STORM HACKS

GIT / GITHUB WORKSHOP

Matthew Wong

INSERT PHOTO

WORKSHOP HOST

Matthew Wong
SFU Surge Logistics Coordinator
Year / Major

BEFORE YOU GET STARTED...

1 INSTALL GIT

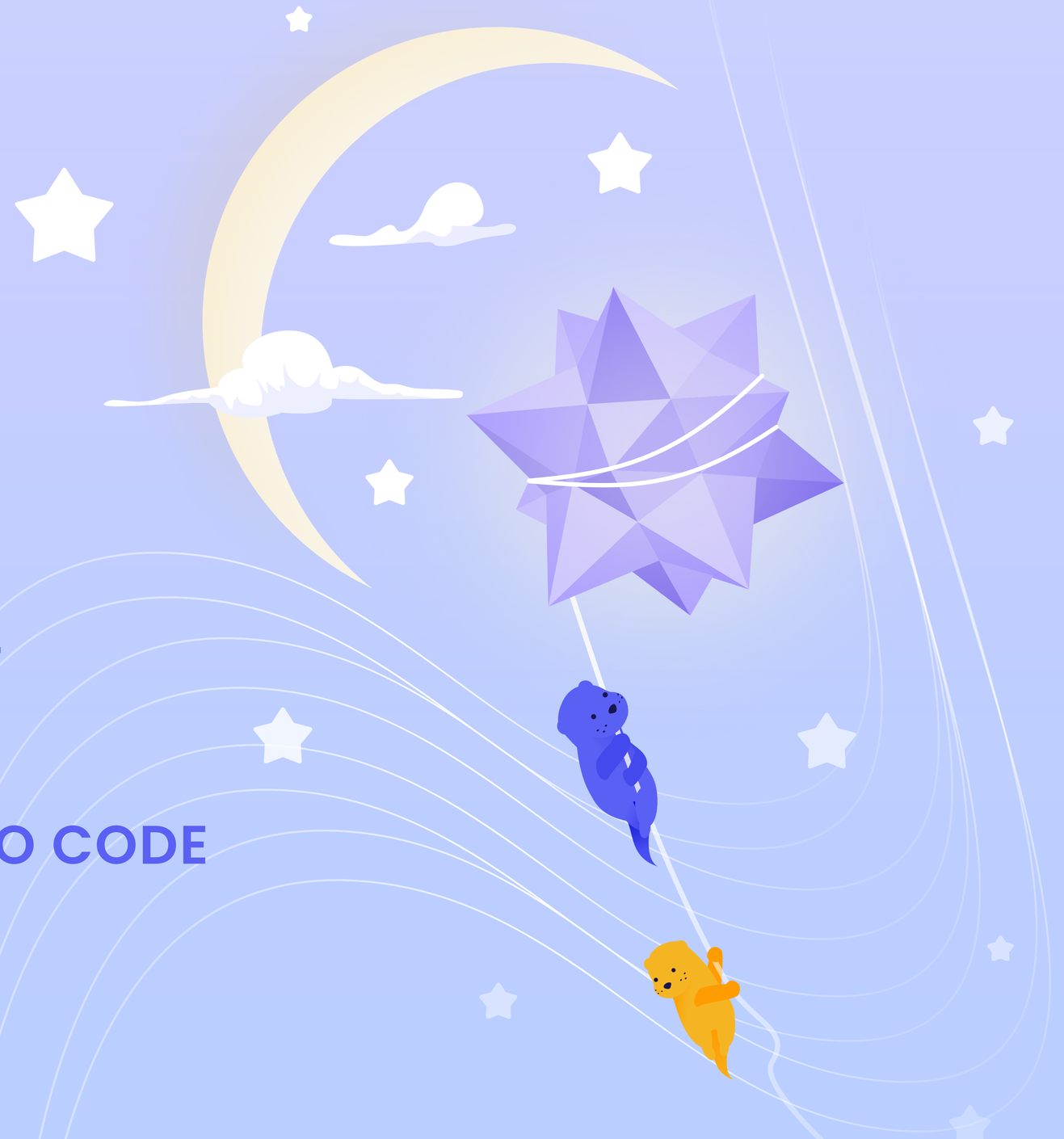
https://github.com/git-guides/install-git

2 MAKE A GITHUB ACCOUNT

https://github.com/join

3 DOWNLOAD VISUAL STUDIO CODE

https://code.visualstudio.com/download



FULL SCHEDULE

PART ONE

- What is Git?
- Why use Git?
- Git vs Github
- Branches
- Merging
- Pull Requests

PART TWO

- Rebasing
- Merge Conflict Resolution
- Merging vs Rebasing
- Reverting

FULL SCHEDULE

PART ONE

- What is Git?
- Why use Git?
- Git vs Github
- Branches
- Merging
- Pull Requests

PART TWO

- Rebasing
- Merge Conflict Resolution
- Merging vs Rebasing
- Reverting

WHAT IS GIT?

VERSION CONTROL: BASICS

- Keeps track of a history of changes to software code
- Allows for collaborative development
- Lets us go back and revert to older versions of our work

SO, GIT IS...

- A type of version control
- A distributed version control system, which means you can work offline!
- (A centralized version control system needs to be online)

GIT AND GITHUB DIFFERENCE

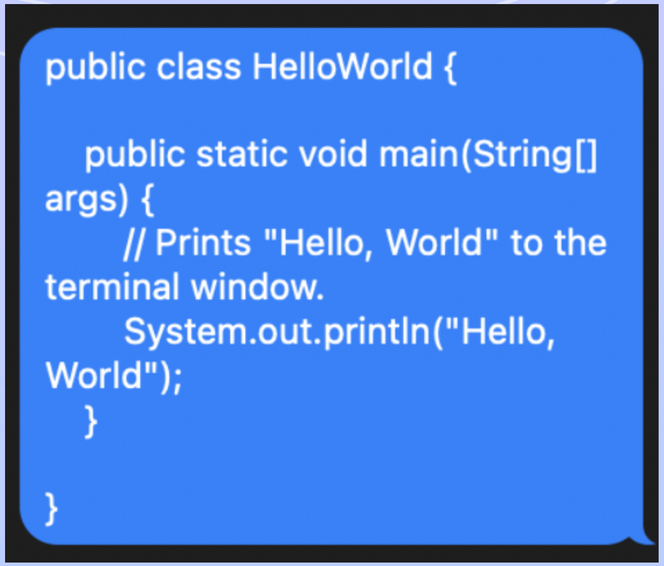
- Git = version control; keeps track of source code + history
- GitHub = cloud-based hosting service; helps you manage Git repositories

WHAT IS A REPOSITORY?

- Where Git stores the metadata and object database for the project
  - Aka all the files related to your code!
- Each developer has a copy of the code in a local repo on their computer and (most likely) a remote repo on a server
  - Local repo: make changes to your own copy of the code

HOW IS GIT USED?

HOW DO YOU COLLABORATE? (W/O GIT?)



hello world

Recipients

hello world

```
public class HelloWorld {

    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }

}
```

Send

MORE ABOUT GIT...

THE 3 STAGES OF A GIT FILE

- **Modified:** changed but not stored in repo yet - READY
- **Staged:** marked that this goes into the next commit to be stored into repo - GET SET
- **Committed:** stored into repo - GO
    - Sort of like a snapshot: when you commit, Git saves a “picture” of what your files look like
        - Each commit represents the changes made to your project in the past, along with details about when it was modified + who made the changes, along with an included message.

LET’S USE GIT!

STEP 1: INSTALLATION

- Instructions: https://github.com/git-guides/install-git
- Check if installed correctly:

```
git --version
```

STEP 2: CONFIGURE

- Making sure your contributions can be identified (+ give you those beautiful commit squares on your GitHub profile!)

```
git config --global user.name "<your name>"
git config --global user.email <email>
```

```
git config --list
```

LET’S CREATE OUR REPO!

STEP 3: CREATE AND CLONE

CREATE

- Name the repo after your username
- Add a README file, Open up the dropdown + copy the repo link
- OR: Create a new repo + clone into Documents folder!
  - cd <folder>

CLONE

- Creates a folder that you can work on locally (offline)

```
git clone <repo link>
```

STEP 4: EDIT!

- Navigate to the folder for the repo on your computer and open up the README.md file with Visual Studio Code
- I'll give you all ~ 5 minutes to write what you'd like to share with the world!
- Please react in Zoom to let me know if the workshop speed is okay so far!
- When you're done, save your file (ctrl-s/cmd-s)
    - in terms of Git stages, your file is now modified!!!

STEP 5: SEE + STAGE CHANGES

```
git status
```

- View differences between your working directory and your git local repo
- This adds all the files in your folder to staging

```
git add .
```

- The code snippets doc has other adding options (single file, multiple specific files) in the toggle if you're interested!

STEP 6: LET'S COMMIT!

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

- Makes a new commit to the local repo with the given commit message

STEP 7: LET'S PUSH!

```
git push
```

- Pushes your changes up to the remote repository
- It's possible for you to wait and push several commits at once, if that's more your style

STEP 8: LET'S REVIEW

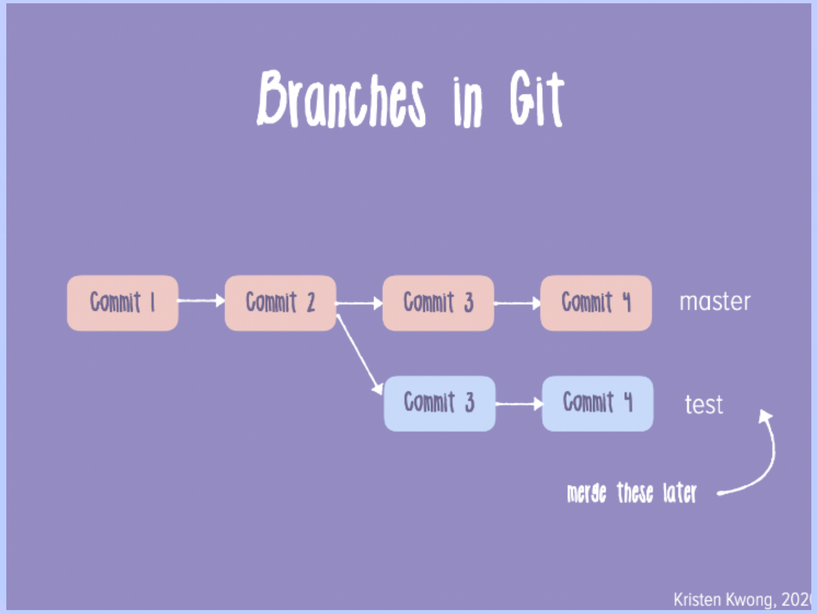
```
git log
```

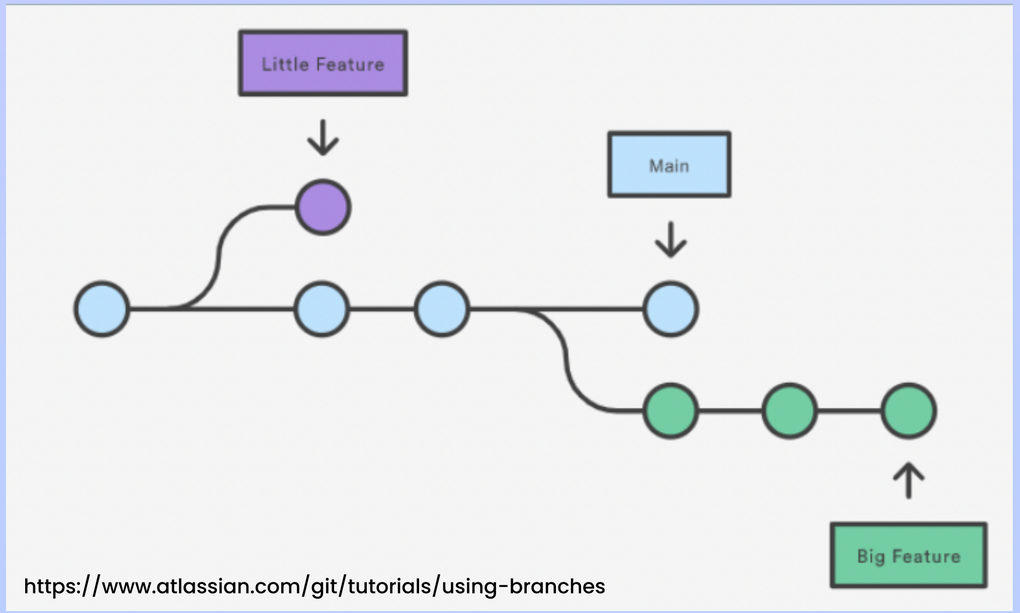
- Pushes your changes up to the remote repository
- It's possible for you to wait and push several commits at once, if that's more your style

BRANCHES

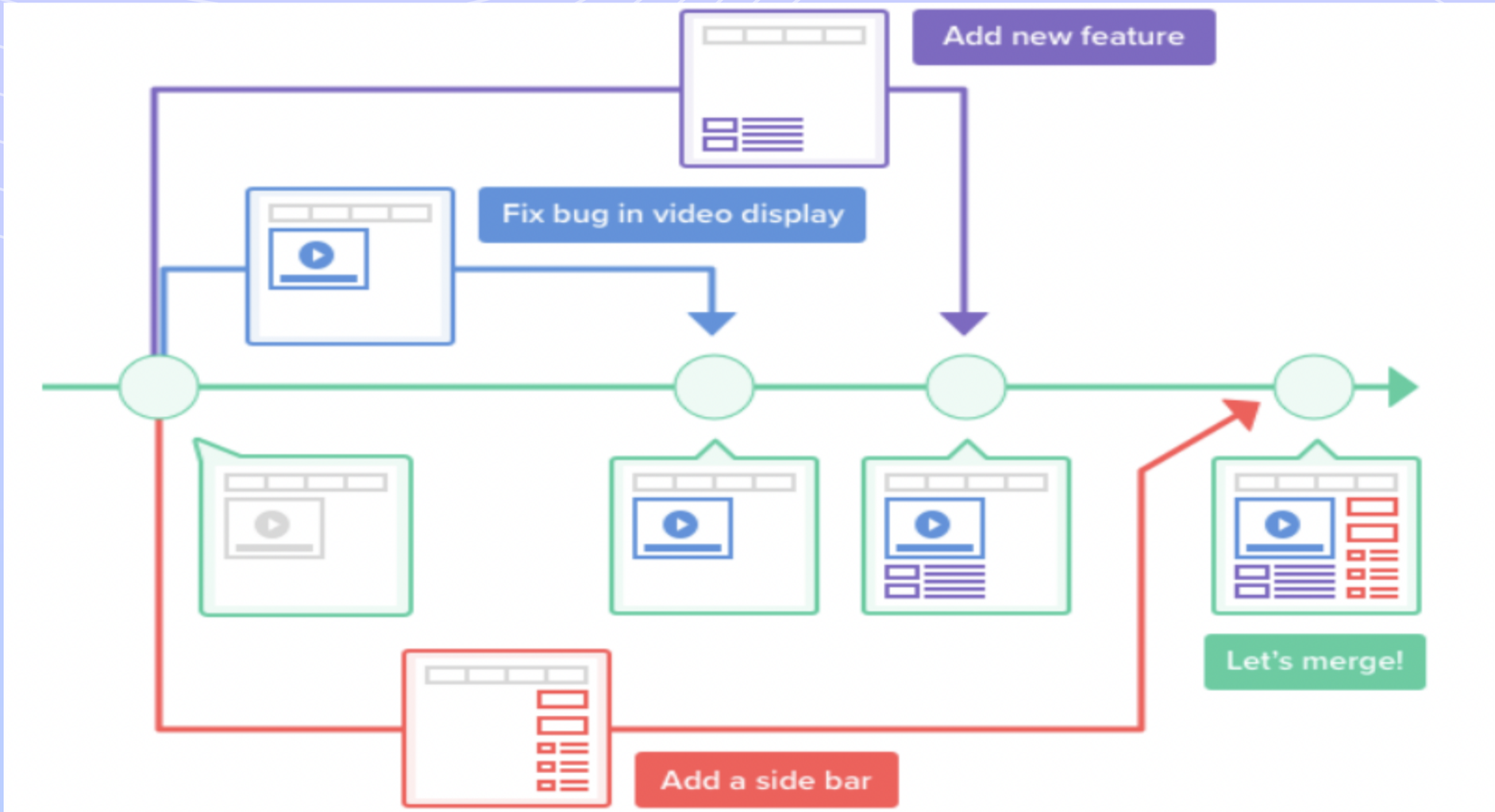
WHAT ARE BRANCHES?

- Branching is when you diverge from the main line of development and work directly on your own independent line





https://www.atlassian.com/git/tutorials/using-branches



https://backlog.com/git-tutorial/using-branches/

PROS...

- Really helpful if you're focusing on specific pieces of the project and collaborating with others!
    - Makes sure you don't have to do too much work to get old code back (because you're isolating your work from others!)
        - Changes in other branches will not affect your work unless you pull them.
    - Helps you to solve any conflicts between you and your collaborators' code + make the best choices for the project!

STEP 9: LET'S MAKE SOME!

```
git branch <name>
```

- You can use the below command to make a new branch
- If you want to make a new branch and go directly to it, you can use the command in the toggle in the code snippet doc

STEP 10: SOME COOL THINGS YOU CAN DO

- List all branches:

```
git branch
```

- Go to a specific branch:

```
git checkout <name>
```

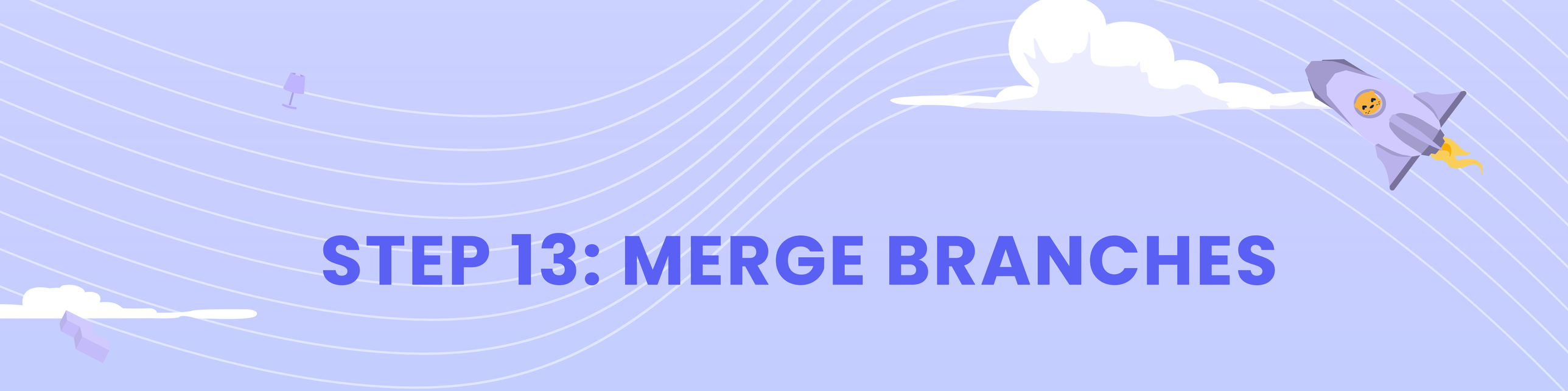
STEP 11: ADD SOME SOME STUFF

- I’ll give you a minute to make a change to your README file
- If you can’t think of anything, just add a punctuation mark somewhere

STEP 12: STAGE + COMMIT YOUR CHANGES

- Fun trick that can make things easier – you can actually do both at once!

```
git commit -am "<commit message>"
```



STEP 13: MERGE BRANCHES

```
git merge main
```

- Merge the new code into main (merging the main branch into the outside branch)!
    - Already up to date

STEP 14: PUSH IT

- Our branch has only been created on our local machines, so we'll need to use a special command to push our branch up to the remote repo

```
git push -u origin <branch>
```

PULL REQUESTS

PULL REQUESTS...

- You might want others to review your code before you push to master
- Why?
  - You might not want to share code that’s incomplete
  - It’s also not that great to push to master – it might break other people’s code
- After pushing your new branch to the remote repo, you can create a pull request to merge it with main

STEP 15: LET'S PULL IT!

- Compare + pull request
- Create pull request
- Merge pull request

FINALE!

- Make sure you get these remote updates on your local machine
- Pulling is super important if you’re working with other people, because they’ll be updating it too!

```
git pull origin main
```

Shortcuts & Basic Team Workflow

1. Make a repo right on GitHub. Share link with team.
2. Team members will use `git clone <link>`
3. Someone works on stuff. They push with `git push -u origin <branch>` to the remote.
4. They will make and merge the pull request.
5. Everyone else uses `git pull` to get the changes.
6. Repeat Steps 3 - 5 until project is done ✨

Kristen Kwong, 2020

5 MINUTE BREAK!

FULL SCHEDULE

PART ONE

- What is Git?
- Why use Git?
- Git vs Github
- Branches
- Merging
- Pull Requests

PART TWO

- Rebasing
- Merge Conflict Resolution
- Merging vs Rebasing
- Reverting

REBASING

```
git rebase <branch>
```

- An alternative to merging
- Used when you want to introduce changes from the main branch into your feature branch

REWRITING HISTORY

```
git rebase -i <branch>
```

- Can rewrite your commit history
- Only use for local commit history and should never be used to alter the main branch’s commit history
- This will open a built-in editor called the Vim editor where you can edit your commit history

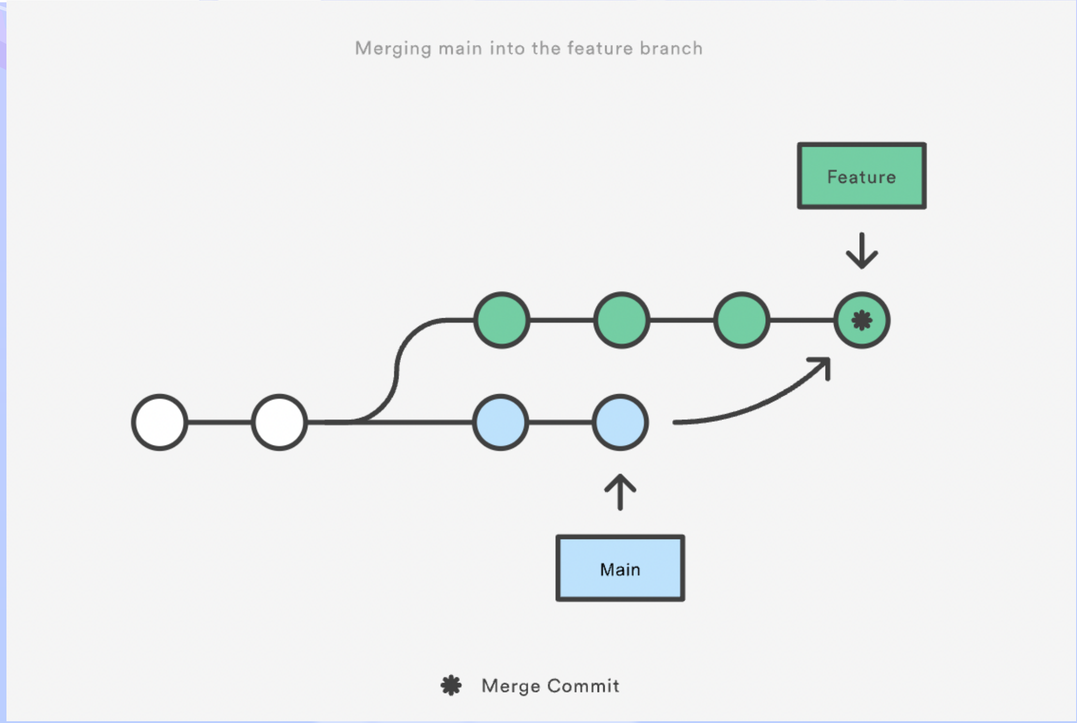
MERGE CONFLICTS

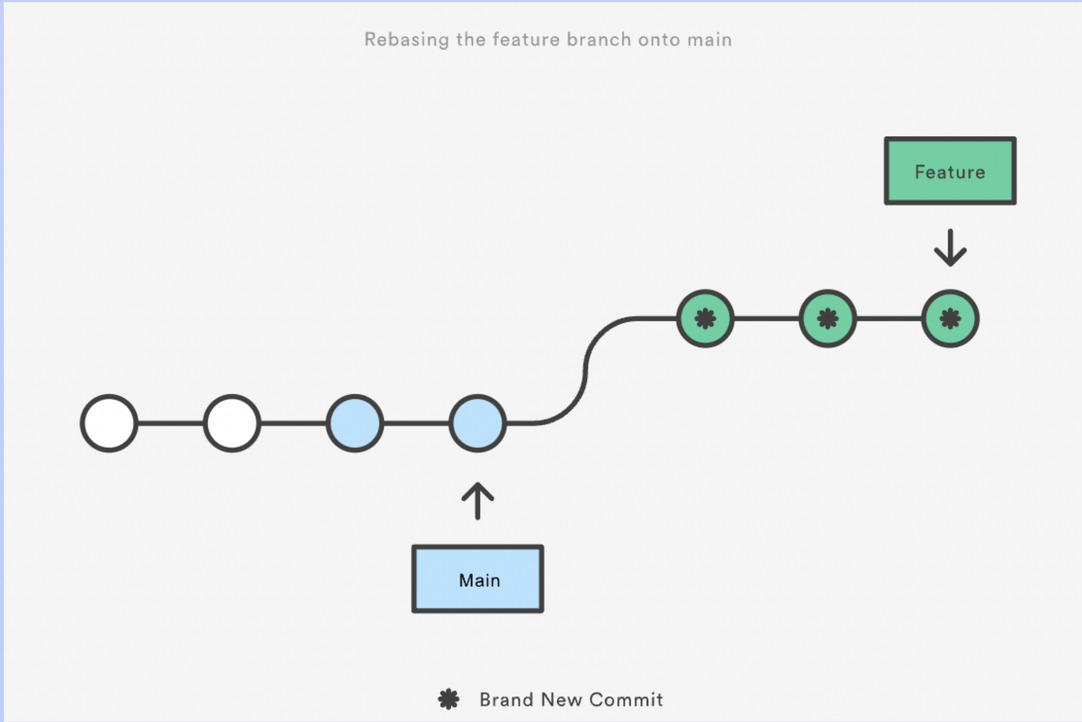
```
git merge <branch>
git rebase <branch>
```

- Occur when changes are made to the same file in 2 different branches
- Solved by choosing what changes are kept

MERGING VS REBASING

White = old main branch
Blue = new changes to main branch
Green = new changes to feature branch





https://www.atlassian.com/git/tutorials/merging-vs-rebasing

MERGING VS REBASING

MERGING PROS

- Overall easier to understand
- Simpler to merge branches and solve conflicts

MERGING CONS

- Extra commit dedicated to merge

REBASING PROS

- More control over commit history
- Cleaner & linear commit history
- No extra commit

REBASING CONS

- Overall more difficult to understand
- Merging branches and solving conflicts can get complex

REVERTING

- Undoing a commit with another commit
- Produces a list of all the previous commits on the repo
- Commit hash
    - A commit’s “name” represented by a string of hexadecimal values
- Reverts the commit with the commit hash

```
git revert <commit hash>
```

QUESTIONS!

THANK YOU FOR YOUR TIME!

We look forward to seeing you at SFU Surge's StormHacks 2023!

If you have any questions, comments, or concerns for us, please feel free to contact us at info@stormhacks.com.