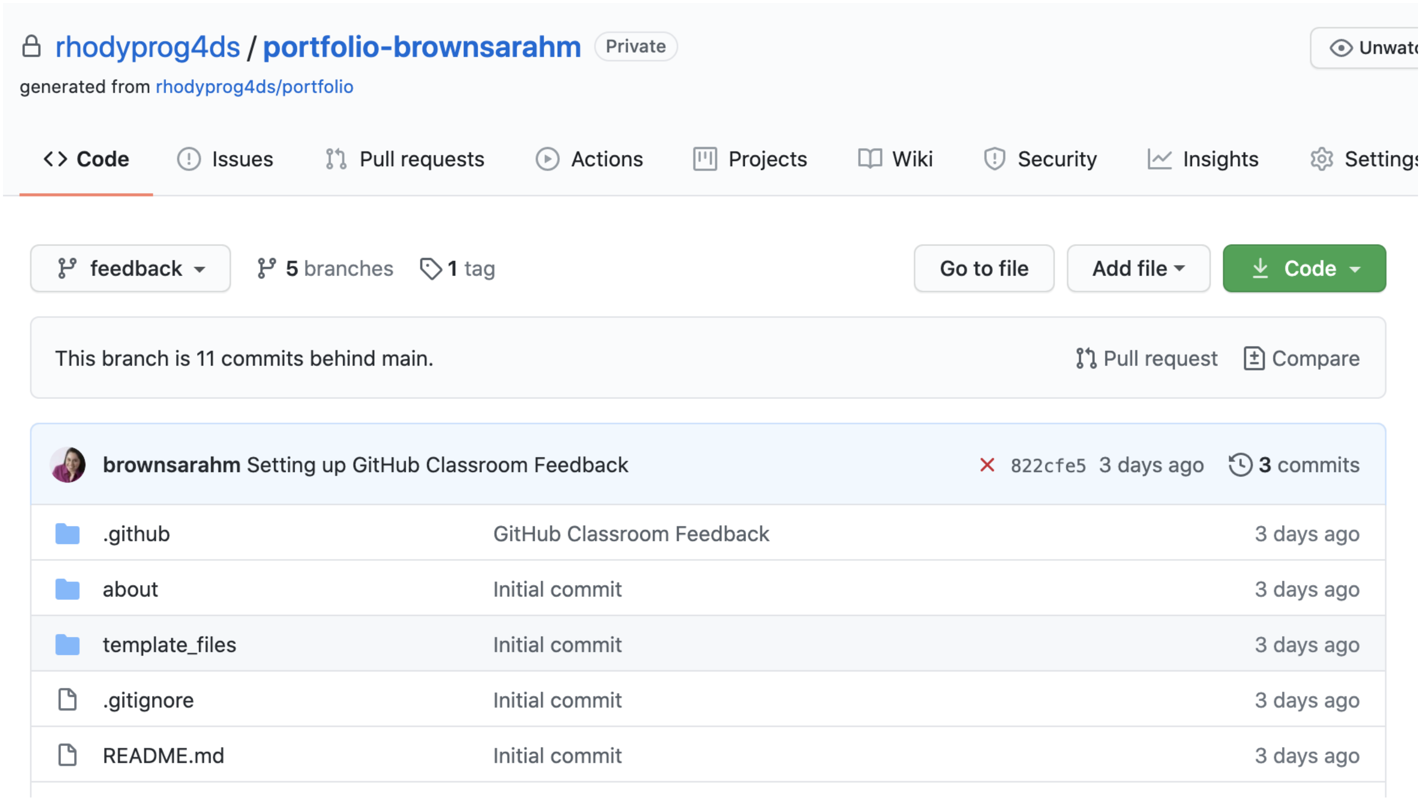
```
git push origin -f
```

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").



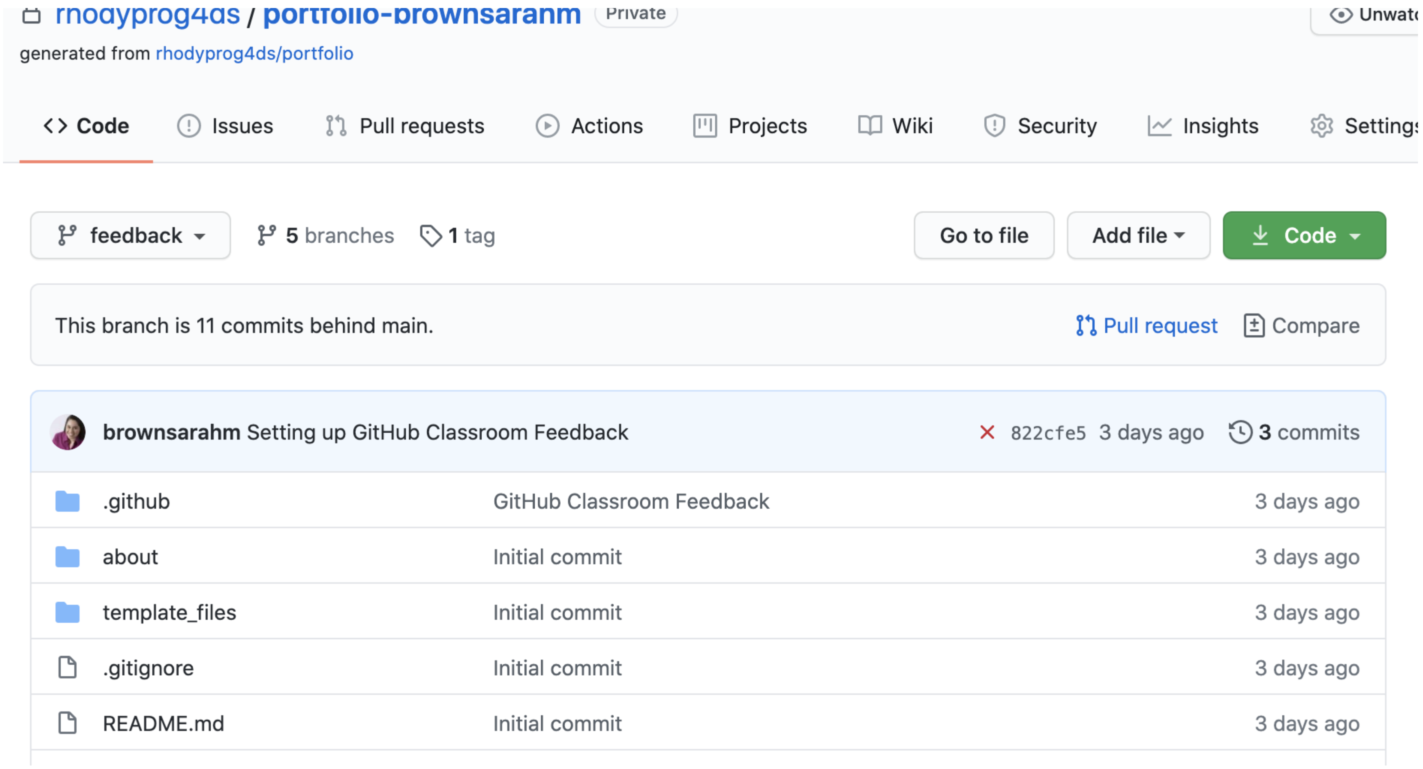
This branch is 11 commits behind main.

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

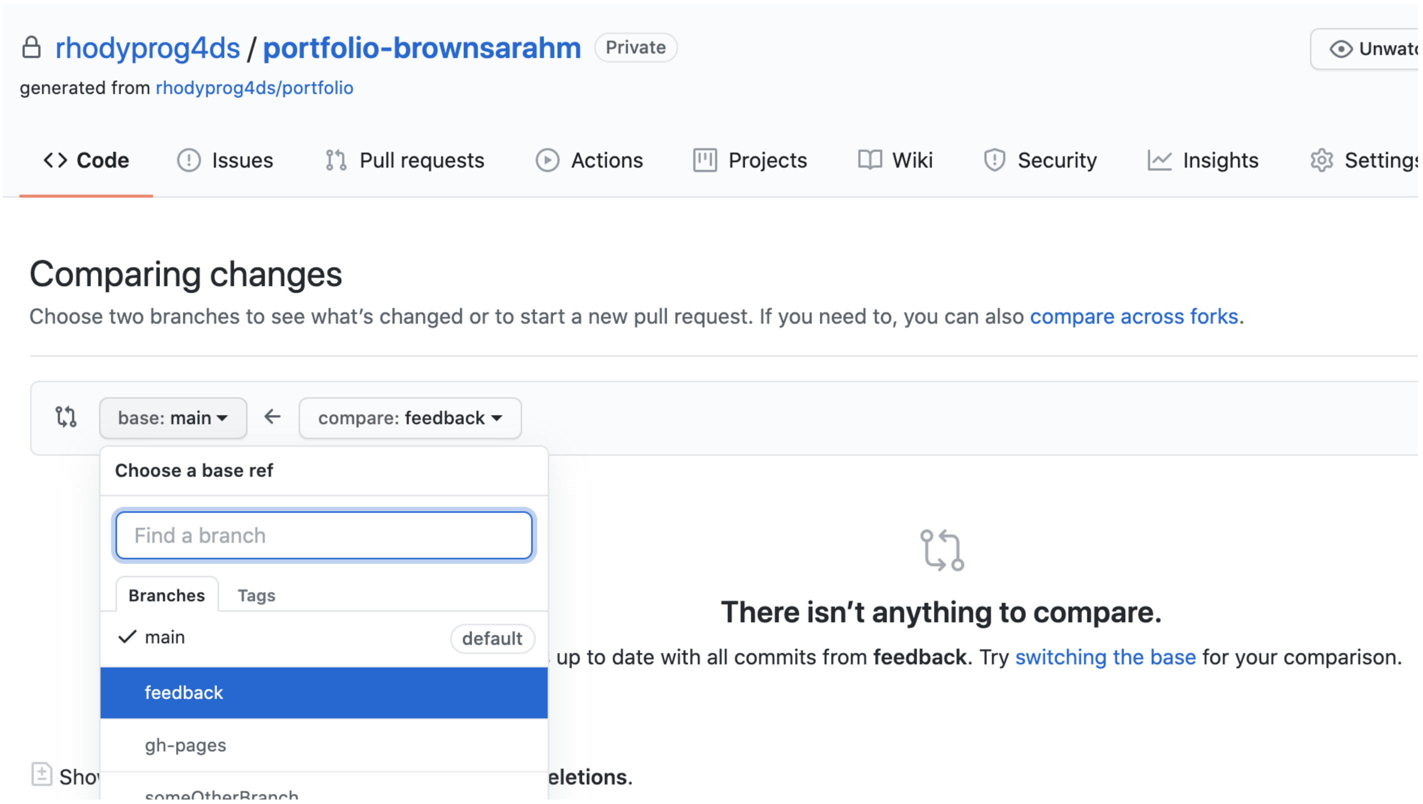
generated from rhodyprog4ds/portfolio

This branch is 11 commits behind main.

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

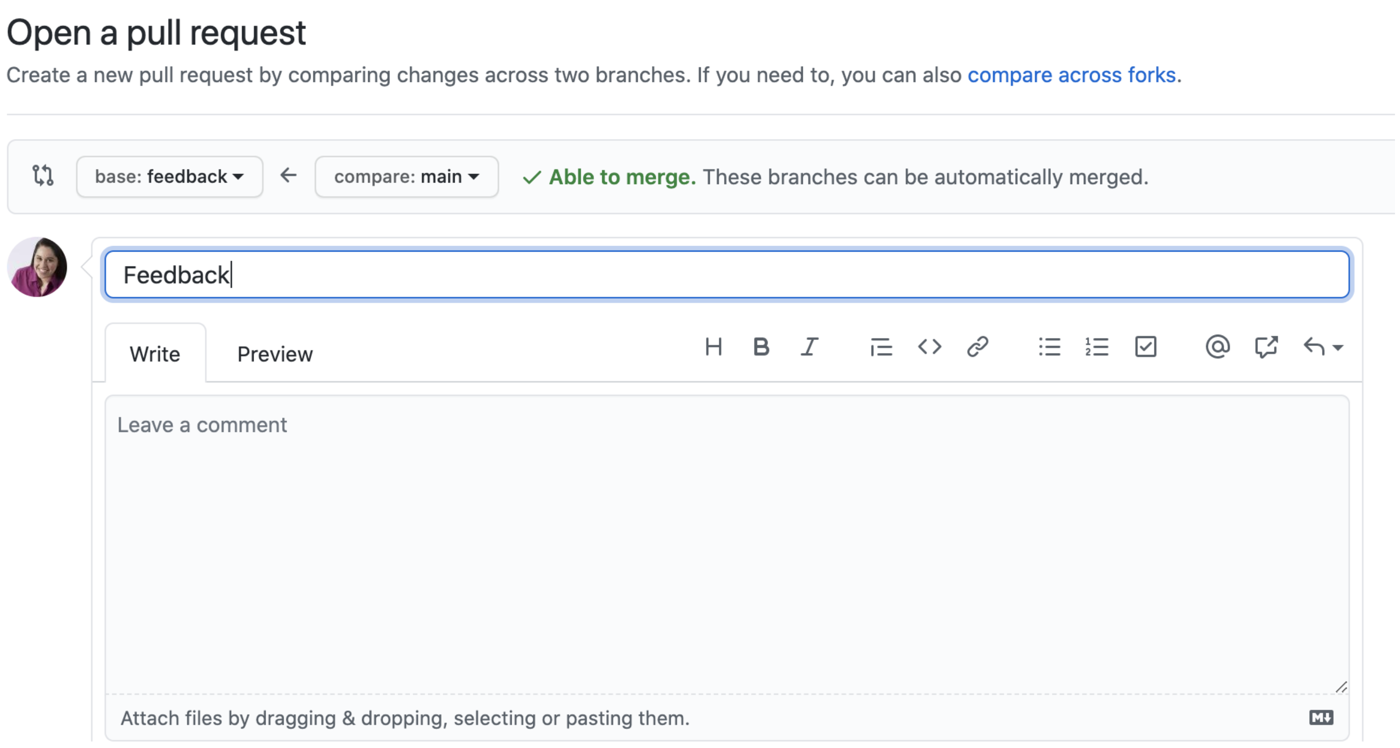
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

There isn't anything to compare.

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

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

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
```

Other Course Software/tools

Courseutils

This is how your badge issues are created. It also has some other utilities for the course. It is open source and questions/issues should be posted to its issue tracker

Jupyterbook

Glossary

Skip to main content

Contributing glossary terms or linking to uses of glossary terms to this page is eligible for community badges

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

argument

input to a command line program

bash

bash or the bourne-again shell is the primary interface in UNIX based systems

bitwise operator

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

branch

a copy of the main branch (typically) where developmental changes occur. The changes do not affect other branches because it is isolated from other branches.

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

divergent

git branches that have diverged means that there are different commits that have same parent; there are multipe ways that git could fix this, so you have to tell it what strategy to use

fixed point number

the concept that the decimal point does not move in the number. Cannot represent as wide of a range of values as a floating point number.

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.

git

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

GitHub

a hosting service for git repositories

Skip to main content

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 objects

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

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

a file in the .git directory that indicates what is currently 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

merge conflict

mermaid

mermaid syntax allows user to create precise, detailed diagrams in markdown files.

hash function

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

hashing

transforming an input of arbitrary length to a unique fixed length output (the output is called a hash; used in hash tables and when git hashes commits).

integrated 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.

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.

issue

provides the ability to easily track ideas, feedback, tasks, or bugs. branches can be created for specific issues. an issue is open when it is created. pull requests have the ability to close issues. see more in the docs

Linker

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

option

also known as a flag, a parameter to a command line program that change its behavior, different from an argument the “location” of a file or folder(directory) in a computer

pointer

a variable that stores the address of another variable

pull (changes from a repository)

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

pull request

allow other users to review and request changes on branches. after a pull request recieves approval you can merge the changed content to the main branch.

PR

short for pull request

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; may start with a name, includes a single file name or may start with `./`

repository

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

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)

sh

abbr. see shell

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

a program that makes shell visible for us and allows for interactions with it

Skip to main content

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.

zsh

zsh or z shell is built on top of the bash shell and contains new features

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

Skip to main content

You will be making a lot of documentation of bits, in your own words. You will be directed to try things and make notes. This based 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

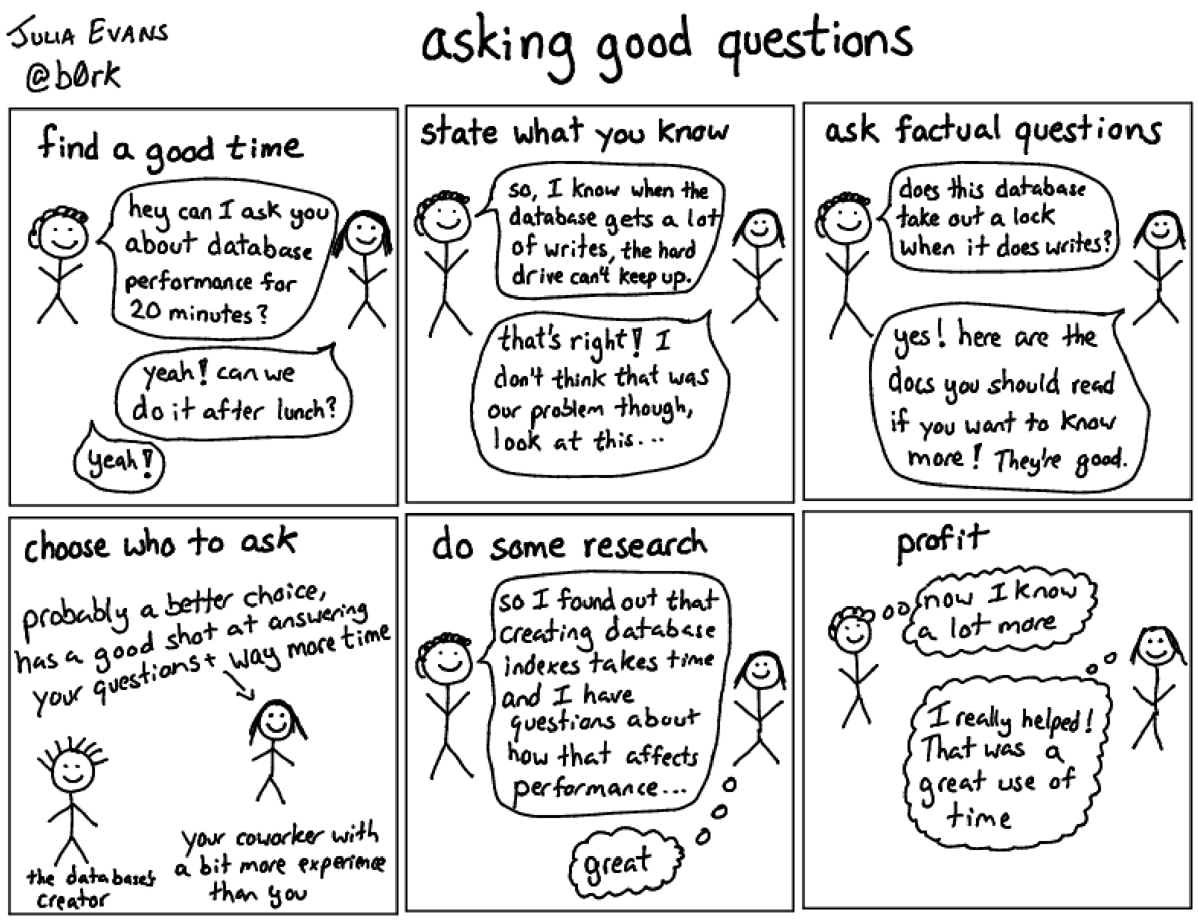
- bash
- C
- Python

Skip to main content

Getting Help with Programming

This class will help you get better at reading errors and understanding what they might be trying to tell you. In addition here are some more general resources.

Asking Questions


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

Skip to main content

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