About this Site

Contents

Resources

- Glossary
- General Tips and Resources
- How to Study in this class
- GitHub Interface reference
- Language/Shell Specific References
- Getting Help with Programming
- Getting Organized for class
- More info on cpus
- Windows Help & Notes
- Advice from Spring 2022 Students
- Advice from Fall 2022 Students
- Advice from Spring 2023 Students
- Advice from Fall 2023 Students

Welcome to the course website for Computer Systems and Programming Tools in Spring 2025.

This class meets TuTh 12:30PM - 1:45PM in Ranger 302 and lab on Monday 3:00PM - 4:45PM 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

Question from Class

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

Further reading

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

Hint

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

Click here!

Special tips will be formatted like this

Check your Comprehension

Questions to use to check your comprehension will looklike this

Contribute

Chances to earn community badges will sometimes be marked like this

Computer Systems and Programming Tools

About this course

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

About this syllabus

This syllabus is a *living* document. You can get notification of changes from GitHub by “watching” the repository You can view the date of changes and exactly what changes were made on the Github repository page.

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

> **Should you download the syllabus and rely on your offline copy?**
>
> No, because the syllabus changes

About your instructor

Name: Dr. Sarah M Brown

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.

Name: Ayman Sandouk Office hours: listed on communication page

Ayman is a Masters studenent at the Universitry of Rhode Island with Bachelors in CS from URI. Ayman’s research is currently focusing on benchmarking LLMs for fairness

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 Ayman 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.

Prismia chat

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

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

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

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

Course Website

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

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

GitHub

You will need a GitHub Account. If you do not already have one, please create one by the first day of class. If you have one, but have not used it recently, you may need to update your password and login credentials as the Authentication rules changed in Summer 2021.

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

> **Important**
>
> You need to install this on Mac

Programming Environment

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

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

Requirements:

- Python with scientific computing packages (numpy, scipy, jupyter, pandas, seaborn, sklearn)
- a C compiler
- Git
- access to a bash shell
- A high compatibility web browser (Safari will sometimes fail; Google Chrome and Microsoft Edge will; Firefox probably will)
- 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**

- If you will not do any side projects, install python via Anaconda video install
- Otherwise, use the base python installer and then install libraries with pip
- 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.

Each section be viewed at two levels of detail. You can toggle the tabs and then the whole page will be at the level of your choice as you scroll.

TL;DR **Full Detail**

this will be short explanations; key points you should **remember**

Learning Outcomes

TL;DR **Full Detail**

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.

You should be a more independent and efficient developer and better collaborator on code projects by the end of the semester.

Principles of Grading

TL;DR **Full Detail**

- Learning happens with practice and feedback
- I value **learning** not perfect performance or productivity
- a C means you can follow a conversation about the material, but might need help to apply it
- a B means you can *also* apply it in basic scenarios or if the problem is broken down
- an A means you can *also* apply it in complex scenarios independently

please do not make me give you less than a C, but a D means you showed up basically, but you may or may not have actually retained much

The course is designed to focus on **success** and accumulating knowledge, not taking away points.

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

TL;DR | Full Detail

We will use developer tools to do everything in this class; in the long term this will benefit you, but it makes the first few weeks hard, so **mistakes in the first few weeks cannot hurt your grade** as long as you learn eventually.

Deadlines are *extra flexible* for 3 weeks while you figure things out.

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.

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

TL;DR | Full Detail

Different badges are different levels of complexity and map into different grades.

- experience: like attendance
- lab: show up & try
- review: understand what was covered in class
- practice: apply what was covered in class
- explore: get a mid-level understanding of a topic of your choice

- build: get a deep understanding of a topic of your choice

To pass:

- 22 experience badges
- 12 lab checkouts

Add 18 review for a C or 18 practice for a B.

For an A you can choose:

- 18 review + 3 build
- 18 pracitce + 6 explore

you can mix & match, but the above plans are the simplest way there

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.

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

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

Can you do any combination of badges?

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

Experience Badges

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`

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

You earn credit for lab by attending and completing core tasks as defined in a lab issue posted to your repo each week. Work that needs to be correct through revisions will be left to a review or practice badge.

You will have to have a short meeting with a TA or intructor to get credit for each lab. In the lab instructions there will be a checklist that the TA or instructor will use to confirm you are on track. In these conversations, we will make sure that you know how to do key procedural tasks so that you are set up to continue working independently.

To make up a lab, complete the tasks from the lab issue on your own and attend office hours to complete the checkout.

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.

> **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
- merging the PR after approval

You should create one PR per badge

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

Community Badges

TL;DR **Full Detail**

These are like extra credit, they have very limited ability to make up for missed work, but can boost your grade if you are on track for a C or B.

🎁 Free corrections

TL;DR **Full Detail**

If you get a 🎁 apply the changes to get credit.

Important

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

This means that, for example, the first time you make a particular mistake, might get a 🎁, but the second time you will probably get a hint, and a third or fourth time might be a regular revision with a comment like `see #XX and fix accordingly` where XX is a link to a previous badge.

IDEA

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

Ungrading Option

TL;DR **Full Detail**

You should try to follow the grading above; but sometimes weird things happen. I care that you learn.

If you can show you learned in some other way besides earning the badges above you may be able to get a higher grade than your badges otherwise indicate.

What do you think?

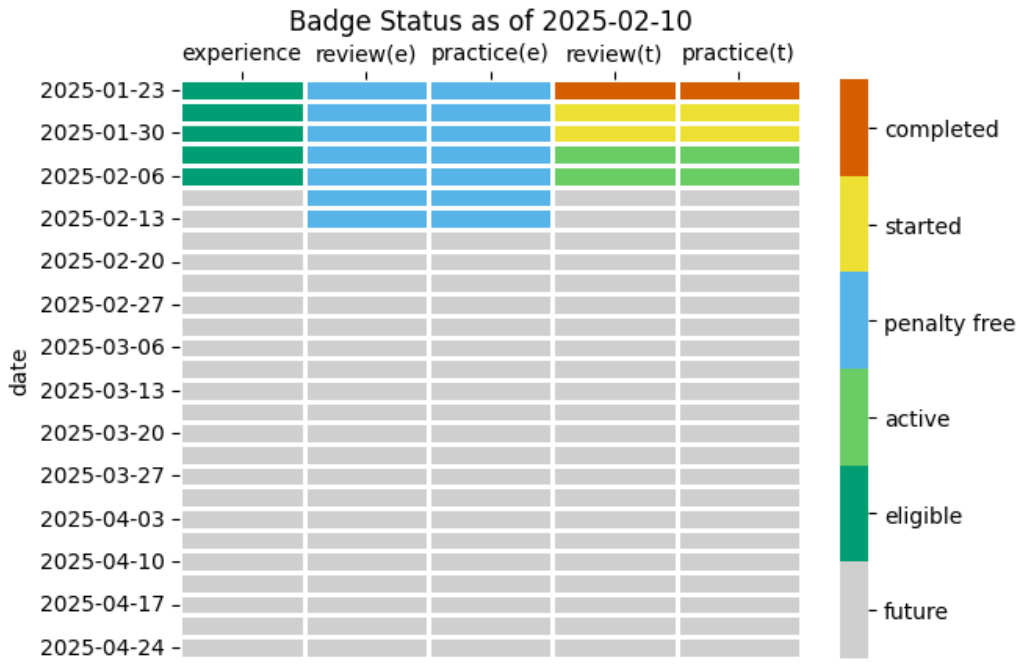
share your thoughts on this option in the discussions for the class and then log it for a community badge!

Badge Deadlines and Procedures

This page includes more visual versions of the information on the grading 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.

▶ Show code cell source

```
Text(0.5, 1.0, 'Badge Status as of 2025-02-10')
```



Getting Feedback

Who should you request/assign?

course item type	issue asignee	PR reviewer
prepare work	not required; can be student	none requierd; merge to experience branch
experience badge	N/A	TA asigned to your group (will be automatic after a few classes)
practice badge	not required; can be student	@instuctors (will then convert to 1/3 people)
review badge	not required; can be student	@instuctors (will then convert to 1/3 people)
explore badge	proposal, assigned to @AymanBx (also student optionally)	@AymanBx
build badge	proposal, assigned to @AymanBx (also student optionally)	@AymanBx
anything merged pre-emptively in penalty free	@AymanBx	clear others

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 instructor.

Review and Practice Badges

These badges have 5 stages:

- posted: tasks are on the course website and 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 only be granted in exceptional circumstances.

Expired badges will receive a comment and be closed

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

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

Explore Badges

Explore badges have 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

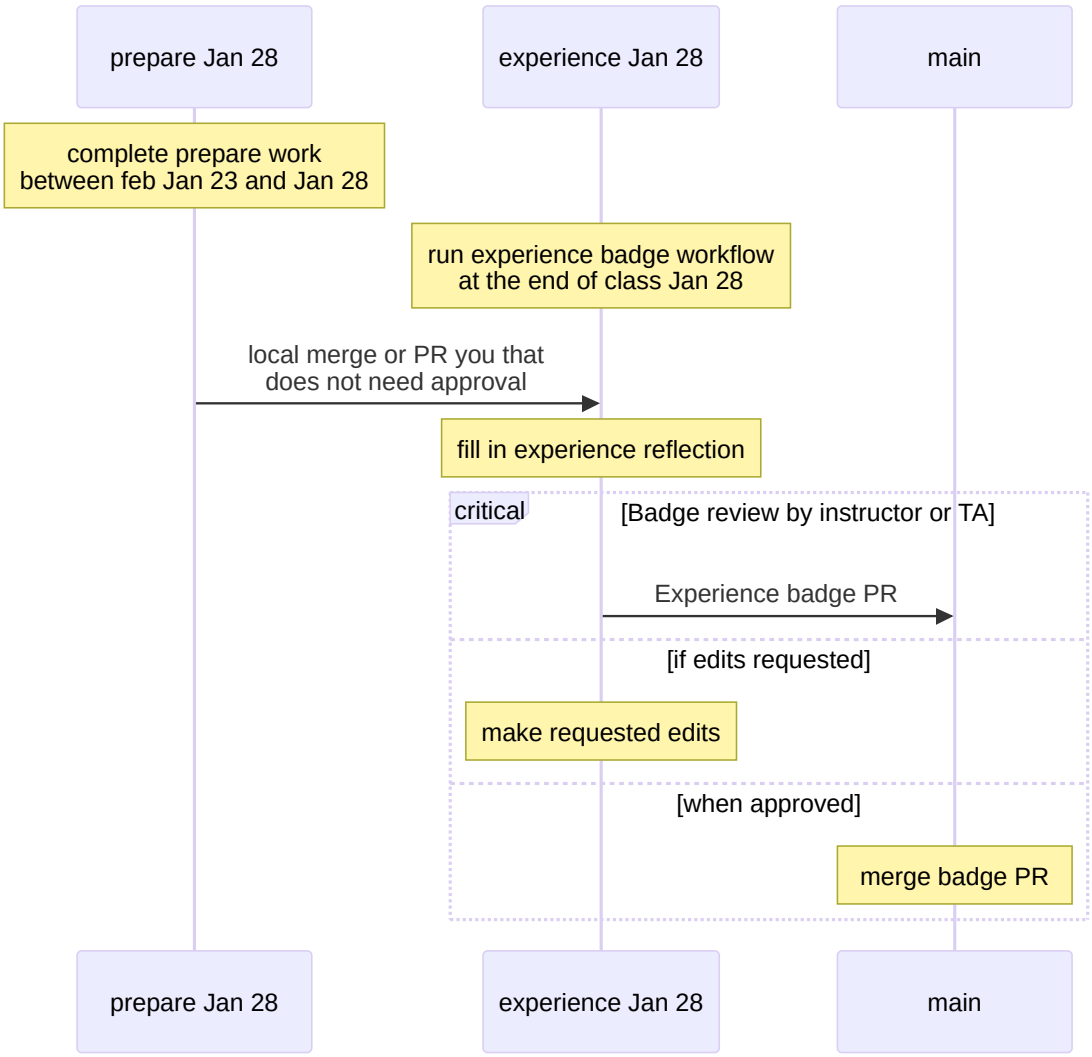
At most one build badge will be evaluated every 4 weeks. This means that if you want to earn 3 build badges, the first one must be in 8 weeks before the end of the semester, March 4. The second would be due April 1st, and the third submitted by the end of classes, April 29th.

Procedures

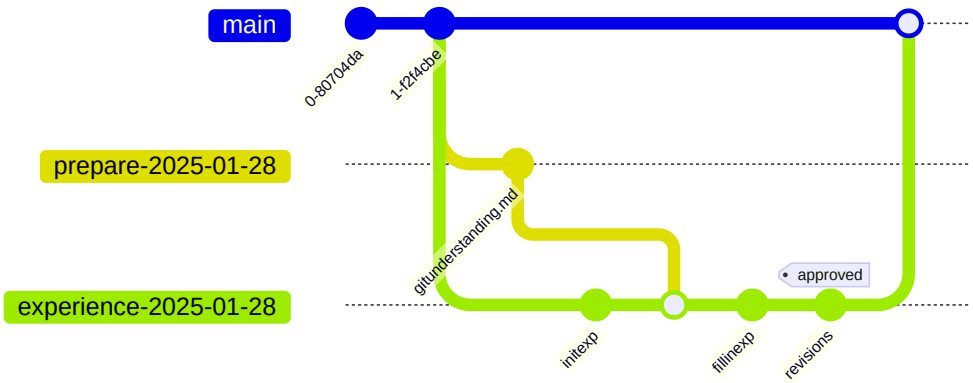
Prepare work and Experience Badges Process

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)



In the end the commit sequence for this will look like the following:



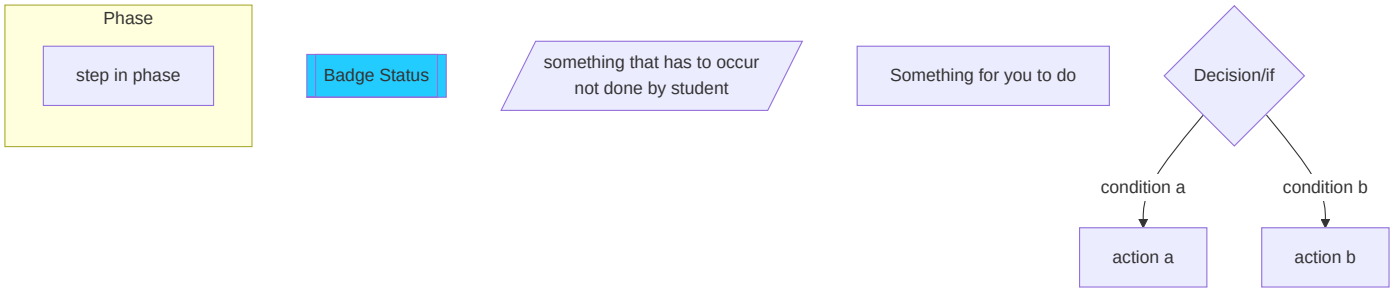
You can merge the prepare into the experience with a PR or on the command line, your choice.

Where the “approved” tag represents and approving reivew on the PR.

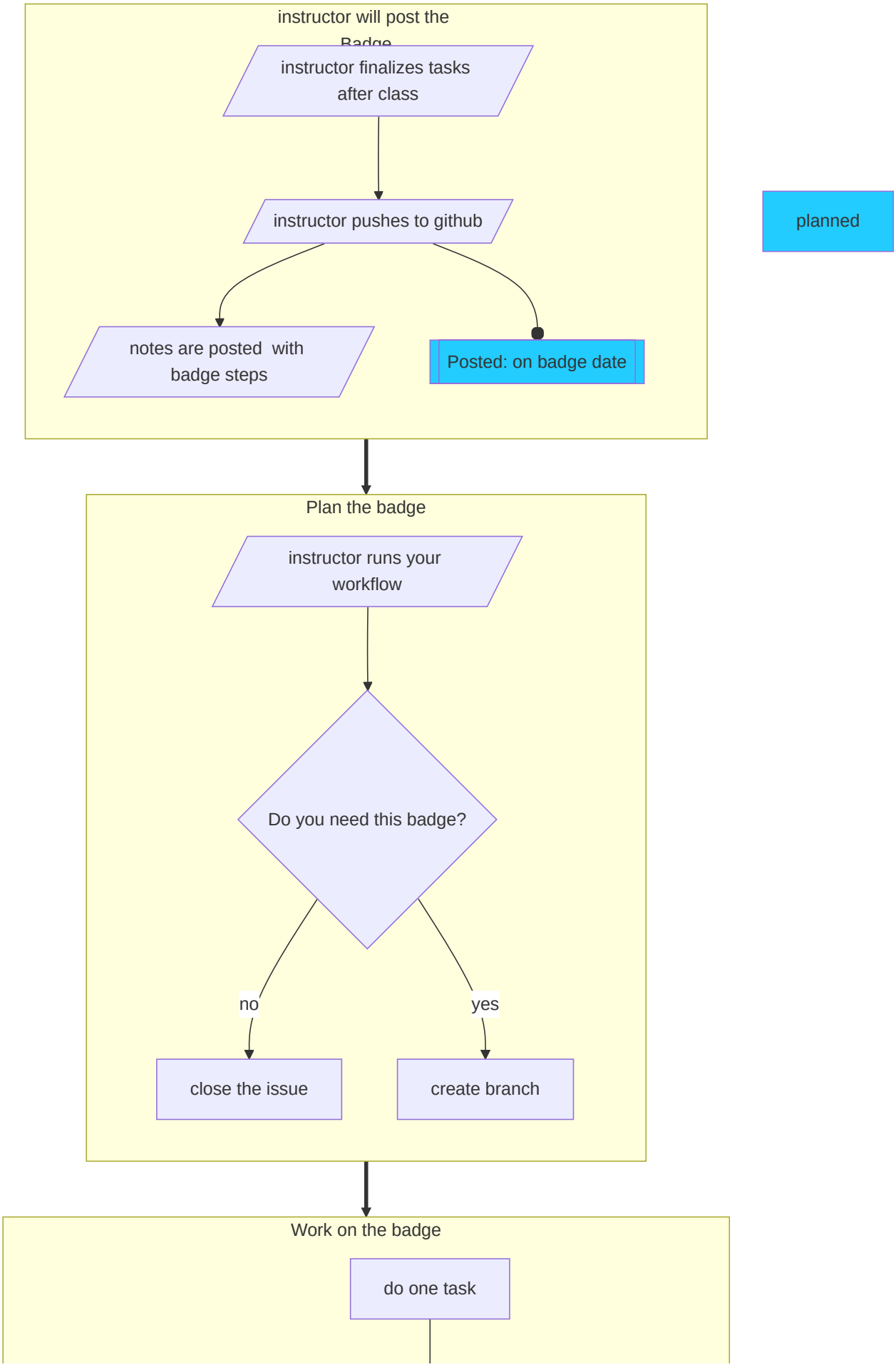
You can, once you know how, do this offline and do the merge with in the CLI instead of with a PR.

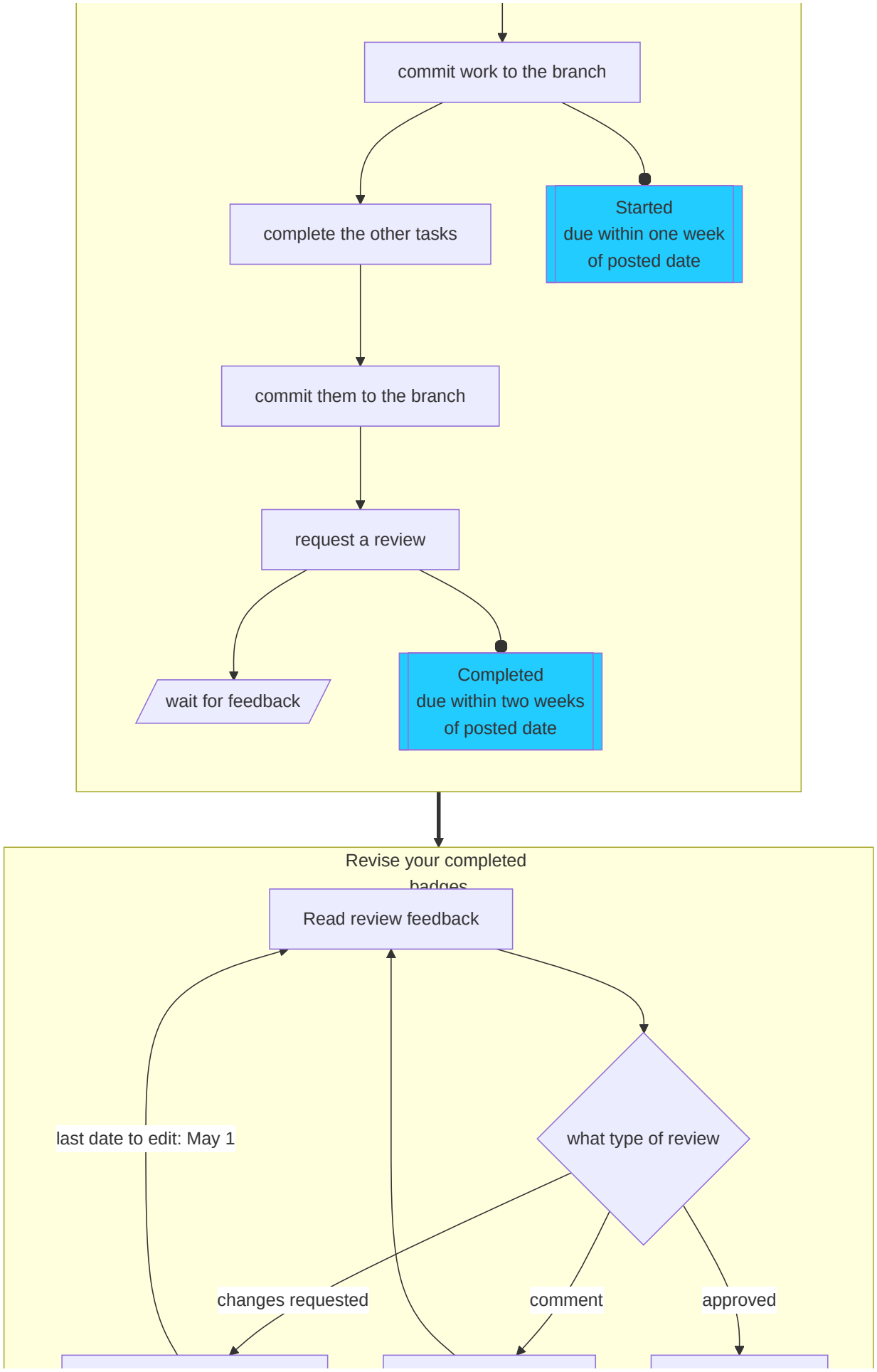
Review and Practice Badge

Legend:



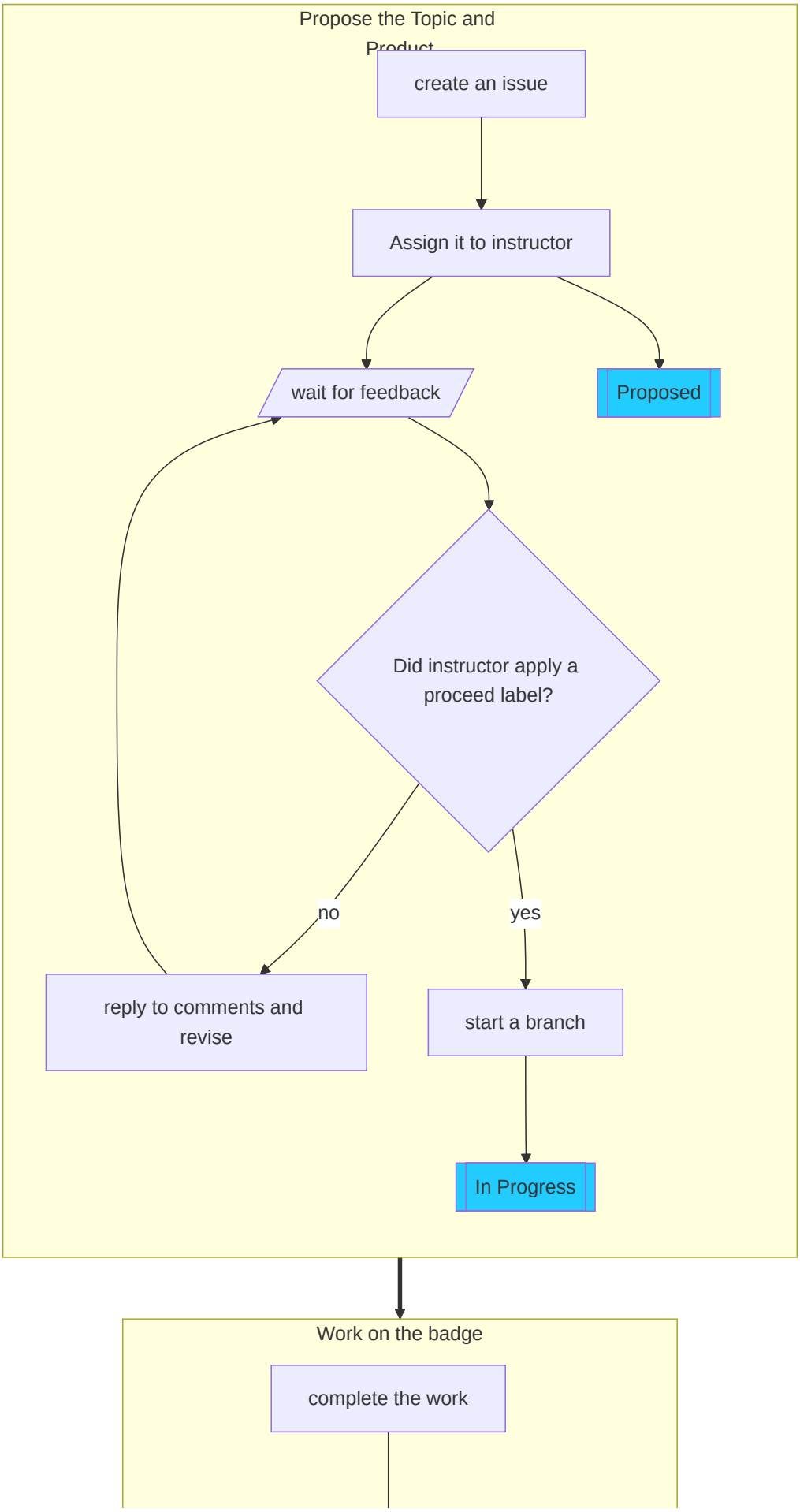
This is the general process for review and practice badges

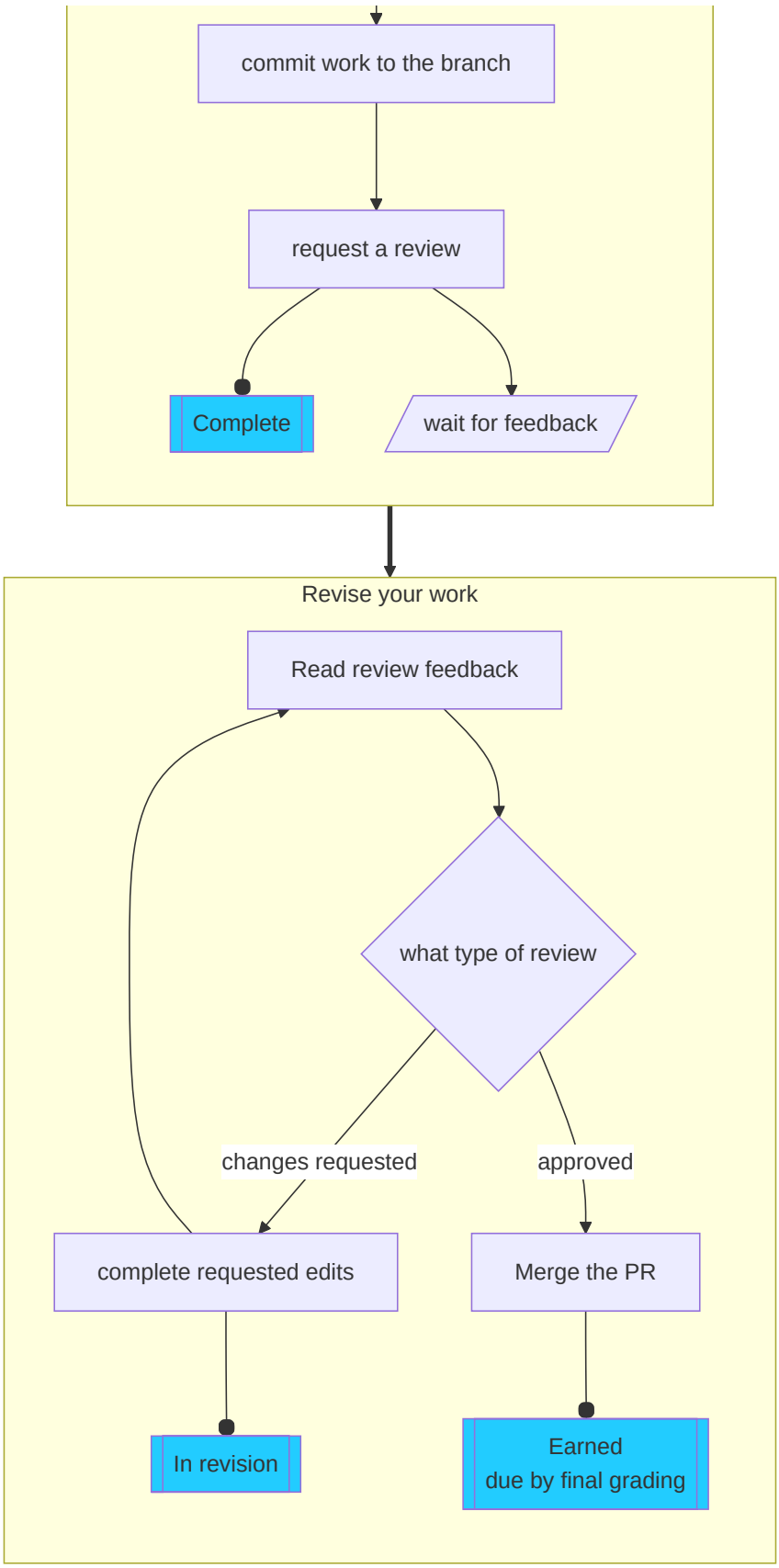

instructor will post the
Badge
instructor finalizes tasks after class
instructor pushes to github
notes are posted with badge steps
Posted: on badge date
planned
Plan the badge
instructor runs your workflow
Do you need this badge?
no
yes
close the issue
create branch
Work on the badge
do one task


commit work to the branch
complete the other tasks
Started
due within one week
of posted date
commit them to the branch
request a review
wait for feedback
Completed
due within two weeks
of posted date
Revise your completed
badges
Read review feedback
last date to edit: May 1
what type of review
changes requested
comment
approved

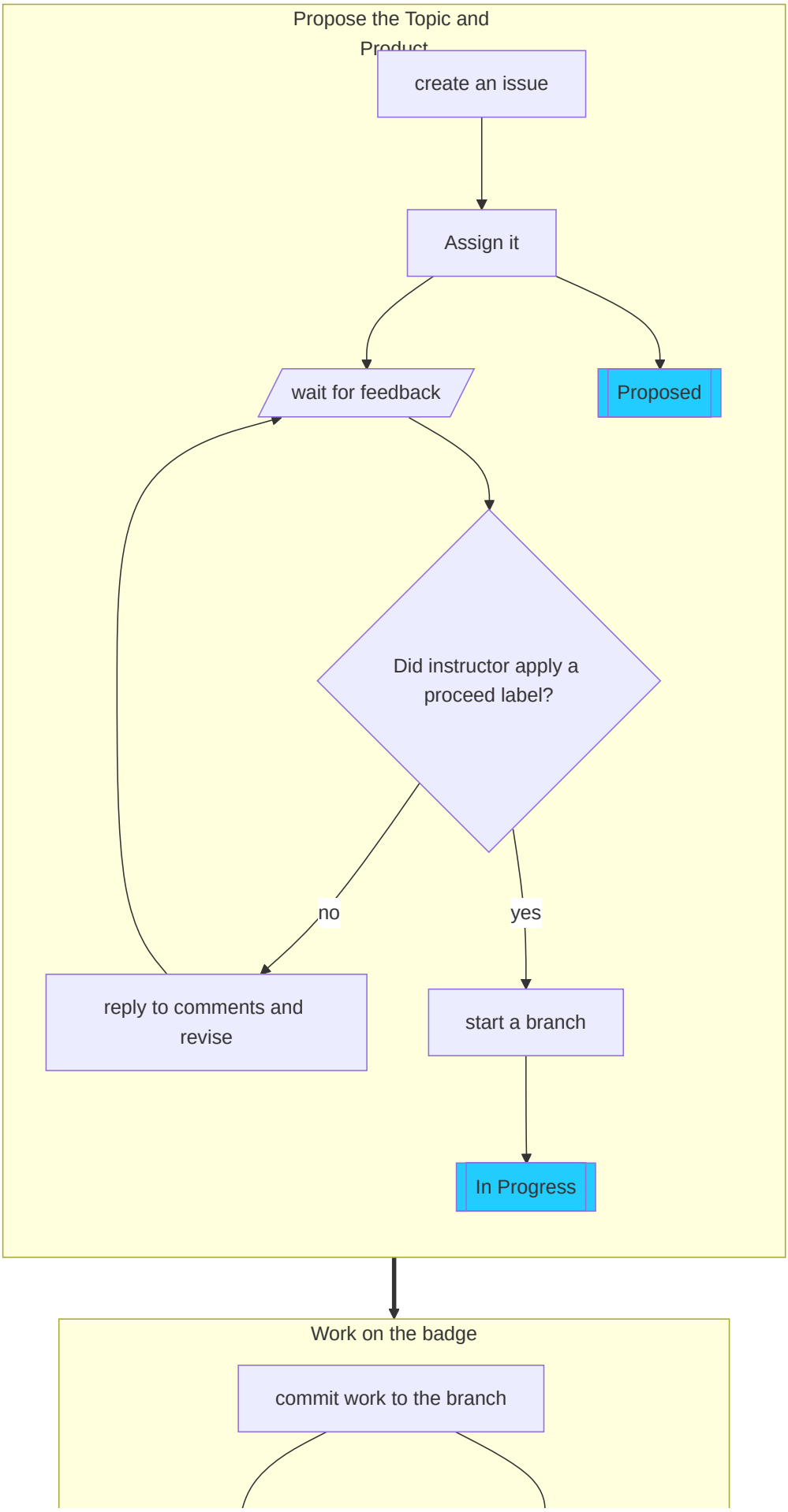

complete requested edits
reply to comments
Merge the PR
Earned
due by final grading

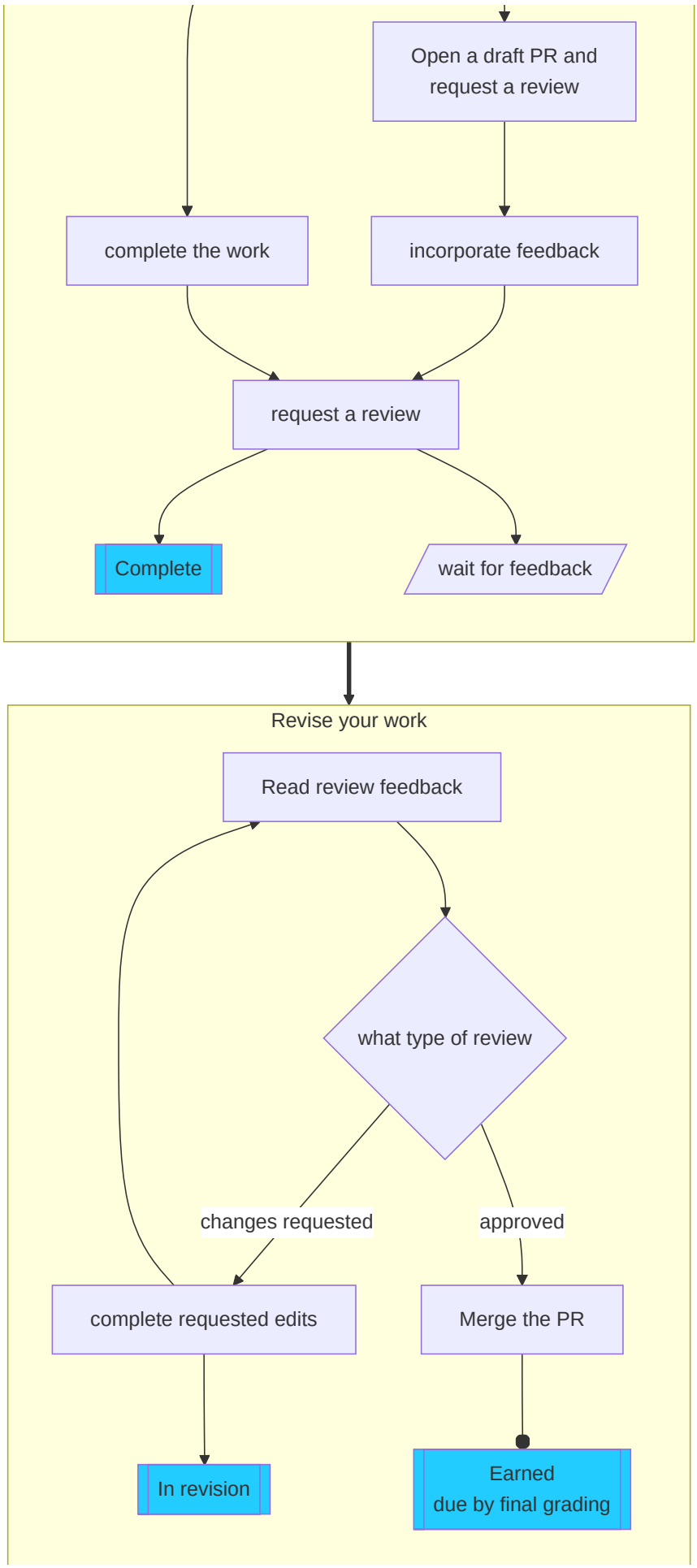
Explore Badges


Propose the Topic and Product
create an issue
Assign it to instructor
wait for feedback
Proposed
Did instructor apply a proceed label?
no
yes
reply to comments and revise
start a branch
In Progress
Work on the badge
complete the work


commit work to the branch
request a review
Complete
wait for feedback
Revise your work
Read review feedback
what type of review
changes requested
approved
complete requested edits
Merge the PR
In revision
Earned
due by final grading

Build Badges


Propose the Topic and Product
create an issue
Assign it
wait for feedback
Proposed
Did instructor apply a proceed label?
no
yes
reply to comments and revise
start a branch
In Progress
Work on the badge
commit work to the branch


Open a draft PR and request a review
complete the work
incorporate feedback
request a review
Complete
wait for feedback
Revise your work
Read review feedback
what type of review
changes requested
approved
complete requested edits
Merge the PR
In revision
Earned due by final grading

Community Badges

You can log them either manualy via files or with help of an action that a past student contributed!

Logger Action

Your KWL repo has an action called “Community & Explore Badge Logger” that will help you

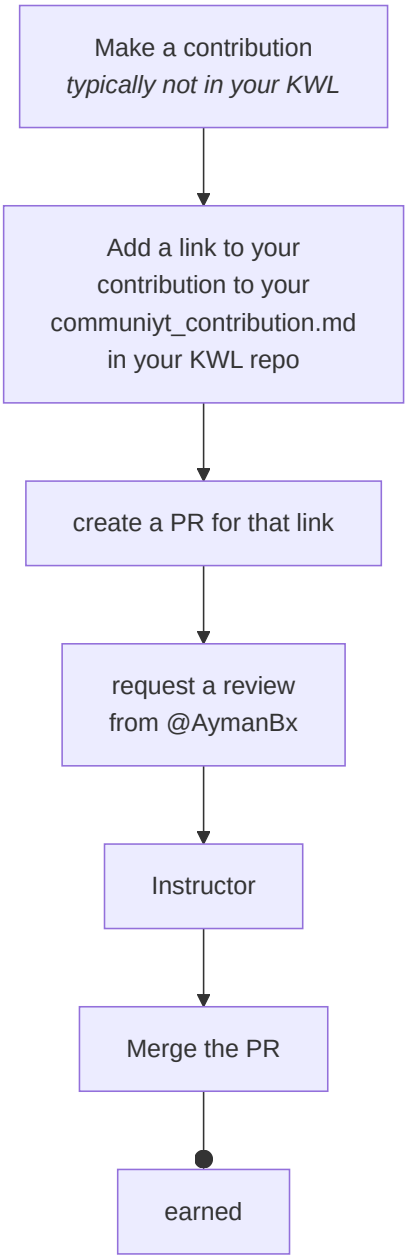
Manual logging

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 instructor will constitute credit for your grade
- request a review on that PR from @AymanBx

Important

You want one contribution per PR` for tracking



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**
>
> These calculations may change a litle bit and this page will be updated.
>
> What is on the Grading page will hold true, but the detailed calculation here will update a little bit in ways that provide some more flexibility.

▶ Show code cell source

Grade cutoffs for total influence are:

▶ Show code cell source

letter	threshold
F	0
D	106
D+	124
C-	142
C	192
C+	210
B-	228
B	246
B+	264
A-	282
A	300

The total influence of each badge is as follows:

▶ Show code cell source

	badge	complexity	badge_type
0	experience	2	learning
1	lab	2	learning
2	review	3	learning
3	practice	6	learning
4	explore	9	learning
5	build	36	learning

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	12 lab checkouts	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; instructor must approve)	9	event
Early bird	(review + practice) submitted by 9/26 >=5	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
Curiosity	at least 15 experience reports have questions on time (before notes posted in evenings; instructor will log & award)	9	event
Community Star	10 community badges	18	auto
Hack the course - Contributor - Build	1 build that contributes to the course infrastructure/website +1 community or review	18	event
Hack the course - Contributor - Explore	1 explore that contributes to the course infrastructure/website + 2 community, with at least 1 review	18	event
Hack the course - Critic	5 total community badge, at least 2 reviews of other course contributions	9	event

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.

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+

in these two examples, doing the work at the start of the semester on time and being attentive throughout increases the grade 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 can output as a yaml file, which is like a dictionary, for an example here we will use a dictionary.

see cspt docs for CLI version

```
example_student = {'experience' :22, 'lab': 13, 'review': 0,'practice': 18,
                    'explore': 3,
                    'build' :0,
                  'community': 0,
                  'hack':0,
                  'unstuck': 0,
                  'descriptive': 1,
                  'early': 1,
                   'question':10 }
```

```
badges_comm_applied = grade_calculation.community_apply(example_student)
badges_comm_applied
```

```
{'experience': 22,
 'lab': 13,
 'review': 0,
 'practice': 18,
 'explore': 3,
 'build': 0,
 'community': 0,
 'hack': 0,
 'unstuck': 0,
 'descriptive': 1,
 'early': 1,
 'question': 10}
```

```
grade_calculation.calculate_grade(badges_comm_applied)
```

```
'A'
```

```
grade_calculation.calculate_grade(badges_comm_applied,True)
```

```
300
```

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 will start by introducing some basics of GitHub and setting expectations for how the course will work. This will include how you are expected to learn in this class which requires a bit about how knowledge production in computer science works and getting started with the programming tools.

What tools do Computer Scientists use?

Next we'll focus in on tools we use as computer scientists to do our work. We will use this as a way to motivate how different aspects of a computer work in greater detail. While studying the tools and how they work, we will get to see how some common abstractions are re-used throughout the fields and it gives a window and good motivation to begin considering how the computer actually works.

Topics:

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

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

For each build:

- 1.5 hour for the proposal (including revisions)
- 22 hours for the project
- 30 min for the final reflection

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 2025.

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.

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

date	question	keyword	conceptual	practical	social	activity
2025--	Welcome, Introduction, and Setup	intro	what is a system, why study tools	GitHub basics	class intros	create kwl repo in github, navigate github.com...
2025--	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
2025--	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
2025--	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
2025--	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...
2025--	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...
2025--	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
2025--	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...
2025--	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
2025--	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
2025--	How can 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
2025--	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
2025--	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

date	question	keyword	conceptual	practical	social	activity
2025--	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...
2025--	What is an IDE?	IDE	IDE parts	compare and contrast IDEs	collaboraiton features, developer communities	discussions and sharing IDE tips
2025--	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
2025--	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
2025--	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
2025--	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...
2025--	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...
2025--	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
2025--	What *is* a computer?	architecture	physical gates, history	interpretting specs	social context influences technology	discussion
2025--	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
2025--	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
2025--	How does this all work together	review	all	end of semester logistics	group work final	review quiz, integration/reflection questions
2025--	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()
```

date	topic	activity
2025-01-27	unix philosophy	design a command line tool that would enable a...
2025-02-03	offline branches	plan for success, clean a messy repo
2025-02-10	git plumbing	git plumbing experiment
2025-02-19	scripting	releases and packaging
2025-02-24	GitHub Basics	syllabus quiz, setup
2025-03-03	os	hardware simulation
2025-03-10	tool familiarity	work on badges, self progress report
2025-03-17	remote, hpc	server work, batch scripts
2025-03-24	Machine representation	bits and floats and number libraries
2025-03-31	Compiling	C compiling experiments
2025-04-07	git plumbing	grade calculation script, self reflection
2025-04-14	working at the terminal	organization, setup kwl locally, manage issues
2025-04-21	hardware	self-reflection, work, project consultations

Support Systems

Mental Health and Wellness:

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, TELUS Health Student Support App, the Wellness Resource Center, the Psychological Consultation Center, the URI Couple and Family Therapy Clinic, and Well-being Coaching.

Academic Enhancement Center

Academic Enhancement Center (for undergraduate courses): All Academic Enhancement Center support services for Spring 2025 begin on January 27th and are offered at no added cost to undergraduate students. Visit AEC for more information about our programs described below. Appointments can be scheduled in TracCloud located in Microsoft 365.

- **STEM Tutoring**: Get peer tutoring for many 100 and 200 level STEM, Business, Nursing, and Engineering courses. Choose weekly or occasional sessions through TracCloud or visit the Drop-In Center in Carothers Library LL004. For

more details visit STEM & BUS Tutoring.

- **Academic Skills Development**: Meet one-on-one with a peer academic coach to build habits and strategies around time management, goal setting, and studying. Contact Heather Price for more information. Click here for more details. UCS 160 and UCS 161 are 1 credit courses designed to improve your academic skills and strategies. Consider enrolling in one of these courses! Contact David Hayes with any questions or to schedule a professional staff academic consultation. Click here for more details.
- The **Undergraduate Writing Center**: Receive peer writing support at any stage of your writing process. Schedule in-person or online consultations through TracCloud or stop by Roosevelt Hall Room 20 -new location! Click here for more details.

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 Resource Team. There you will also find people and resources to help.

Disability, Access, and Inclusion Services for Students Statement

This course is specifically designed to use universal design principles. Many of the standard accomodations that the DAI office provides will not apply to this course, because of how it is designed: there are no exams for you to get extra time on, and no slides for you to get in advance. However, I am happy to work with you to help you understand how to use the build in support systems for the course.

URI wide:

> 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.

Academic Honesty

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

- Using material, directly or paraphrasing, from published sources (print or electronic) without appropriate citation
- Claiming disproportionate credit for work not done independently

- Unauthorized possession or access to exams
- Unauthorized communication during exams
- Unauthorized use of another's work or preparing work for another student
- Taking an exam for another student
- Altering or attempting to alter grades
- Fabricating or falsifying facts, data or references
- Facilitating or aiding another's academic dishonesty
- Submitting the same work for more than one course without prior approval from the instructors

> **Tip**
>
> Most assignments are tested against LLMs and designed so that outsourcing it to an LLM will likely lead to a submission that is below the bar of credit.

AI Use

All of your work must reflect your own thinking and understanding. The written work in English that you submit for review and practice badges must 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 development environment 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.

> **Important**
>
> It is not okay to copy-paste and submit anything from an LLM chatbot interface in this course

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 by 1/3 of the maximum possible (2 explore or 1 build) for each infraction.

Attendance

“Attendance” is not explicitly checked, but participation in class through prismia is monitored, and lab checkouts and experience badges grade your engagement in the activities of lab and class respectively.

Viral Illness Precautions Statement

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. The Centers for Disease Control and Prevention (CDC) recommends that all people who are experiencing viral illness should stay home and away from others until symptoms improve and they are fever free (without medications) for 24 hours. They should take added precautions for the next 5 days.

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

Excused Absences

Absences due to serious illness or traumatic loss, religious observances, military service, 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), for any reason, you can follow the makeup procedures, no extensions will be provided typically for this; if extenuating circumstances arise, then ask Any instructor.

For *extended excused absences*, (3 or more classes) email Ayman 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 for excused absences. In your plan, include what class sessions you missed by date.

For unexcused absences, the makeup procedures apply, but not the planning assistance via email, only regularly scheduled office hours, unless you have class during all of those hours and then you will be allowed to use a special appointment.

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 announcements 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://compsys-progtools.github.io/spring2025/syllabus
> ```
>
> put this on a branch called `watch_community_badge` and title your PR “Community-Watch”

Help Hours

Day	Time	Location	Host
Monday	9am-12pm	Zoom	Ayman Sandouk
Tuesday	2pm-4pm	Tyler - Rm 139	Elijah Smith-Antonides
Wednesday	10am-12pm	Tyler - Rm 139	Trevor Moy

Online office hours locations and appointment links for alternative times 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

> **TLDR**
>
> Contribute a TLDR set of tabs or mermaid visual to this section for a community badge.

For assignment help

- use the badge issue for comments and @ mention instructors
- **send in advance, leave time for a response**
- **always** use issues in your repo for content directly related to assignments. If you push (changes to a repository) your partial work to the repository and then open an issue, we can see your work and your question at the same time and download it to run it if we need to debug something
- use issues or discussions 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)

> **Note**
>
> 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.

> **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

Today:

- intros
- what the *learning* goals of the course are
- see how in class time will work
- start learning git/github by doing

Not Today:

- syllabus review (on your own time/lab Monday)
- cours policy discussion (next week)

1.1. Introductions

- Ayman Sandouk
- Trevor Moy
- Elijah Smith-Antonides

1.2. Why think like a computer?

With Large Language Models (LLMs) able to write code from English (or other spoken languages, but LLMs are generally worse at non English)

Let's discuss some examples.

Many things in this course *are* things you will use **everyday** some of it is stuff that will help you in the trickest times.

What differentiates LLM (Large Language Modules) from Computer Scientists?

1.3. GitHub

- This class is not a GitHub class
- GitHub is the main tool that we will be using in this course
- Not expecting you to be familiar with it
- Homework is submitted through GitHub in a non-traditional way
- Great practice for real-life software development with a team or even individual work
- More on that later

I look forward to getting to know you all better.

1.4. Prismia

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

questions can be “graded”

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

Questions can be multi-choice

or open ended

And I can share responses, grouped up

1.5. 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

> as an aside another Professor describing what she does not like about learning management systems (LMS). Brightspace is one, she talks about Canvas in the post, but they are similar.

I will not chase you.

I do not judge your reasons for missing class.

- **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 longer absences, I will help you plan how to get caught up, and you must meet university criteria for excused absence

1.5.1. My focus is for you to learn

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

1.5.2. Learning comes in many forms

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

1.6. Learning is the goal

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

1.7. 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?

1.8. This is a college course

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

1.9. “I never use what I learned in college”

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

1.10. Learning is hard

1.11. How does this work?

1.11.1. In class:

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

1.11.2. Outside of class:

1. Read notes 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
4. Read articles/ watch videos to either fill in gaps or learn more details
5. Bring questions to class

1.12. Getting started

Your KWL chart is where you will start by tracking what you know now/before we start and what you want to learn about each topic. Then you will update it throughout the semester. You will also add material to the repository to produce evidence of your learning.

Accept the assignment to create your repository (aka repo)

We have a glossary!!

repository

1.12.1. What is a directory?

A “directory” in computing is a file system structure that acts like a container, organizing and managing files and other directories (often called folders) on a computer

pro tip: links are often hints or more information

```
Commits represent a message to your future self/teammates/viewers.
They mark what signifigant change has been made between one "checkpoint" in the status of a project and t
```

1.13. What is this course about?

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

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

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

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

1.13.1. How it fits into your CS degree

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

1.14. In your degree

this describes the BS; BA drops some of the math

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

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

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

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

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

1.15. GitHub Docs are really helpful and have screenshots

- editing a file
- pull request

they pay people to update them so I direct you to theirs mostly instead of recreating them

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 (editing a file step 5)
4. adding prior knowledge
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

join a team

we will use this for discussions and see more in lab, but having you join now makes it easier for me to make you a member and give you access to a special member view

1.16. Git and GitHub terminology

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

1.16.1. GitHub Actions

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

Navigate to your actions tab

1.16.2. 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

On the created PR, go to the `Files` section, edit the file, and commit the changes.

1.16.3. Fix your repo

So, it is apparently a weird bug in GitHub, that the actions were not working,

but it is easy (though a little annoying) to fix.

1.16.4. Make the edits

On each file in the .github/workflows folder that ends in .yml edit in some small way

> **Important**
>
> if the file has the word “reviewer” in, change the reviewer from “brownsarahm” to “AymanBx”

else: make any small change (eg add an additional blank line in a place where there is a blank line already).

1.16.5. Run forgotten badge action

1. Go to the actions tab of your repo
2. Select the action that has the name `Forgotten Badge (Late, but was in class)`
3. In the blue banner that appears click `Run Workflow`
4. leave the branch set to main
5. Enter the date `2025-01-23`
6. Wait a minute or so for it to run, when it has a green checkmark, go to your PR tab
7. select the PR with the title Experience Report 2025-01-23
8. Go to the files tab of that PR and edit it (use the 3 dots menu in the top right of the file box)
9. fill in the information and commit to the same branch (do not open an additional PR)
10. assign @instructors to review your PR.

For screenshots, see the Manually running a workflow, on GitHub

1.17. Prepare for next class

1. (for lab Monday) Read the syllabus section of the course website carefully and explore the whole course website
2. (for lab Monday) Bring questions about the course
3. (for class Tuesday) 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? (nothing written to submit, but you can use the issue to take notes if you would like)

1.18. Badges

Review | **Practice**

1. accept this assignment and join the existing team to get access to more features in our course organization.
2. Post an introduction to your classmates on our discussion forum (include link to your comment in PR comment, must accept above to see)
3. Read the notes from today's class carefully
4. Fill in the first two columns of your KWL chart (content of the PR; named to match the badge name)

1.19. We have a Glossary!!

For example, the term we used above:

repository

> **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

1.20. Questions after class

1.20.1. Should I be doing some practice/review badges between classes?

- Before each class, you need to complete the tasks assigned in the `prepare` issue
- At the end of each class, run the `experience report (in class)` action and fill out the file at your own time
- After each class, you get to pick one of the two (Practice/Review) and do that one. You only get graded for one of them

2. More orientation

Today we will:

- continue getting familiar with the structure of GitHub

- clarify more how the course will flow
- practice with new vocabulary

Last class 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-2025-01-28 issue and discuss the questions with your classmates at your table
4. If you have issues that currently say Error: not found (prepare 01-30 and practice 01-28)

*hint: my KWL repo URL is: `https://github.com/compsys-progtools/spring25-kwl-AymanBx`

What do you associate “learn new vocabulary” with?

Note: it is actually *always* the first step of learning, or joining a community.

What are GitHub issues for?

- [x] bug reporting and tracking
- [ ] proposing changes to the code by comparing two branches
- [ ] discussing things tangentially related to the code

What are GitHub issues for in our class?

- [ ] discussing things tangentially related to the code
- [ ] proposing changes to the code by comparing two branches
- [x] Assignment or issue that needs to be fixed in out repo/project

What are Pull Requests for?

- [ ] bug reporting and tracking
- [x] proposing changes to the code by comparing two branches
- [ ] discussing things tangentially related to the code

What are Pull Requests for in our class?

- [ ] bug reporting and tracking
- [x] Request an instructor to view my work and approve it or comment on it
- [ ] discussing things tangentially related to the code

Go to you PR tab

What is an experience badge?

- [ ] A way to prove I was in class (take attendance)
- [ ] A program that means at the end of the semester I get a medal for each one I have

- [x] A way to remind my future self and show my instructor what I've learned from class and ask questions

2.2. How do we work with experience badges

Checklist:

1. Merge prepare work into this PR
2. Link prepare issue to this PR
3. Complete experience report
4. Add activity completion evidence per notes

We fixed the `forgottenexperience.yml` file and then ran it manually.

2.3. Making up for action issue last week

Yesterday we fixed the issue with our actions and were able to run the Forgotten badge action

What date did you make your experience badge for?

1. Date is supposed to be 2025-01-23
2. Redo and copy the content from the old one

Then we edited the file it created to add a title on the line with one `#` (should be line 10) uisng the 3 dots menu in the top right of the file on the `files changed` tab of the PR.

Which of the following is true? *hint: look at your experience badge from yesterday (and chat with neighbors)*

- [ ] once a PR is open you cannot add commits to either branch involved
- [ ] once a PR is open if you add commits to the proposing branch, you have to open a new PR
- [x] once a PR is open if you add commits to the proposing branch, they are visible in the existing PR

2.3.1. Remember

Your experience pull request (badge) already had a changed file in it And then you edited the file again to answer the existing prompts

> **Note**
>
> When you add more commits to a branch that has a PR, it automatically updates the PR.

2.3.2. Where does this file exist?

Find the message that says `github-actions wants to merge 1 commit into main from experience-<somenumber>`

2.3.3. What does this experience- represent?

```
This is a branch kf your repo that has one file that is new (different from the main branch where that fi
This file is the experience report that you are asked to fill out after every lecture
```

2.3.4. How do I know the location of the file?

> **Note**
>
> Here we are learning *by example* and then *synthesizing* that into more concrete facts.

my goal is to teach you to get better at learning in that way, bc it is what employers will expect

To do this:

- I 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.

There is a time breakdown in the syllabus that suggests and recommends a good way to distribute your time in the semester for the class.

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

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

2.4. Programming is Collaborative

There are two very common types of collaboration

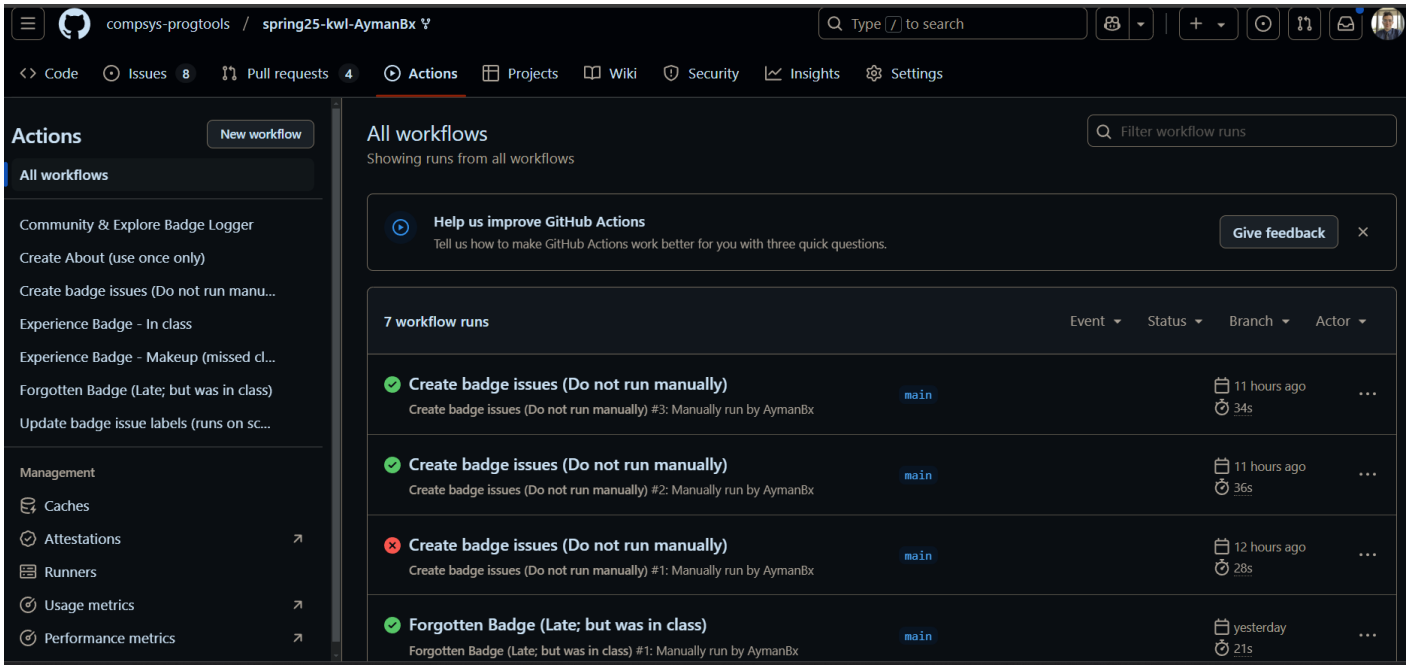
- code review (working independently and then reviewing)
- pair programming (sitting together and discussing while writing)

We are going to build your skill in the *code review* model. This means you need to collaborate, but collaboration in school tends to be more stressful than it needs to. If students have different goals or motivation levels it can create conflict. So there will be some chances for collaboration where people can show up at the level they want without impacting others.

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

2.5. GitHub Actions Tab

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



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

On my actions page, in the screenshot above, how many **successful workflow runs** are shown?

- [ ] 4
- [ ] 5
- [x] 3

On my actions page, in the screenshot above, how many **total workflow runs** are shown?

- [x] 4
- [ ] 5
- [ ] 3

On my actions page, in the screenshot above, how many workflows are **available to run**?

- [ ] 4
- [x] 7
- [ ] 5

Your time to practice:

1. Navigate to your actions tab
2. Run the “Create About (use only once)” Action.

You should have:

- fixed action files
- Labs issues closed
- experience report for 2025-01-23

2.6. Prepare for next class

1. View all existing issues (Prepare, View)
2. Select one out of the two for each date and complete the tasks in it. (Pick wether you want to do practive or review for a certain date)
3. Make sure you ask for out review on any open pull request (other than feedback)
4. Choose where you want to save files for this class locally on your computer and make note of that location. (nothing to submit; but we will be working locally and you need to have a place)
5. Think about how you think about files and folders in a computer. What do you know about how they are organized? how they're implemented? (nothing to submit)

2.7. Badges

Review **Practice**

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)
2. "Watch" the course website repo, specifically watch Releases under custom (to get notifications)
3. map out your computing knowledge and add it to your kwl chart repo. this can be an image that you upload or a text-based outline in a file called prior-knowledge-map. (optional) try mapping out using mermaid syntax, we'll be using other tools that will faciltate rendering later (what we will learn will connect a lot of ideas, mapping out where you start, sets you up for success)

2.8. Questions after class

3. Why Systems?

3.1. 5 minutes

3.2. Questions?

Issues, Pull requests, Prepare, Practice, Review, Lab…

3.3. Working offline

Today more clear motivation for each thing we do and more context.

Today we will learn to work with GitHub offline, this requires understanding some about file systems and how content is organized on computers.

We will learn:

- relative and absolute paths
- basic bash commands for navigating the file system
- authenticating to GitHub on a terminal
- how to clone a repo
- how fetch and checkout work

3.4. 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` (preferred)
- if you have WSL (if you do not, no need to worry) you should be able to set your linux shell to `bash` (I believe it already is set to bash)

If you use `pwd` you can see your current path

```
pwd
```

```
Users/ayman
```

It outputs the absolute path of the location that I was at.

we start at home `~`

We can **c**hange **d**irectory with `cd`

---

if we use `cd` without a path, it goes back to home `cd ~` would do the same

---

We can **m**a**k** a new **dir**ectory with `mkdir`

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

You might:

- make a systems folder in your Documents folder
- make an inclass folder in the CSC311 folder you already made
- use the CSC311 folder as your in class working space

If I run the following commands, what do I expect as the output?

```
cd
pwd
```

- [ ] cannot tell, do not know where you started
- [x] /c/Users/ayman
- [ ] /c/Users/ayman/Documents
- [ ] the same as whereever you were before

When you use `pwd` what type of path does it return?

- [x] absolute
- [ ] relative

The first slash `/` represents the `/(root)` directory, which is the starting place for the search

---

But what does it mean when a path starts with `.` or `..`?

To go back one step in the path, (one level up in the tree) we use `cd ..`

```
cd ..
```

`..` is a special file that points to a specific relative path, of one level up. (In other words, parent folder/directory)

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

---

Did you notice the dieffernce?

```
cds Documents/
```

```
bash: cds: command not found
```

notice that command not found is the error when there is a typo

```
cd Documents/
pwd
```

```
/c/Users/ayman/Documents
```

```
mkdir systems
```

```
/c/Users/ayman/Documents/systems
```

```
cd ..
pwd
```

```
/c/Users/ayman/Documents/
```

If we give no path to `cd` it brings us to home.

```
cd
pwd
```

```
/c/Users/ayman
```

Then we can go back.

```
cd Documents/systems/
pwd
```

```
/c/Users/ayman/Documents/systems
```

Do you have any content in the folder?

```
ls
```

Could be empty of if you had created it before you would see the content you had in it

We can use two levels up at once like this:

```
cd ../../
pwd
```

```
/c/Users/ayman
```

```
cd Documents/systems/
```

We can the `tab` to complete once we have a unique set of characters. If what we have is not unique enoguh yet, bash will do nothing when you press tab once, but if you press it multiple tiems it will show you the options:

```
cd Do
```

Press `tab` twice. The consol outputs:

```
Documents/ Downloads/
cd Do
```

```
cd Doc
```

Press `tab`... The consol auto completes the path

```
cd Documents/
```

What character is always at the start of absolute paths?

- [ ] :
- [x] /
- [ ] *
- [ ]]

What type of path is `../../Downloads` ?

- [ ] absolute
- [x] relative

3.5. A toy repo for in class

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

---

Preferred:

1. view the template
2. click the green “use this template” button in the top right
3. make `compsys-progtools` the owner
4. set the name to `gh-inclass-<your gh username>` replacing the `<>` part with your actual name

---

Backup: accept the assignment

3.6. Authenticating 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

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

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 with `gh auth login`

---

we will do opiton 2 for today

3.6.1.1. GitBash (windows mostly)

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

3.6.1.2. Native terminal (MacOS X/Linux/WSL)

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

3.6.1.3. 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`

3.6.2. Cloning a repository

We will create a local copy by cloning

Type `pwd` first to make sure you're in the Systems folder

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

```
Cloning into 'gh-inclass-AymanBx`...
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.
```

Confirm it worked with:

```
ls
```

```
gh-inclass-AymanBx
```

We see the new folder that matches our repo name

3.7. What is in a repo?

We can enter that folder

```
cd gh-inclass-AymanBx/
```

When we compare the local directory to GitHub

```
ls
```

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

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

```
ls -a
```

```
.               .git
..              .github
```

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

3.8. How do I know what git knows?

`git status` is your friend.

```
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 (more on that later …).

3.9. Making a branch with GitHub and working offline

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

“create an about file”

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-AymanBx
 * [new branch]      1-create-an-about-file -> origin/1-create-an-about-file
```

We can look at the repo to see what has changed.

```
git status
```

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

nothing to commit, working tree clean
```

This says nothing, because remember git status tells us the relationship between our working directory and the .git repo.

Next, we switch to that branch.

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

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

---

and verify what happened

```
git status
```

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

nothing to commit, working tree clean
```

Run your Experience Badge (inclass) action. Github how to

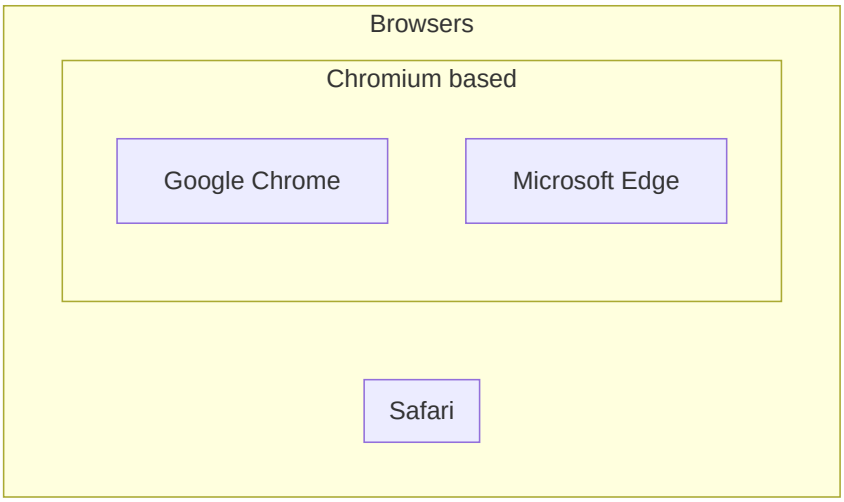
1. Go to the Actions tab
2. Click the Experience Badge (inclass) on the left hand side
3. Click the run workflow button on the right
4. Click the run workflow button in the popup

3.10. 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. Put this in a file called `terminal-vocab.md` on a branch linked to this issue.
2. Check your kwl repo before class and see if you have recieved feedback, reply or merge accordingly.

3.11. Example

Example “venn diagram “ with mermaid subgraphs



3.12. Badges

Review **Practice**

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

1. Read the notes. If you have any questions, post an issue on the course website repo.
2. Using your terminal, download your KWL repo. Include the command used in your badge PR.
3. Try using setting up git using your favorite IDE or GitHub Desktop. Make a file gitoffline.md and include some notes of how it went. Was it hard? easy? what did you figure out or get stuck on? Is the terminology consistent or does it use different terms?
4. **lab** Explore the difference between git add and git commit: try committing and pushing without adding, then add and push without committing. Describe what happens in each case in a file called gitcommit.md. Compare what happens based on what you can see on GitHub and what you can see with git status.

3.13. Questions after class

4. Git Offline

4.1. Absolute Path vs Relative Path

Navigate to your inclass repo on your terminal

```
pwd
```

```
/c/Users/ayman
```

Which command will help me navigate between folders?

- [ ] ls
- [x] cd
- [ ] pwd
- [ ] mkdir

> **Note**
>
> Remember: cd stands for **c**hange **d**irectory

What type of path do I use with the command `cd`?

- [ ] absolute path only
- [ ] relative path only
- [x] Either one will work

Using absolute path means you know how to navigate to your desired folder starting from the root `/`

The `/` is the starting point to where all your files and folders can be found

---

Using you **relative** path, means that you know how to reach your desired folder relative to where you are right now

How do I get to Tyler 052?

Absolute path: USA/Rhode Island/Kingston/Flag rd/Green house rd/Red building/052 Relative path: leave Ranger 302 (…)/ leave Ranger Hall (…)/ Through quad/ pass by engineering/ Red building/ 052

The `/` between folder names means “from here, go to” We can separate the `cd` command that contains `/` into multiple commands

---

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

```
pwd
```

```
/c/Users/ayman/Documents/systems
```

Let’s try another way

```
cd ../../
pwd
```

```
/c/Users/ayman
```

```
cd Documents/systems/
```

```
pwd
```

```
/c/Users/ayman/Documents/systems
```

We got to the same destination

Watch your steps carefully for this one (Don't hit `Enter`)

```
cd gh-
```

Hit `Tab`

```
cd gh-inclass-AymanBx
```

Now you can hit `Enter`

> **Note**
>
> The terminal determined that the existing folders/files that begin with "gh-" are limited to only one and helped you complete it with the press of `tab`

getting to GitHub from your local system

Now on the other side of things, navigate to your inclass repo on GitHub

the step below requires that you have the `gh` CLI.

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

What files/folders can you see on there?

```
.github/workflows
```

How many branches do you have?

```
main
1-create-an-about-file
```

Select the 1-create-an-about-file

Same files so far (No changes made yet)

Back to your terminal

Let's go back to comparing files between online and local repos

```
ls -a
```

> **Note**
>
> We used `-a` as an option for showing us hidden files/folders (anything that starts with a `.`)

```
.           .git
..          .github
```

We'll notice a few things here

We have three extra files/folders

We know what `..` refers to by now. It's a pointer to the parent direcctory

The `.` is a pointer to where I am right now (the current directory)

Let's test out this theory

```
pwd
```

```
/c/Users/ayman/Documents/systems/gh-inclass-AymanBx
```

```
cd .
pwd
```

```
/c/Users/ayman/Documents/systems/gh-inclass-AymanBx
```

Our place didn't change. It checks out!

4.2. .git

The third difference was the `.git` folder

`.git` folder is created by the `git` tool that we use on our terminal to hold special information about the project we're working on

---

Everytime we start a command with the word `git` we're calling that tool and then telling it what we want it to do

---

So far we saw `**git** status` and `**git** fetch`

`status` & `fetch` were the commands that we asked git to execute

```
.git is a black box ( until furhter notice ;-) )
It holds tracking information about what changes were made locally and online
```

We use `git fetch` to update the .git box with changes that happed on the GitHub repo

Let's try it

```
git fetch
```

If you got nothing then your .git box is already up to date with the online **repo** (project)

Check the current status of your project

```
git status
```

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

nothing to commit, working tree clean
```

`git status` is your friend, we will use it all the time to keep track of our movements

Notice two key lines here

`Your branch is up to date with 'origin/1-create-an-about-file'.` is telling you that your local project matches the online one at the moment (as far as it knows)

More importantly, `nothing to commit, working tree clean` is telling you that your local repo (everything out side of the .git folder) matches the tracking information of git (everything inside the .git folder)

Much more on that later

Lastly, one more small thing we noticed

Github showed us `.github/workflows` whereas `ls -a` only showed my `.github`

As discussed, the `/` tells us that this there is a file/folder within a folder

Github combined them in one because there are no other files/folders on the .github folder

How do we prove it?

```
ls .github
```

```
workflows/
```

Listing the contents of .github showed us that it contains the folder “workflows” with it

> **Note**
>
> Conclusion: The `ls` command can take an **argument** that is a path and it will list the contents of the folder passed in that path for us

4.3. Creating a file on the terminal

The `touch` command creates an empty file.

```
touch about.md
```

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

```
ls
```

```
about.md
```

```
git status
```

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

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

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

Notice: Working tree is not clean anymore.
There is one “untracked file”

This means the working directory (everything outside .git) had changes but those changes were not **added** to the .git box to be **tracked**

```
nano about.md
```

What year are you in? When do you expect to graduate?

Ctrl + s (Only windows users) Ctrl + x Enter (Only mac users would need this)

---

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.

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.

`cat` con**cat**enates the contents of a file to standard out, where all of the content that is shown on the terminal is.

> **Note**
>
> Standard out is a special file in your device that your programs/commands executed will put their output into. The terminal prints the content of standard out (aka: std out), hence we get the output printed on the terminal

```
cat about.md
```

and we can see the contents

```
git status
```

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

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

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

4.4. git add

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 everything in the current working directory.

```
git add .
```

And again, we will check in with git

```
git status
```

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

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

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.

Notice: git tells you that if you changed your mind on that file and you don’t want it staged to be commited to the next “checkpoint” anymore, you can simply take it out of the staging area using the command `git restore --staged <file_name>`

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

4.5. git commit

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`.

> **Note**
>
> A commit is a “checkpoint” in your project that you want to save, to be able to go back at any point in time. A commit saves the status of all the files in the project that had been modified since the last check point and **staged** for the new commit

```
git commit -m "create about - closes #1"
```

We used a closing keyword so that it will close the issue.

> **Warning**
>
> When you make your first commit you will need to do some config steps to set your email and user name.

```
[1-create-an-about-file c7375fa] create about - closes #1
 1 file changed, 3 insertions(+)
 create mode 100644 about.md
```

---

one more check

```
git status
```

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

nothing to commit, working tree clean
```

> **Note**
>
> Now your working tree is clean again, but your **local** branch (as reflected inside the black box .git) now looks different from what’s on the online GitHub repo (Sometimes referred to as the “Upstream”)

4.6. git push

Git suggests that we use the `push` command to update the online repo with the local one.

And push to send to github.com

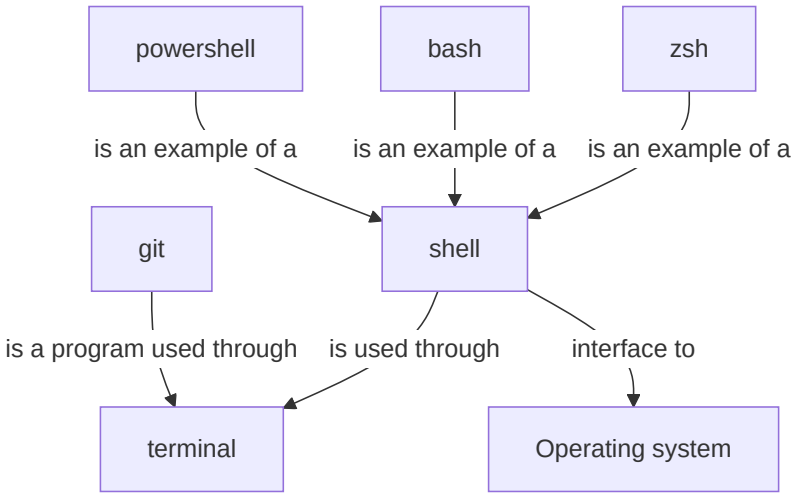
```
git push
```

```
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 341 bytes | 341.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/compsys-progtools/gh-inclass-AymanBx
   98ca2d6..9e8a
```

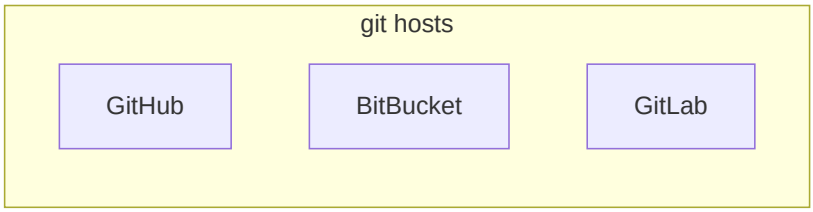
Now check out your online repo. Make sure your on the “1-create-an-about-file” branch

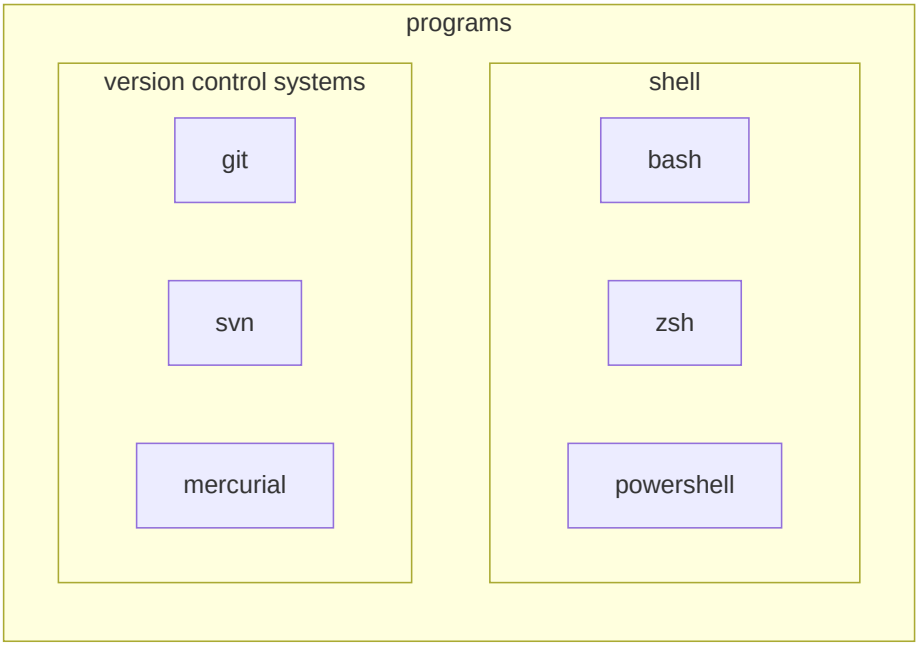
about.md should exist now on there!

4.7. Summary



Another way to think abut things (and adds some additional examples to help you differentiate between categories and examples of categories)





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

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

command	explanation
`push`	send to the remote

4.8. Prepare for next class

- [ ] Make sure you're inclass repo has two branches on github `main` & `1-create-an-about-file`
- [ ] Make sure you're inclass repo has two branches locally as well using the command `git branch` on your terminal when you're inside the inclass folder
- [ ] Make sure you can see the `about.md` file on github when you select the `1-create-an-about-file` branch and not the main branch

4.9. Badges

Review **Practice**

- [ ] Review the notes from 02-04
- [ ] Ask questions about what we did in the Experience report for 02-04

4.10. Questions after 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.

Ti

You
on y

Minimum Rows

> **Warning**
>
> To be updated

Required Files

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

date	file	type
2025-01-28	brain.md	/_practice
2025-01-30	gitoffline.md	/_review
2025-01-30	gitoffline.md	/_practice

Team Repo

> **Warning**
>
> We will not use this in 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.

For the glossary, follow the jupyterbook syntax.

For the cookbook, use standard markdown.

to denote code inline `use single backticks`

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

to make a code block use 3 back ticks

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

To nest blocks use increasing numbers of back ticks.

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

Collaboration

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

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

Peer Review

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

In your summary comments answer the following:

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

Choose an action:

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

Review Badges

Review After Class

After each class, you will need to review the day's material. This includes reviewing prismia chat to see any questions you got wrong and reading the notes. 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, they represent the minimum bar for C level work.

Prepare for the next class

These tasks are usually not based on material that we have already seen in class. Mostly they are to have you start thinking about the topic that we are *about* to cover before we do so. Often this will include reviewing related concepts that you should have learned in a previous course (like pointers from 211) Getting whatever you know about the topic fresh in your mind in advance of class helps your brain get ready to learn the new material more easily; brains learn by making connections.

Other times prepare tasks are to have you install things so that you can engage in the class.

The correct answer is not as important for these activities as it is to do them **before class**. We will build on these ideas in class. These are evaluated on completion only[1], but we may ask you questions or leave comments if appropriate, in that event you should reply and then we will approve.

---

[1] you will get full credit as long as all of the things are *done in good faith* even if not correct. However if it looks like you tried to outsource (eg to LLM) or plagiarize a solution, you will not earn credit for that.

Practice Badges

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

Practice badges are a chance to first review the basics and then try new dimensions of the concepts that we cover in class. After each class, you will need to review the day's material. This includes reviewing prismia chat to see any questions you got wrong and reading the notes. The practice badge will also ask you to apply the day's material in a similar, but distinct way. They represent the minimum bar for B-level understanding.

KWL File List

Explore Badges

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

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

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

How do I propose?

Create an issue on your kwl repo, label it explore, and "assign" @AymanBx.

In your issue, describe the question you want to answer or topic to explore and the format you want to use. There is no real template for this, it can be as short as one sentence, but there may be follow up questions.

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.
- If you do the work elsewhere, log it like a community badge but in a file called `external_explore_badges.md`

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

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

### Method

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

### 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:

- any code or text should be managed with git (can be GitHub or elsewhere)
- if you write any code it should have documentation
- if you do experiments the results should be summrized
- if you are researching something, a report should be 2-4 pages, plus unlimited references in the 2 column ACM format.

This guidance is generative, not limiting, it is to give ideas, but not restrict what you *can* do.

Updates and work in Progress

These can be whatever form is appropriate to your specific project. Your proposal should indicate what form those will take.

Summary Report

This summary report will be added to your kwl repo as a new file `build_report_title.md` where `title` is the (title or a shortened version) from the proposal. Use the template below for the summary report.

```
# <your project title> Summary Report

## Abstract
 <!-- a one paragraph "abstract" type overview of what your project consists of.  This should be written

## Reflection
 <!-- a one paragraph reflection that summarizes challenges faced and what you learned doing your project

## Artifacts

<!-- links to other materials required for assessing the project.  This can be a public facing web resour
```

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

Your Profile (CV/Resume) Website

Use a static site generator, like one of the below.

Astro

- https://astro.build
- https://docs.astro.build/en/getting-started/
- requires npm installation

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

Auto-approved proposals

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

Add docs to another project

You can add documentation website to another project

```
## Project Docs

Add documentation website for <code proejct>.

### Objectives

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

This project will provide information for a user to use <the project> The information will live in the re

### Method

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


### Deliverables


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


### Milestones

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

Developer onboarding

You can add documents that provide a developer onboarding experience to other code you have written

```
## Developer onboarding

Add developer onboarding information to <inssert project title here>

### Objectives

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

This project will provide information for a potential contributor to add new features to a code base. The

### Method

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

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
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 -->
```

Project Examples

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

Syllabus and Grading FAQ

How much does activity x weigh in my grade?

How do I keep track of my earned badges?

Also, when are each badge due, time wise?

Who should I request to review my work?

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

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

What should work for an explore badge look like and where do I put it?

Git and GitHub

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

My command line says I cannot use a password

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

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
```

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

Changing paths on windows

To edit a path on windows, go to the search bar and type ‘edit environment variables’, click the environment variable button, click on ‘path’ then new, then insert the new path

Avoiding windows security block

The closest thing to work around the security block is to exclude files, to exclude a file, take note of the file and know where to find it, go to windows security, virus protection and threat protection, scroll down to exclusions, add or exclude folders, then add the specific folder that is getting blocked

Glossary

> **Tip**
>
> 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.

detached head

a state of a git repo where the head pointer is set to a commit without a branch also pointing to the commit

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

.gitignore

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

git 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

when two branches to be merged edit the same lines and git cannot automatically merge the changes

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

path

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 `./`

release

a distribution of your code, related to a git tag

remote

a copy of the repository hosted on a server

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

tree objects

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

yml

see YAML

YAML

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

zsh

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

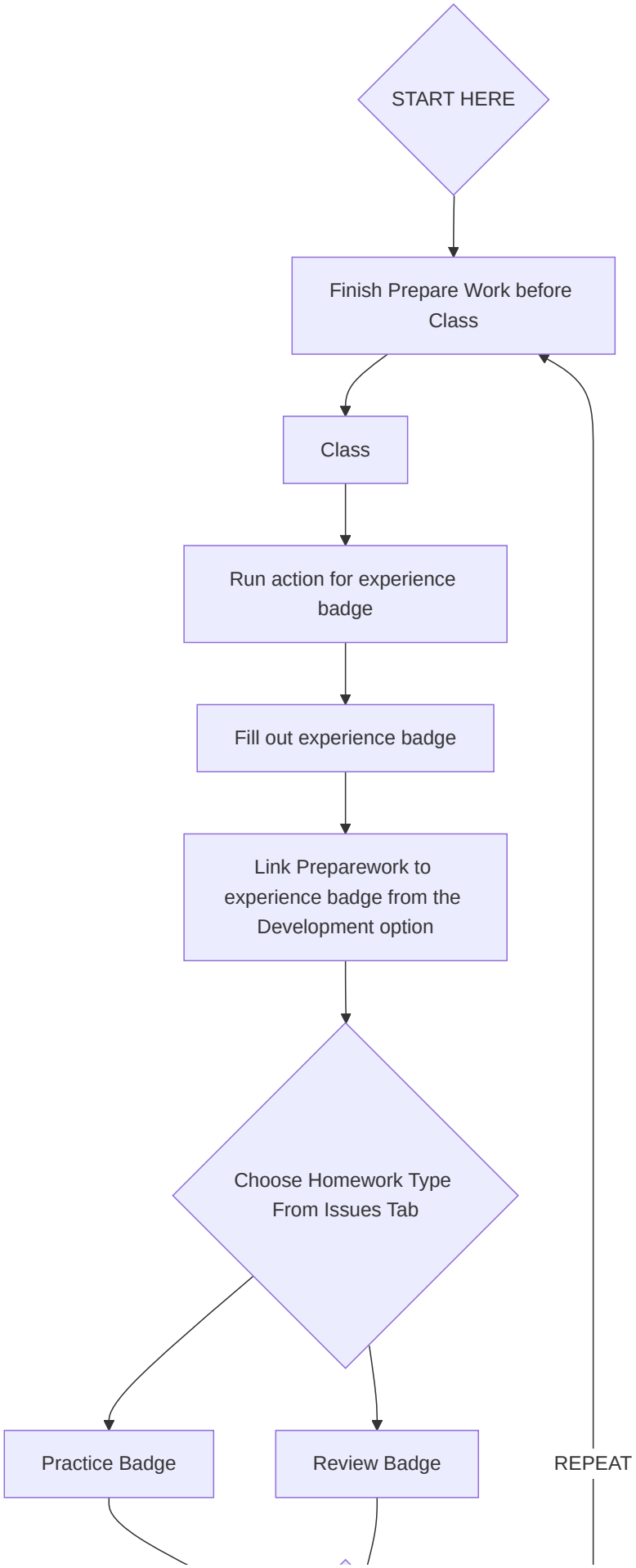
General Tips and Resources

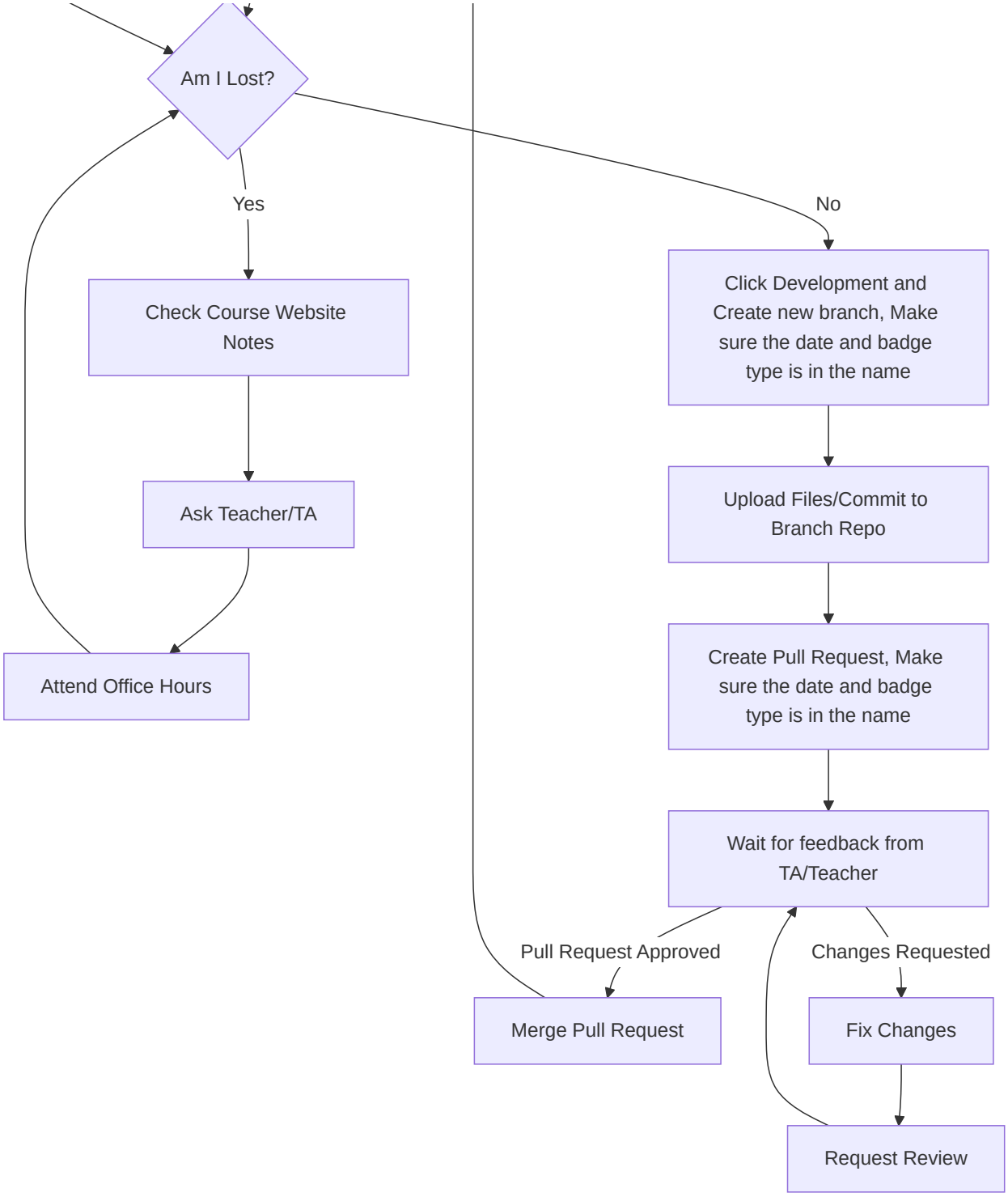
This section is for materials that are not specific to this course, but are likely useful. They are not generally required readings or installs, but are options or advice I provide frequently.

on email

- how to e-mail professors

Class Workflow


START HERE
Finish Prepare Work before Class
Class
Run action for experience badge
Fill out experience badge
Link Preparework to experience badge from the Development option
Choose Homework Type From Issues Tab
Practice Badge
Review Badge
REPEAT



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 bo
programı
Program
available
that linke

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

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.

Repository specific page

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)

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

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.

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.

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

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.

README file

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

Language/Shell Specific References

- bash
- C
- Python

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
`echo 'message'`	repeat ‘message’ to stdout
`>`	write redirect
`>>`	append redirect
`rm file`	remove (delete) `file`
`cat`	concatenate a file to standard out (show the file contents)

git commands

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
`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
`git commit -a -m 'msg'`	the `-a` option adds modified files (but not untracked)

Getting Help with Programming

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

Asking Questions

One of my favorite resources that describes how to ask good questions is this blog post by Julia Evans, a developer who writes comics about the things she learns in the course of her work and publisher of wizard zines.

Describing what you have so far

Stackoverflow is a common place for programmers to post and answer questions.
As such, they have written a good guide on creating a minimal, reproducible example.

Creating a minimal reproducible example may even help you debug your own code, but if it does not, it will definitely make it easier for another person to understand what you have, what your goal is, and what's working.

Getting Organized for class

The only **required** things are in the Tools section of the syllabus, but this organizational structure will help keep you on top of what is going on.

Your username will be appended to the end of of the repository name for each of your assignments in class.

File structure

I recommend the following organization structure for the course:

```
CSC3392
  |- kwl-
  |- gh-inclass
  |- semYYYY
  |- ...
```

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 `compsys-progtools` organization as the owner, not your personal acount. Since your account is not the owner, they do not show on your profile.

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.

More info on cpus

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

Windows Help & Notes

CRLF Warning

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

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

git documentation of the feature

Jupyter Book - Issues During or After Installation

1. **Check Python Installation**:
   Run `python --version`. If it shows a version, continue. If not, install Python from https://www.python.org/downloads/ If you get a "Permission Denied" message, see the "Adding Permissions" section below If you know python is installed, see the "Checking Paths" section below
2. **Install Jupyter Book**:
   Ensure Jupyter Book is installed using `pip install -U jupyter-book`. If `jupyter-book --version` returns then "command not found," see "Checking Paths" below If you get a "Permission Denied" message, see the "Adding Permissions" section below
3. **Check Installation Errors**:
   If there were no errors during installation, skip to Step 4.
   If there were errors with a path (e.g., missing "Scripts" folder), see the "Checking Path" section.
4. **Check for Directory**: Ensure the following directories exist: (Can check through File Explorer or through terminal)

   ```
   C:\Users\[YOUR USERNAME]\AppData\Roaming\Python\Python[VERSION#]\
   C:\Users\[YOUR USERNAME]\AppData\Roaming\Python\Python[VERSION#]\Scripts\
   ```

   If it does, move on If not, ensure that Python was installed correctly from the website download, NOT the Windows Store

5. **Final Troubleshooting**:
   If issues persist, contact your Professor or TA for help! :)

Checking Paths

If you get a `Command Not Found` message when trying to run `python` or `pip`, most likely your environment variable paths missing. First ensure the following directories exist: (Can check through File Explorer or through terminal)

```
C:\Users\[YOUR USERNAME]\AppData\Roaming\Python\Python[VERSION#]\
C:\Users\[YOUR USERNAME]\AppData\Roaming\Python\Python[VERSION#]\Scripts\
```

If they do, there are two methods to ensuring the path variables exist and adding them if not:

Method 1

1. Go to your taskbar Search
2. Search for and open “Edit the system environment variables”
3. Click “Environment Variables…” in the window that opened
4. Click “Path” line then “Edit…”

Method 2

1. Press `Windows + R`, type `sysdm.cpl`, and press Enter.
2. Go to the **Advanced** tab → **Environment Variables**.
3. Add the path containing the “Scripts” folder to the `Path` variable. Save and exit.
4. Reopen Git Bash and try `jupyter-book --version`. If it works, you’re done!
    - If “Access denied” occurs, try `sudo jupyter-book --version`. Windows Defender may prompt you to unlock the file. After unlocking, try the command again.

Adding Permissions

If you get a `Permission Denied` message when trying to run commands, Windows Security is blocking it.

- If you get a pop-up notification when trying to run a command, simply click “unblock” each time
- If you don’t get a pop-up notif, follow the steps below to add an exclusion for Python

1. Go to your taskbar Search
2. Search for and open “Windows Security”
3. Go to the “Virus & threat protection” section on the left
4. Under “Virus & threat protection settings” click “manage settings”
5. Scroll down to the “Exclusions” section and click “Add or remove exclusions”
6. Click “Add an exclusion”, then “Folder”
7. Find and select the folder Python installed in
    - Should be `C:\Users\[YOUR USERNAME]\AppData\Roaming\Python\`