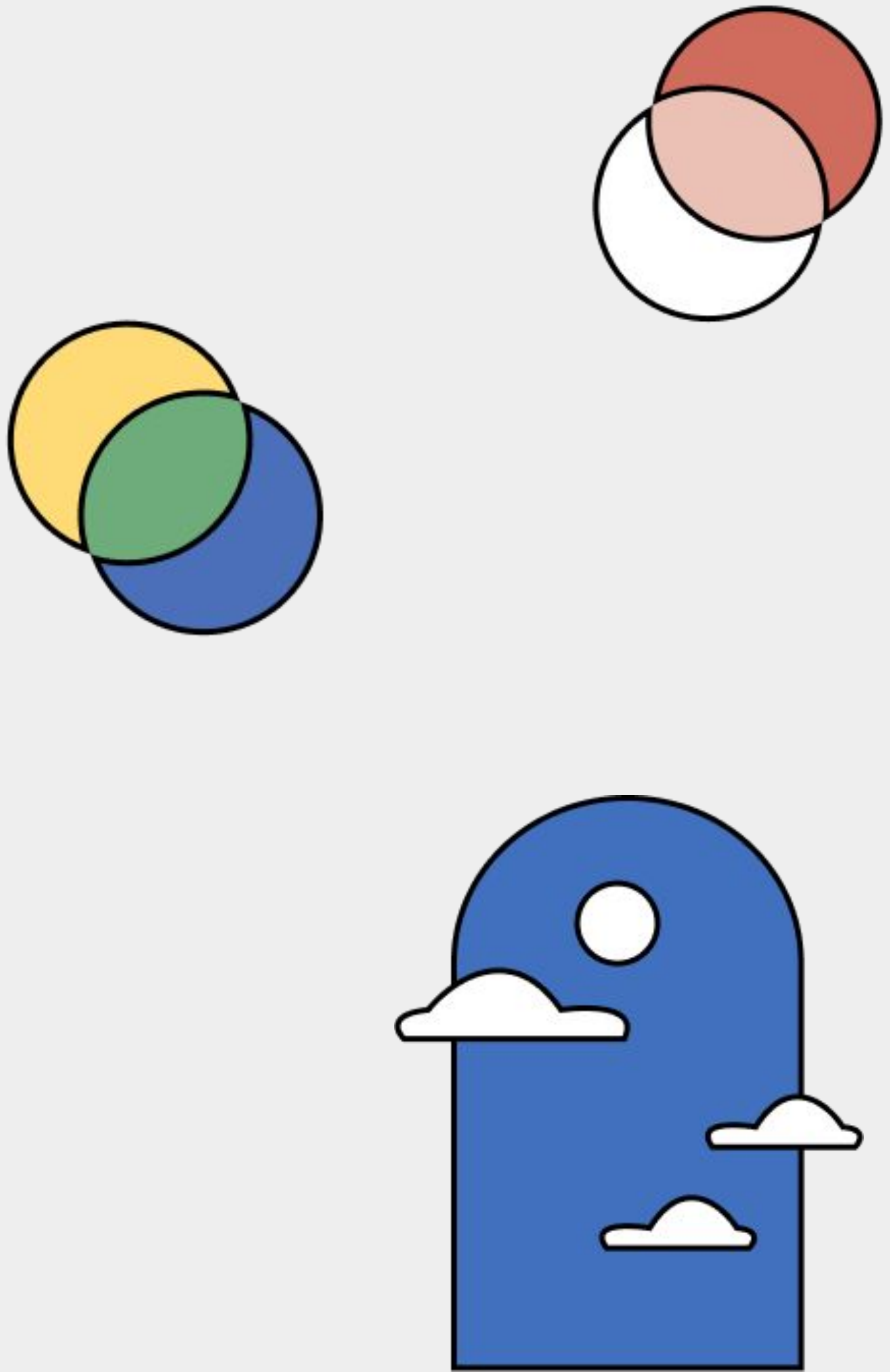
spark

Git Workshop

Spring 2022



Hello!

Matthew Dong

Spark Blue Instructor

Yuhan Liu

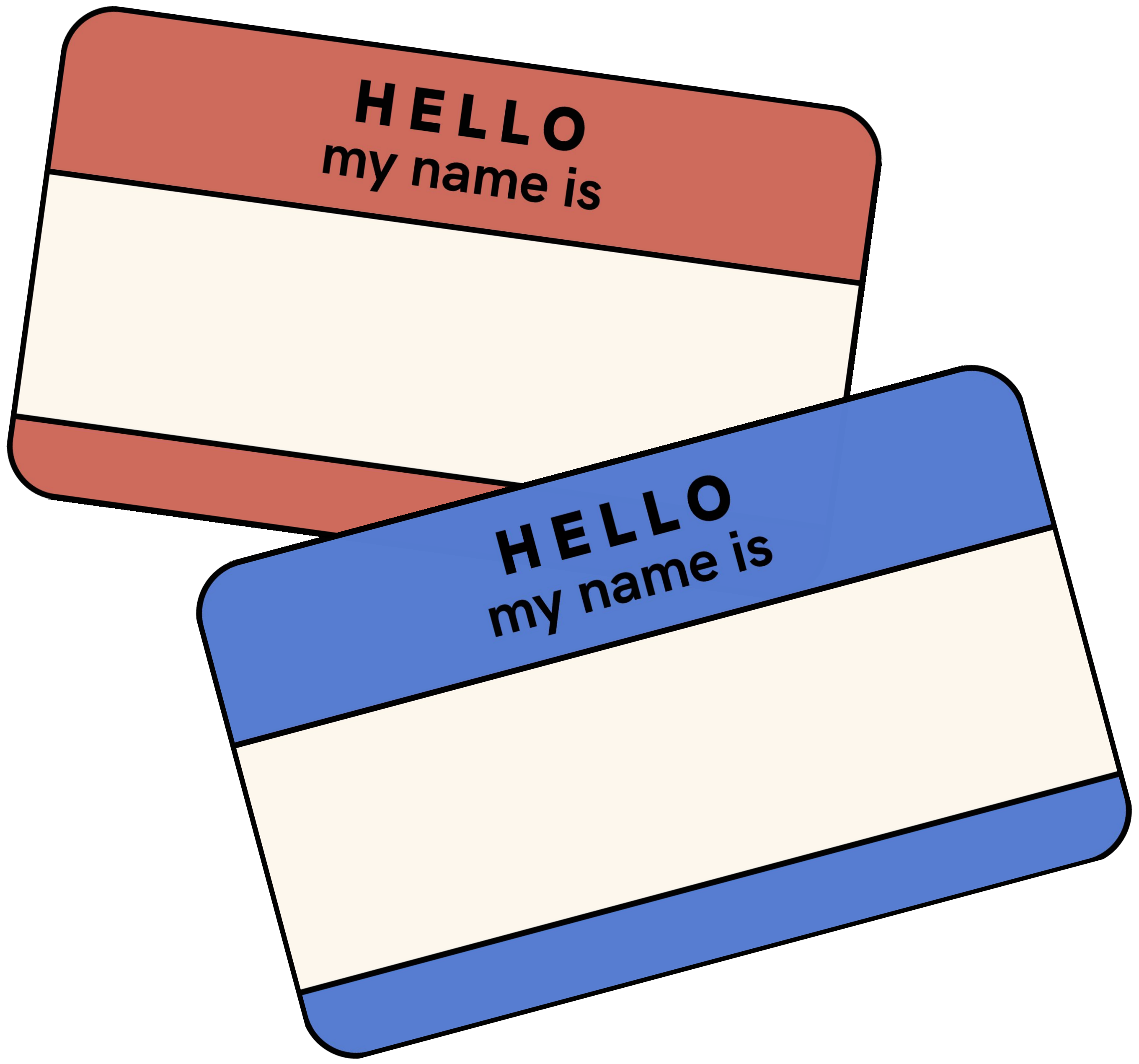
Spark Blue VP

Logistical Expectations

- **Communication**
  - In general, slack all of Yuhan, Grace, & Christina in a single thread for all communication (unless personal)
- **Attendance**
  - Come prepared to weekly meetings (bring laptop + well-rested brain) and be prepared to participate/follow along
  - Let Yuhan, us know in a single Slack thread **at least 24 hours in advance** about any conflicts (unless COVID-related or emergency)
- **Deliverables**
  - Please turn in your deliverables by the stated deadline!
  - This isn't class, so **please let us know if you need an extension** on an deliverable or **have any concerns**
    - But note that deliverables build on each other, so if you get super behind on one, you'll get a slow start to the next :(
- **Are you stuck or have any questions?**
  - **#spark-overflow, #sp22-blue-dev**
  - Slack us / instructors!! (include everyone in thread for faster response time)

Introductions

- Name
- Year
- Major
- Spring-related vibes question



Getting Started

Overview

- What is Git?
- What is GitHub?
- Why do we need it?

What Exactly is Git?

Git is a software for tracking changes in files.

- Programmers use git to track changes made to code.
- Git synchronizes code between different people.

What Exactly is GitHub?

GitHub is a user interface, or UI, wrapper around Git, much like Spotify is a UI wrapper around music. In these cases, a core technology (Git; music) is wrapped in a graphical user interface (GitHub; Spotify).

- Sites like GitHub host Git repositories, or files that are tracked by Git.

Why do we need Git?

- Git allows collaborators to make independent changes to their file version by merging the changes into one file.
  - By pulling (accessing the file) and pushing (updating the file), both you and your collaborators have access to the most up-to-date version.

Why do we need Git?

- Git allows collaborators to make independent changes to their file version by merging the changes into one file.
    - By pulling (accessing the file) and pushing (updating the file), both you and your collaborators have access to the most up-to-date version.

- Git also allows programmers to test changes to their code without changing the original.
    - If you don’t like the changes you made, you can revert to an older version of your file.

Is GitHub the Only One of Its Kind?

There are other tools that do what GitHub does.

- Bitbucket
- Git Tower
- Git Lab

This is only a partial list.

Git Basics

Overview

- Creating repos
- Adding files
- Commiting files into the repo

Creating a New Repository

```
git init
```

Alternatively, you can create your own repo by creating your own folder and uploading it to GitHub.

Creating a New Repository

`git init`

Alternatively, you can create your own repo by creating your own folder and uploading it to GitHub.

In order to put a project under Git's control, you'll need to initialize the process inside the project's root folder, add the files to Git's staging area, then commit the files.

Making Changes in a Repo

1. ***`git add`***
2. ***`git commit`***
3. ***`git push`***

More on each step in the next slides!

Adding Files to the Repository

`git add <filename>`

Once you’re ready to commit work to the “staging area”, you’ll do so via the ***add*** flag.

This tells git to include the file.

Committing Files to the Repository

```
git commit -m <message>
```

With files added to the staging area, you’re now ready to commit.
Git commit tells git to save a version of the repo.

Committing Files to the Repository

```
git commit -m <message>
```

With files added to the staging area, you're now ready to commit. Git commit tells git to save a version of the repo.

Git commit allows you to include a brief message about what you've changed. For example:

```
git commit -m 'Change how keyboard emphasis is styled
```

Remember: "keep your changes small, and commit often."

Let’s play with Git!

https://github.com/PennSpark/git-workshop/blob/main/workshop.md#starting-your-journey

Git Workflows

Overview

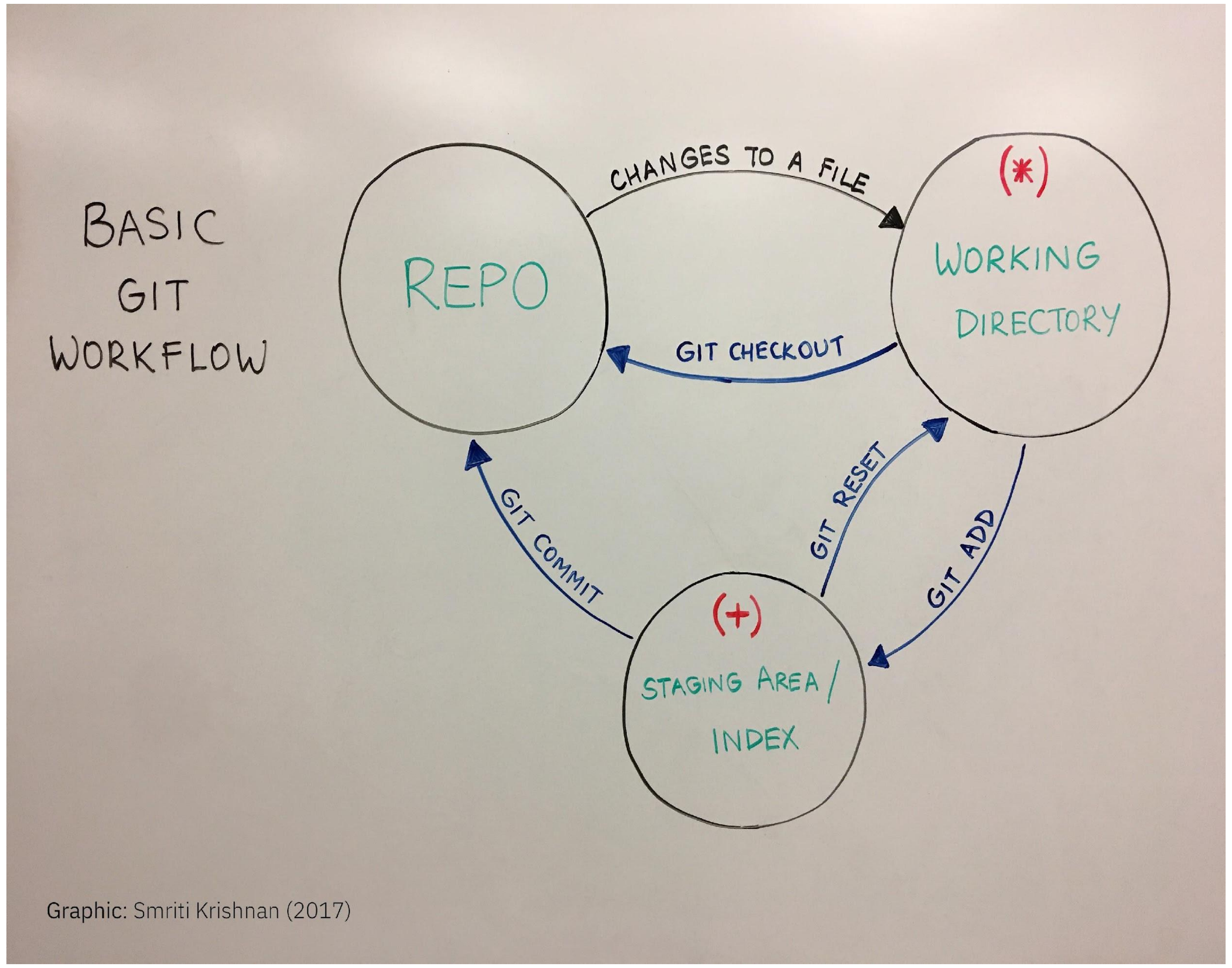
- VCS States
- Status
- Checking differences

The 3 States of a File in Git’s View

Files in a Git-controlled repository can only be in one of three states: tracked, ignored, or untracked.

- **Tracked** means Git will track changes and stages associated with a file.
- **Ignored** means Git will explicitly ignore modifications, including deletions, of a file.
- **Untracked** means Git is unaware of a file present in a repository.

The Pathways of File Changes



Graphic: Smriti Krishnan (2017)

Repo (Git Directory) → Working Directory → Staging Area/Index

As you work with files, their changes move between the working directory, the index, and the Git directory (repo).

- Modified files live in the working directory
- staged files (via the ***git add*** command) reside in the index
- committed files (via the ***git commit*** command) go in the Git directory.

Checking The Repository’s Status

```
git status
```

To see differences sitting in the working directory, changes set in the staging area, untracked files, deleted files, etc, run this command

Checking Differences

```
git diff
```

The ***git diff*** command shows the differences between the last committed change(s) and the current one(s) in the working directory.

Checking Differences

```
git diff
```

The ***git diff*** command shows the differences between the last committed change(s) and the current one(s) in the working directory.

The flag ***--cached*** shows changes placed in the staging area that have not yet been committed. If you decide that you want to revert those changes back into the working directory, you can run ***git reset***.

Checking Differences

```
git log
```

To display the entire history of a repository, run ***git log***. Append the ***--reverse*** flag to the command to show the history in reverse.

```
git log -p <filename>
```

To display the entire history of a file, run ***git log -p <filename>***.

Discarding Changes; Resetting a File

Use ***git checkout <filename>*** to discard all changes to a file and revert back to the state of the file at the last commit.

Let’s do some more with Git!

https://github.com/PennSpark/git-workshop/blob/main/workshop.md#lets-see-what-just-happened

Public Repos

Overview

- Cloning
- Pushing/Pulling
- Forking & Pull Requests

Cloning a Repository

`git clone`

Cloning a repository replicates the entire project and Git repository.

For example, ***git clone git@github.com:facebook/react.git*** will create a folder called ***react*** in the directory from where you invoked the ***clone*** command and replicate the entire project and Git history.

Pushing: Adding your own updates

When changes have been placed in the staging area, then committed to the Git directory, they are ready to be pushed (assuming you have a remote repository). The process is simple:

```
git push
```

Your changes will be fetched from your local machine and merged into the remote repository.

Pulling: Accessing collaborators’ updates

If you’re working with someone else, or you simply use multiple machines, you’ll need to pull changes made by others (or by you on a different machine) before you can push your changes. Again, the process is simple:

```
git pull
```

Your local machine will perform Git’s ***fetch*** feature, followed by its ***merge*** feature.

Pull Requests

A pull request means that you as the owner of a forked branch of a project are requesting that the owner of the original branch pull your changes into her/his repository. You may or may not have write access to the repository into which you want your changes pulled.

Removing Files

Removing tracked files can be done in one of two ways.

The first method excludes Git altogether; you simply delete files as you normally would. You still add the deleted files to the staging area as you would any other file. (Yes, you’re even required to stage deleted files.)

Removing Files

The first method excludes Git altogether; you simply delete files as you normally would. You still add the deleted files to the staging area as you would any other file. (Yes, you're even required to stage deleted files.)

The second method requires the ***git rm*** command. When files are deleted using ***git rm***, deleted files are automatically staged, saving you the trouble of carrying out the ***git add*** command. This is the only difference between both file removal methods.

Forking

Forking means that the forked branch is going in another direction. This could be to expand on work already done, or to issue a pull request at a later time.

Consider Linux: At some point, the folks at Amazon forked the Linux project to create the Kindle operating system. The Linux kernel has also been forked to create different variants of Linux: Debian, Fedora, etc.

Branching

Overview

- Definition
- Useful Commands

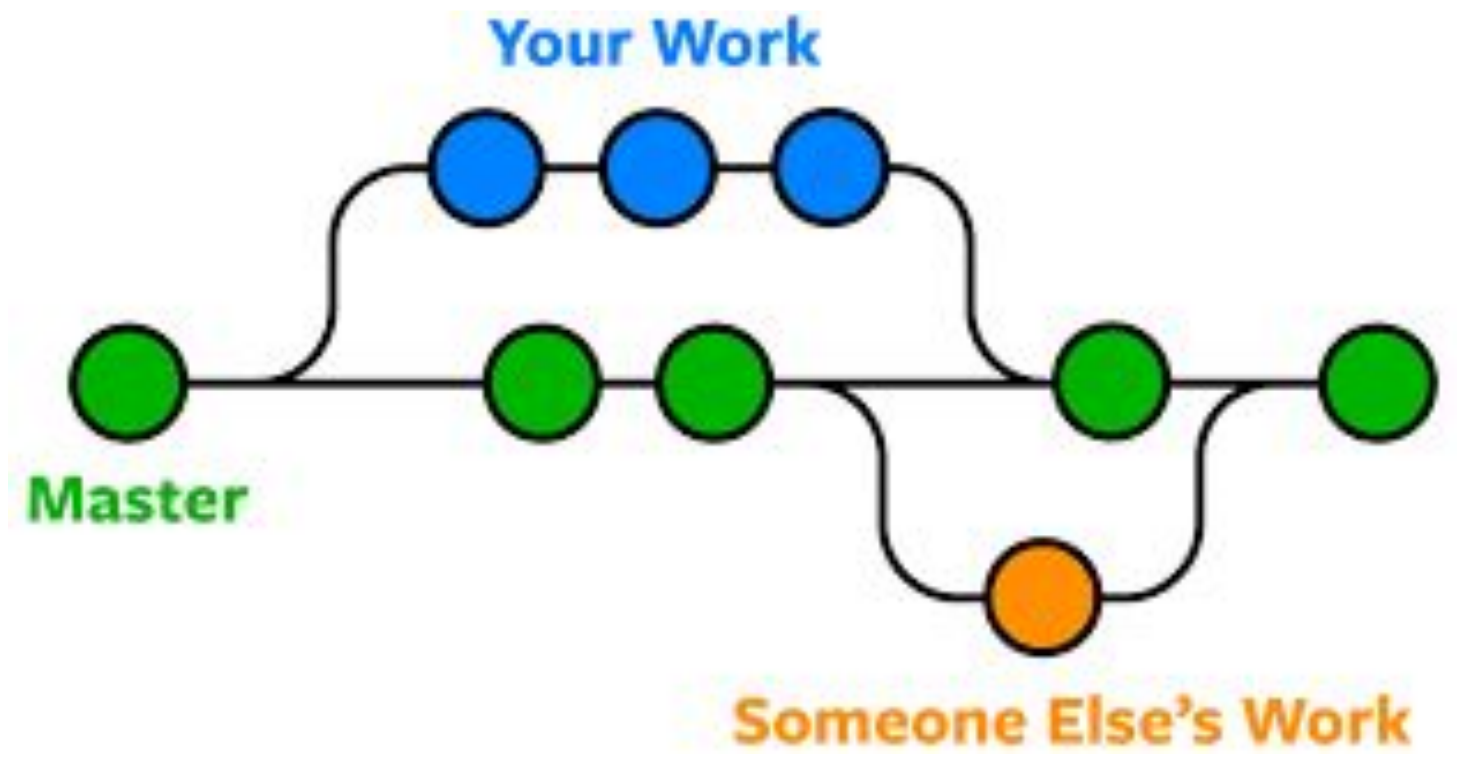
Branching

Branching is the ability to take your project in another direction from a certain point. This allows you to create new work based on an existing snapshot of a project without affecting the project.

The primary branch Git creates is called ***master***. This is merely a convention; you can choose to rename the branch anything else.

Branching

A common branching procedure is to create a ***dev*** branch in which to do all development, then merge ***dev*** into ***master*** at certain points in the development of a project.



Branching

git branch
shows all branches of code.

git branch <branch_name>
makes a new branch.

git checkout <branch_name>
switches to working on a different branch
– we call this "checking out" a different branch
s

Branching

The following command will look at all of the remote branches:

```
git branch -r
```

The following command will look at all of the local branches (to your machine).

```
git branch -l
```

Branching

Branches can be made from other branches. Let’s create a feature branch from ***dev*** called ***new-navigation***. First, we’ll checkout out the ***dev*** branch:

```
git checkout dev
```

And now we create our new branch:

```
git checkout -b new-navigation
```

Let’s do some more with Git!

https://github.com/PennSpark/git-workshop/blob/main/workshop.md#branching

Merging

Overview

- Definition
- Useful Commands
- Rebasing

Merging

git merge

Branches are typically merged into other branches, although they don't have to be.Issuing the ***git merge <branch>*** command merges ***<branch>*** into the current branch. For example, say you want to merge ***dev*** into ***master***.

First, checkout ***master***:

```
git checkout master
```

Then merge ***dev***:

```
git merge dev
```

Merge Conflicts

When collaborators make different changes to the same lines of code, a “merge conflict” arises… and git is unable to automatically solve it.

Git will raise a conflict flag and show you all the versions of the line(s) with conflicting changes.

Merging

```
git merge --no-ff
```

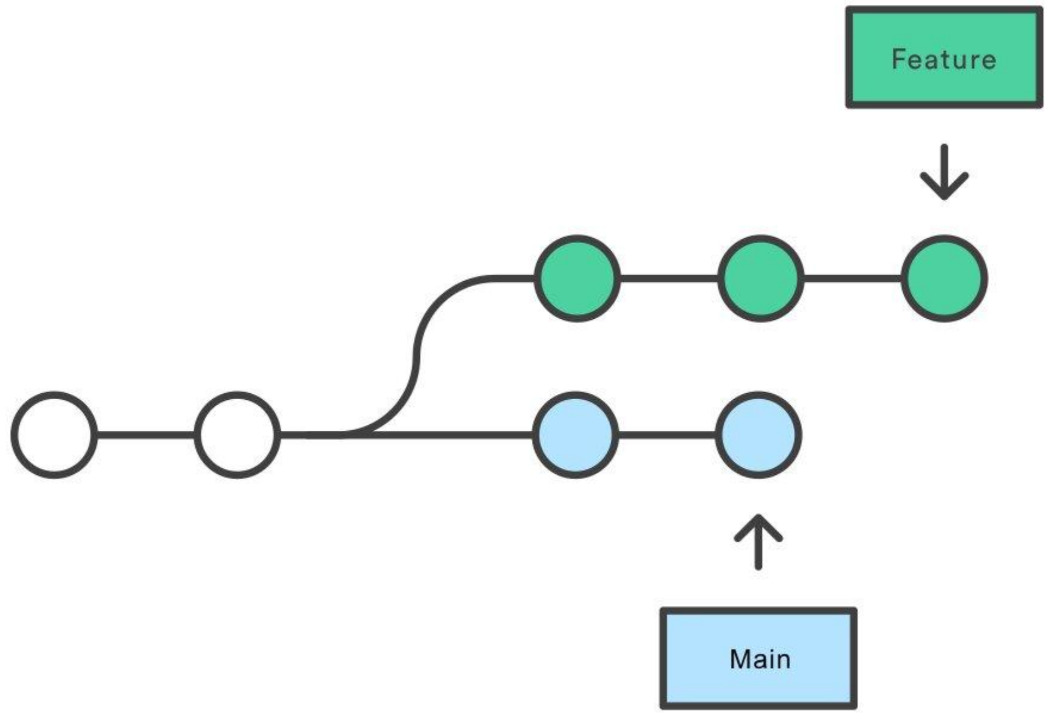
You would have noticed that The Terminal did not ask for any input from you. Most times, however, you want to assign a message to the merge. Appending the ***--no-ff*** (for no fast forward) to the merge command invokes your text editor so you can associate a message with the merge. Thus, the previous command could be modified as such:

```
git merge --no-ff dev
```

Note on Rebasing

Consider what happens when you start working on a new feature in a dedicated branch, then another team member updates the main branch with new commits that are relevant to your feature. This results in a forked history that looks like the following:

A forked commit history



Rebasing (cont.)

The easiest way to resolve this would be to merge the main branch into the feature branch by doing the following:

```
git checkout feature
git merge main
```

This creates a new “merge commit” in the feature branch that ties the history of both branches. This is nice, because it is *non-destructive* but can pollute your feature branch history.

The Rebase Option

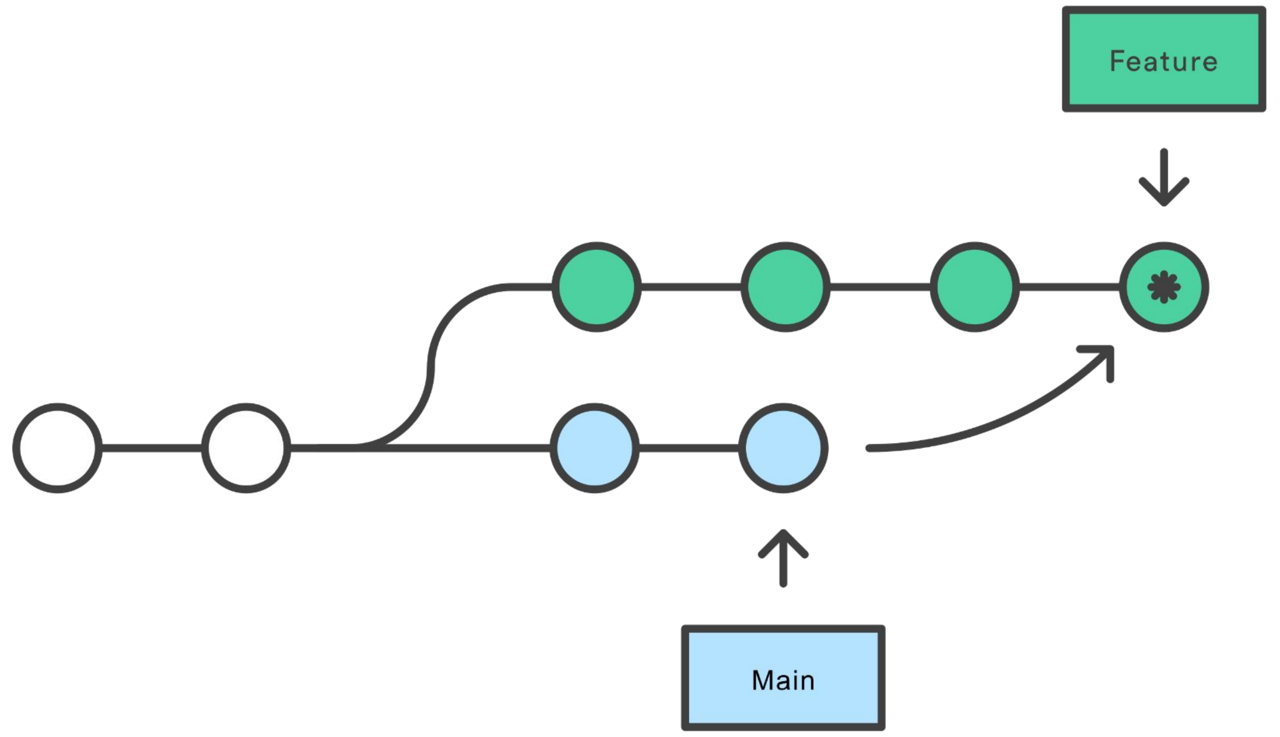
As an alternative to merging, you can rebase the feature branch onto the main branch using the following commands:

```
git checkout feature
git rebase main
```

This moves the *entire* feature branch to begin on top of the tip of the main branch. Instead of using a merge commit, rebase *rewrites* the project history by creating brand new commits for each commit in the original branch

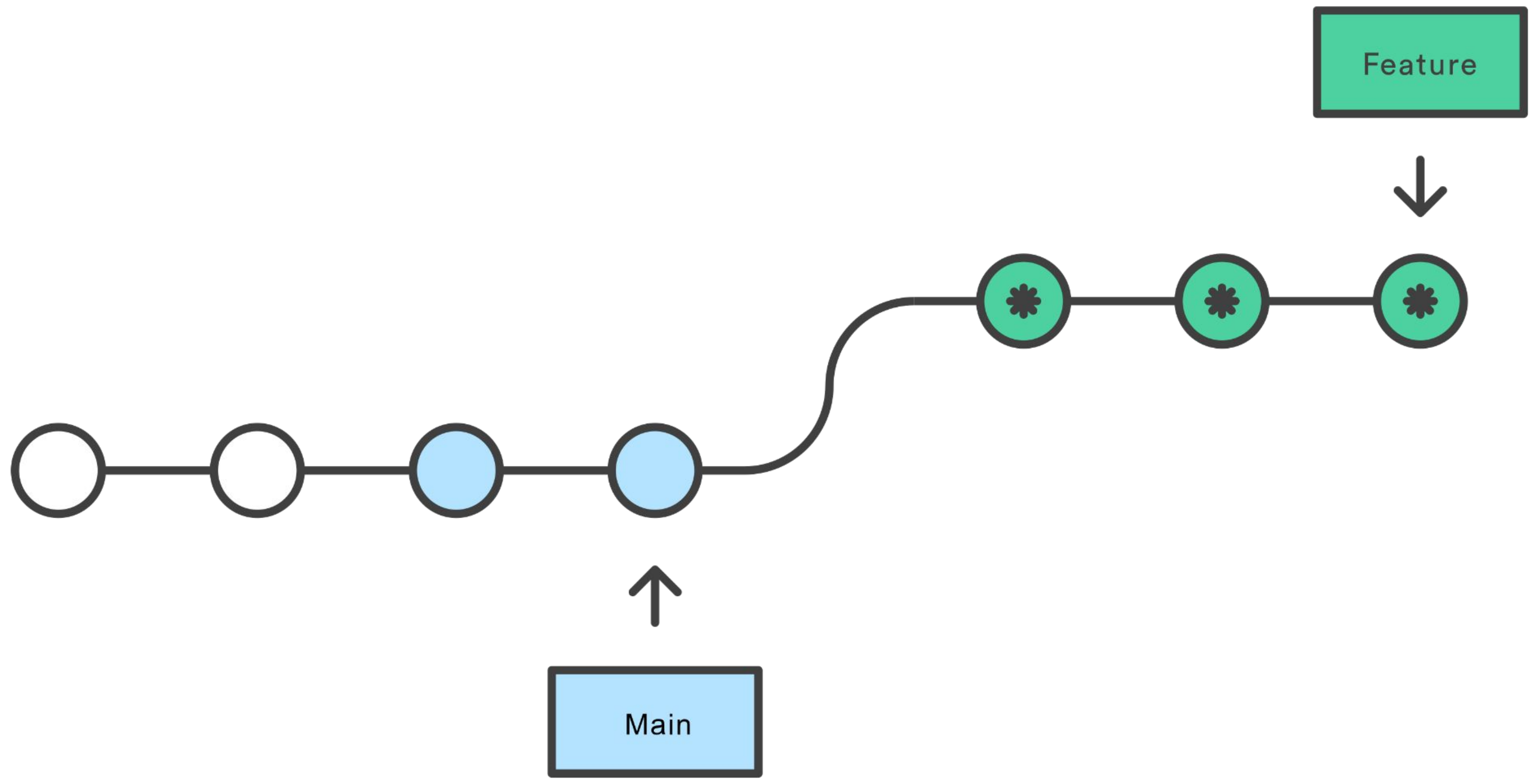
Visual Comparison

Merging main into the feature branch



✱ Merge Commit

Rebasing the feature branch onto main



✱ Brand New Commit

Let’s do some more with Git!

https://github.com/PennSpark/git-workshop/blob/main/workshop.md#merging

Resources

Overview

- Cheat Sheets
- Deliverable

Cheat Sheets

There are myriad of cheat sheets online. Here are just a few:

- Atlassian PDF
- GitHub PDF
- Git Tower HTML and other formats and languages
- Andrew Peterson of NDP Software A fun, interactive cheat sheet

Suggested Readings

Git for Humans by David Demaree.
Published by A Book Apart, which is known for publishing easy-to-understand short books that are well-designed.

Pro Git by Scott Chacon and Ben Straub.
Free in eBook formats, this is a good reference and very detailed.

Git Pocket Guide by Richard Silverman.
Available for **free** online, this book is a good, strong reference, and, in paperback, an easy book to carry.

Version Control with Git by Jon Loeliger and Matthew McCullough. Every detail you'd want to know about Git is contained in this tome.

Week 1 Deliverable: About Me Card

TASK

Your task in this assignment, should you choose to accept it, is to write a simple README.md file that displays information about you on your GitHub. This can include your major, hobbies, and other interests. The instructions are here.

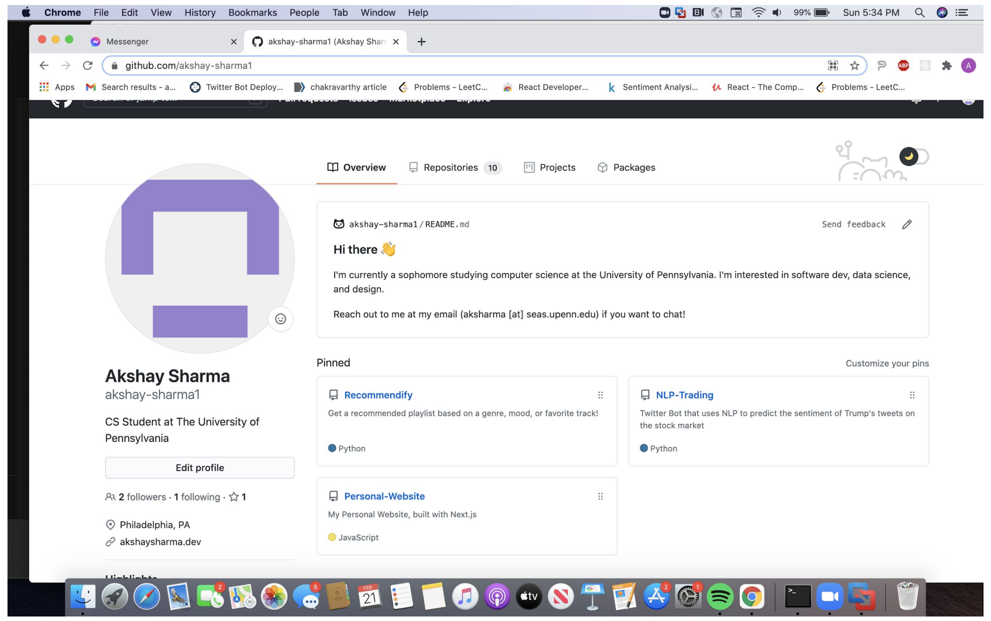
Requirements:

- README.md must show up on your GitHub Profile (GitHub setup guide here)
- Must use local code editor & local repository to edit README file (instead of just using GitHub GUI)
- Be creative! Feel free to include whatever you want to!

DUE

2/13

Example



akshay-sharma1 / README.md

Send feedback

Hi there 👋

I'm currently a sophomore studying computer science at the University of Pennsylvania. I'm interested in software dev, data science, and design.

Reach out to me at my email (aksharma [at] seas.upenn.edu) if you want to chat!

Akshay Sharma
akshay-sharma1

CS Student at The University of Pennsylvania

2 followers · 1 following · ☆ 1

Philadelphia, PA
akshaysharma.dev

Pinned

Recommendify
Get a recommended playlist based on a genre, mood, or favorite track!
Python

NLP-Trading
Twitter Bot that uses NLP to predict the sentiment of Trump's tweets on the stock market
Python

Personal-Website
My Personal Website, built with Next.js
JavaScript

Submit Here