



Git

Version Control Presented by UCLA ACM Hack

Speaker: Nathan Zhang
Slides: Katelyn Yu



Link to the Slides:

<https://tinyurl.com/3f2wrh4u>

Link to the README:

<https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



What is git?

Open-source, distributed version control software



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>

What is git?

Let's talk about video games: specifically save points

- Allow you to save the your progress in you video game
- If you mess up at some point, you can go back to your save point and continue playing from there



What is git?

Save points are to **video games** as **git** is to **program development**

Git allows you to save snapshots of your folder to return to at a later time

- Can see what changes were made to the files between the save point and now
- Can revert the state of your program to that saved state



It's always "What is git?" not "Why is git?"

Imagine video game save points. Namely, the idea that you didn't make one.

And then you make a fatal mistake.



And now you gotta start ALL OVER AGAIN. FROM THE BEGINNING. RIP.

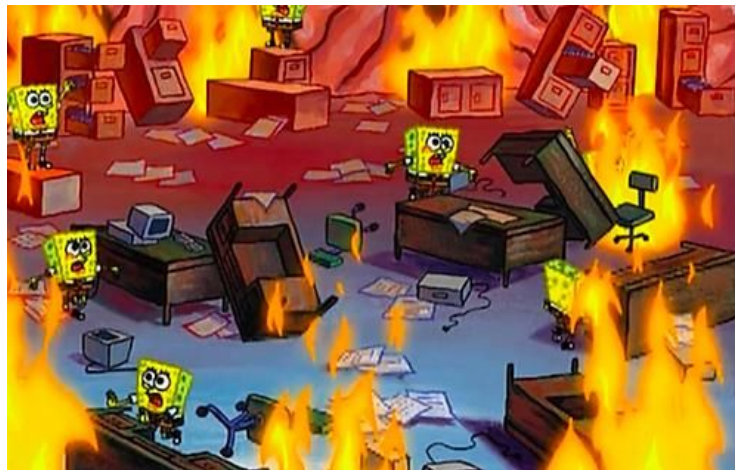
README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

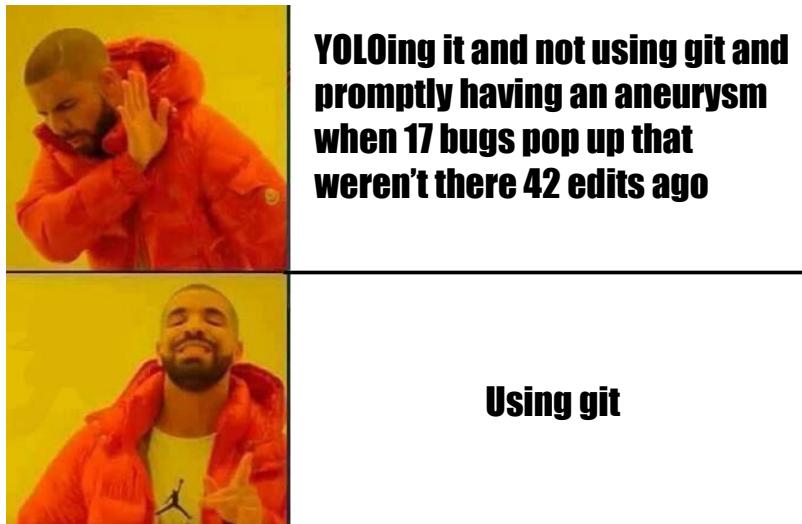
It's always "What is git?" not "Why is git?"

Same with your coding project!



It's always "What is git?" not "Why is git?"

Moral of the story: please use git



Installing git (Linux or Windows)

Git is a command line tool that is used through the terminal.

Linux:

```
$ sudo apt-get install git-all
```

Windows:

[Direct Installation](#)

Link to these slides are: <https://tinyurl.com/3f2wrh4u>

Note: Any phrase starting with a "\$" is meant to be run on terminal (or cmd for windows). Ignore the "\$" when actually using the command



Installing git (Mac)

Note: Any phrase starting with a "\$" is meant to be run on terminal (or cmd for windows). Ignore the "\$" when actually using the command

1) Install Homebrew Package Manager

```
$ /bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh  
)"
```

2) Install git using Homebrew

```
$ brew install git
```



Git configuration

We want people to know who's doing what changes in our project code. This will come in handy when you are working with other people in a git project.

```
$ git config --global user.name "John Doe"
```

- Logs your name

```
$ git config --global user.email johndoe@example.com
```

- Logs your email



Mini-Demo

- Opening terminal
- Installing git
- git configuration

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

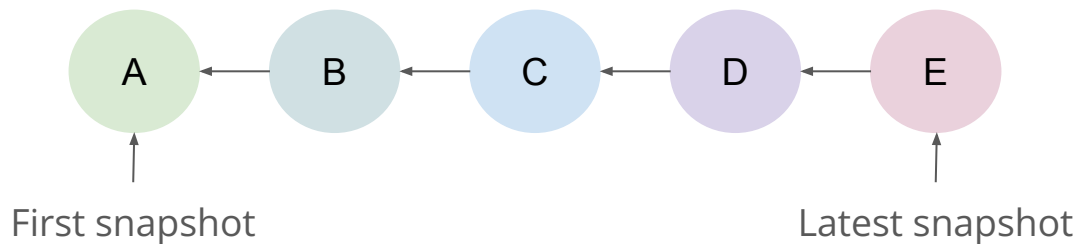
COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



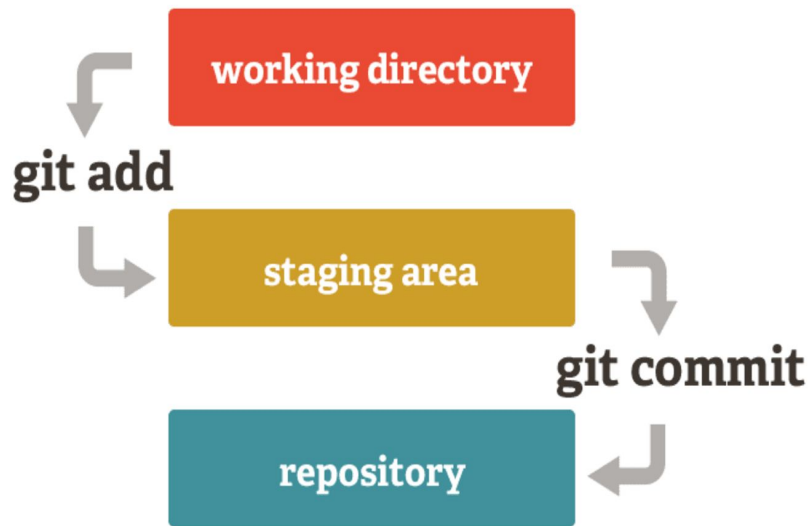
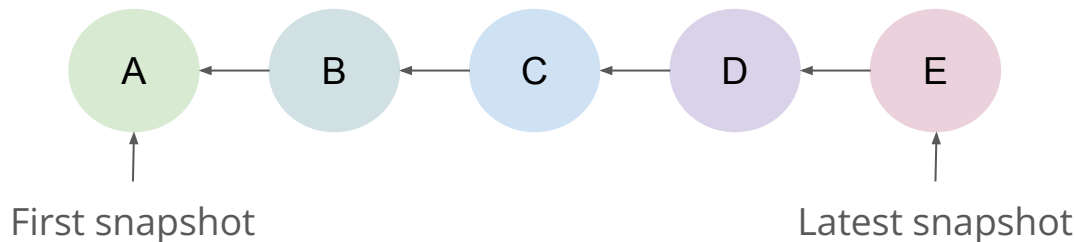
What does saving look like?

- Circle = snapshots of the state of your files when you saved
- Each circle points to the previous circle
 - Previous circle is the previous save point
 - We end up with a timeline for our program's evolution



Git Workflow

- Working directory: the files on your personal device
- Staging area: all the changes that you want to make a snapshot of
- Repository: collection of snapshots
 - ie this diagram from last slide



Git Workflow: Working Directory

Where's the working directory?

- On your personal computer!
 - It's the folder that has the project code you want git to take snapshots of

How does this have anything to do with git?

- You have to tell git to track changes to these files



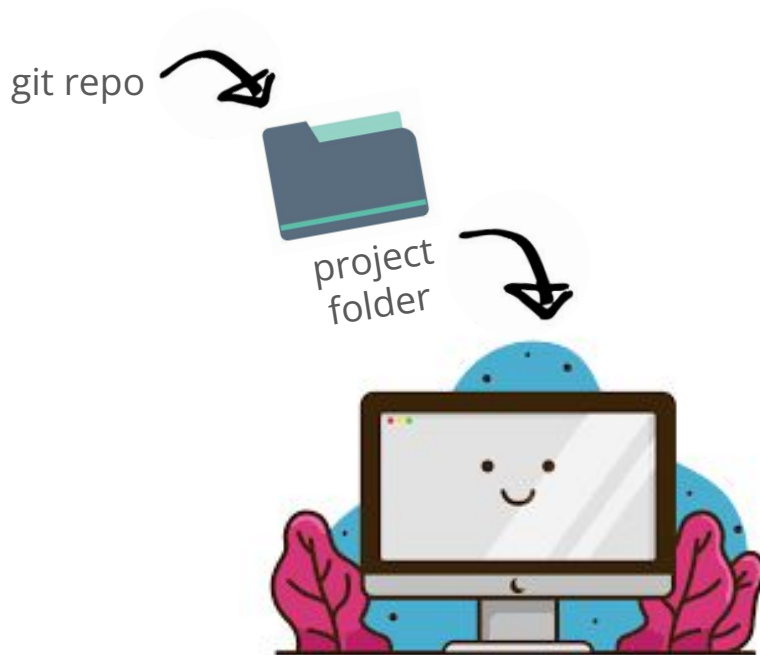
Git Workflow: Working Directory

How do we introduce git to the project files on our local device?

```
$ git init
```

- creates an empty git repository in the folder you currently are in

The folder you create the git repo in is your working directory



Git Workflow: Staging Area

What's the staging area?

- Holds all the changes you've made that git will make include in the next snapshot
 - Git will pay attention to and track the changes of the files that are placed here



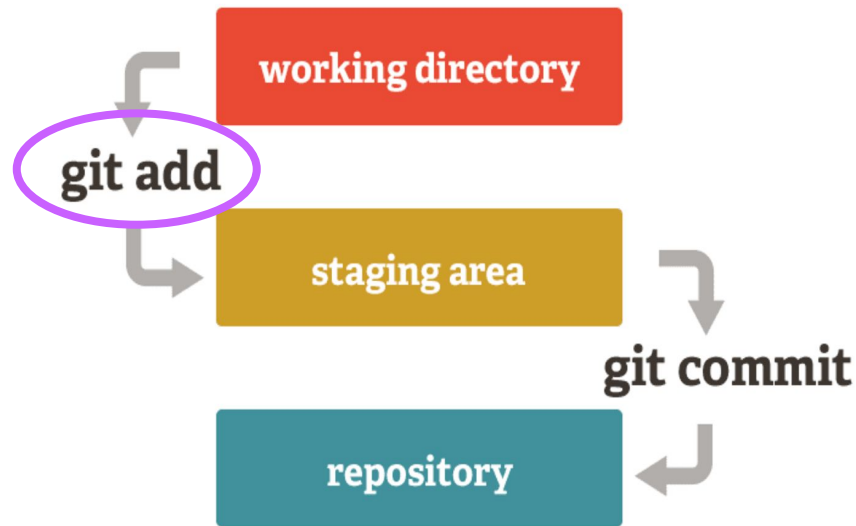
Git Workflow: Staging Area

Note: [square brackets] are placeholders
Eg. `git add file.c`

How do we get things into the staging area?

```
$ git add [files/folder]
```

- Adds any changes you have made to the working directory since your last commit to the staging area
- If you want to add everything to staging area: `$ git add .`



Git Workflow: Repository

What is the repository?

- It's the collection of all snapshots made in time order

What are these snapshots?

- Think of them as wrappers around everything that you put in the staging area
 - All the changes you made condensed into one bundle



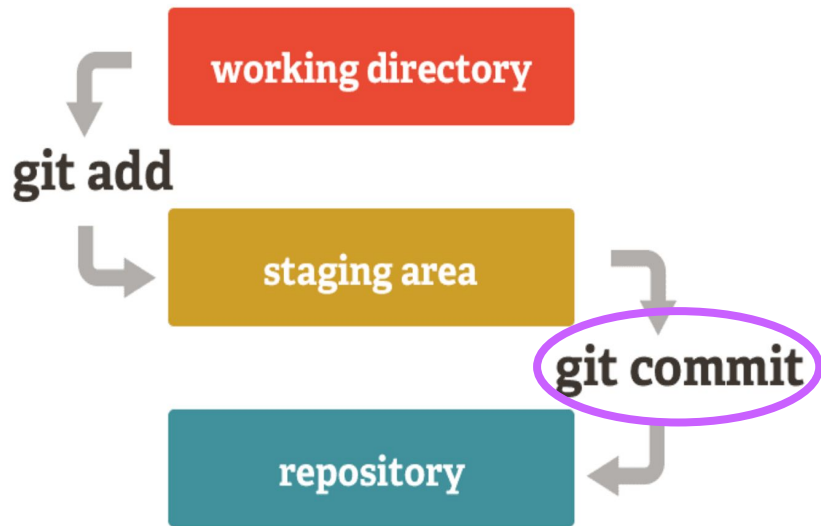
Git Workflow

How do we get our files from the staging area into the repository?

`$ git commit`

- Packages everything up in the staging area into a commit and puts the commit in the repo
 - Commit = snapshot
- Good practice to include a commit message: `$ git commit -m "[your commit message]"`

Note: [square brackets] are placeholders
Eg. `git commit -m "This is a nice and succinct commit message"`



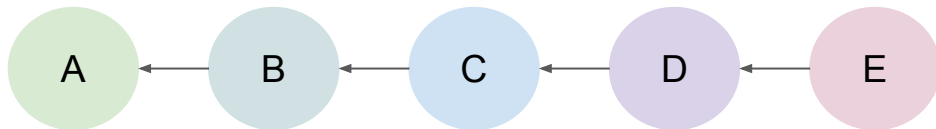
Extra Git Commands

`$ git status`

- Displays the changes you have made to your files since your last commit

`$ git log`

- See the list of commits the repo has
- Per commit, displays
 - Who committed
 - When they committed
 - Commit message



Extra Git Commands

```
$ git restore [file(s)/folder]
```

- Undoes changes in the working directory
 - Like going back to your last save point

```
$ git restore --staged [file(s)/folder]
```

- Removes staged files from the staging area



Demo Part the Second

- Create a project folder
- Create a git repo in that folder
- Create a commit



Daniel Stori {turnoff.us}

How do we collaborate on repositories?



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>

Wot is Github?



- ~~What people use to cheat on CS assignments~~
- A git-based version control system, stored in the cloud
 - You and your collaborators can easily upload and download code to this online repository
- Git \neq Github
 - Github is an online repository that enables collaboration
 - Git is the version control system on our own devices

What does using GitHub look like?

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

A visual representation



"master" version

Each computer and GitHub will have their own copies of the code



local copy



local copy

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



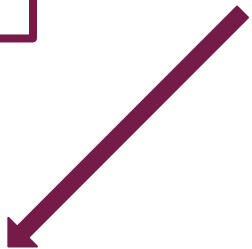
acm.hack

A visual representation

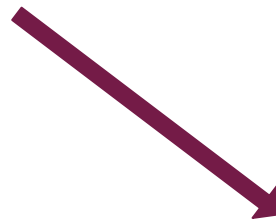
We can download
local copies of code
onto our own
computers



"master" version



local copy



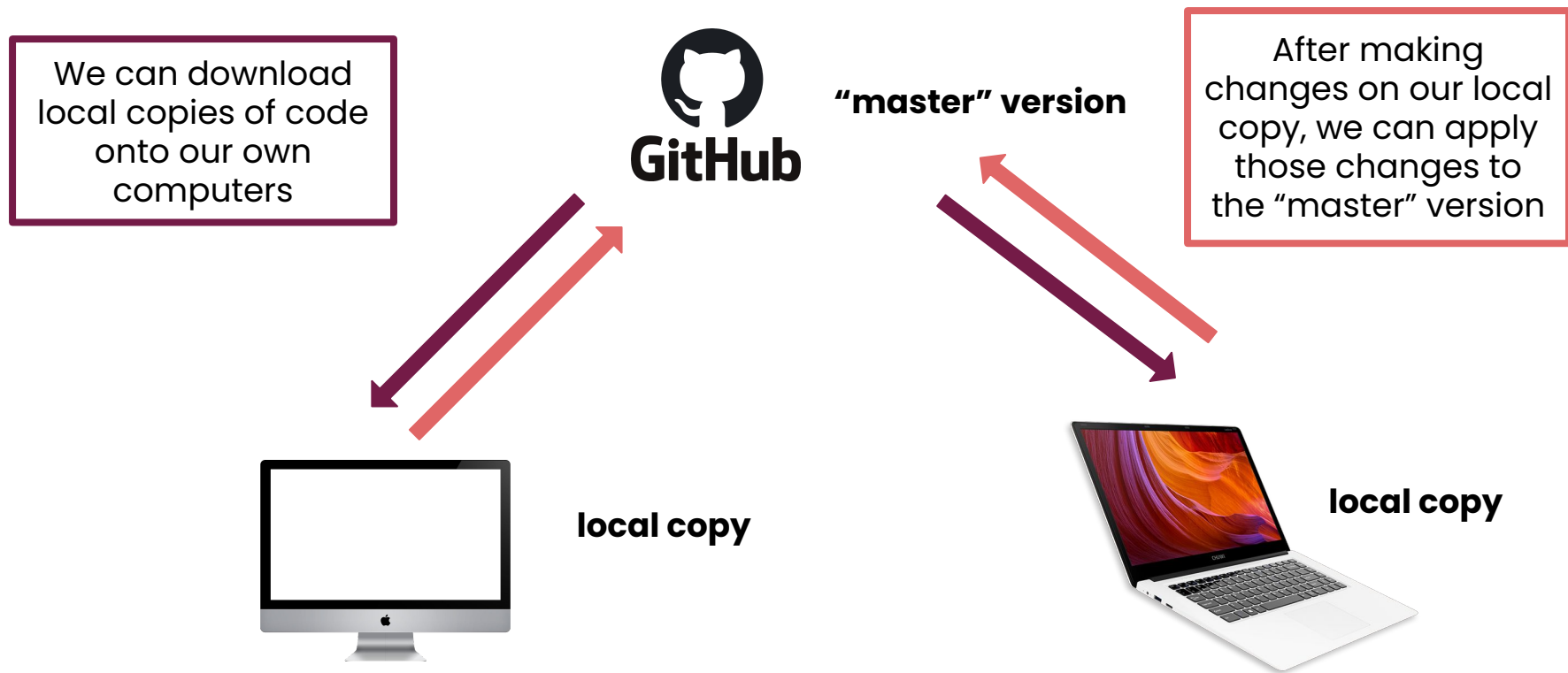
local copy

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

A visual representation



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

Why use separate copies?

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

Shared code



**your friend
(not so
smart)**

```
#include <stdio.h>

int main() {

    printf("hello world");

}
```



**you
(smart)**

Shared code



your friend

```
#include <stdio.h>

int main() {

    printf("hello world");
    printf("uwu");

}
```



you

Shared code



your friend

```
#include <stdio.h>

int main() {

    printf("hello world");
    printf("uwu");
    printf( oh no )

}
```



you

Shared code

```
#include <stdio.h>

int main() {

    printf("hello world");
    printf("uwu");
    printf( oh no )

}
```

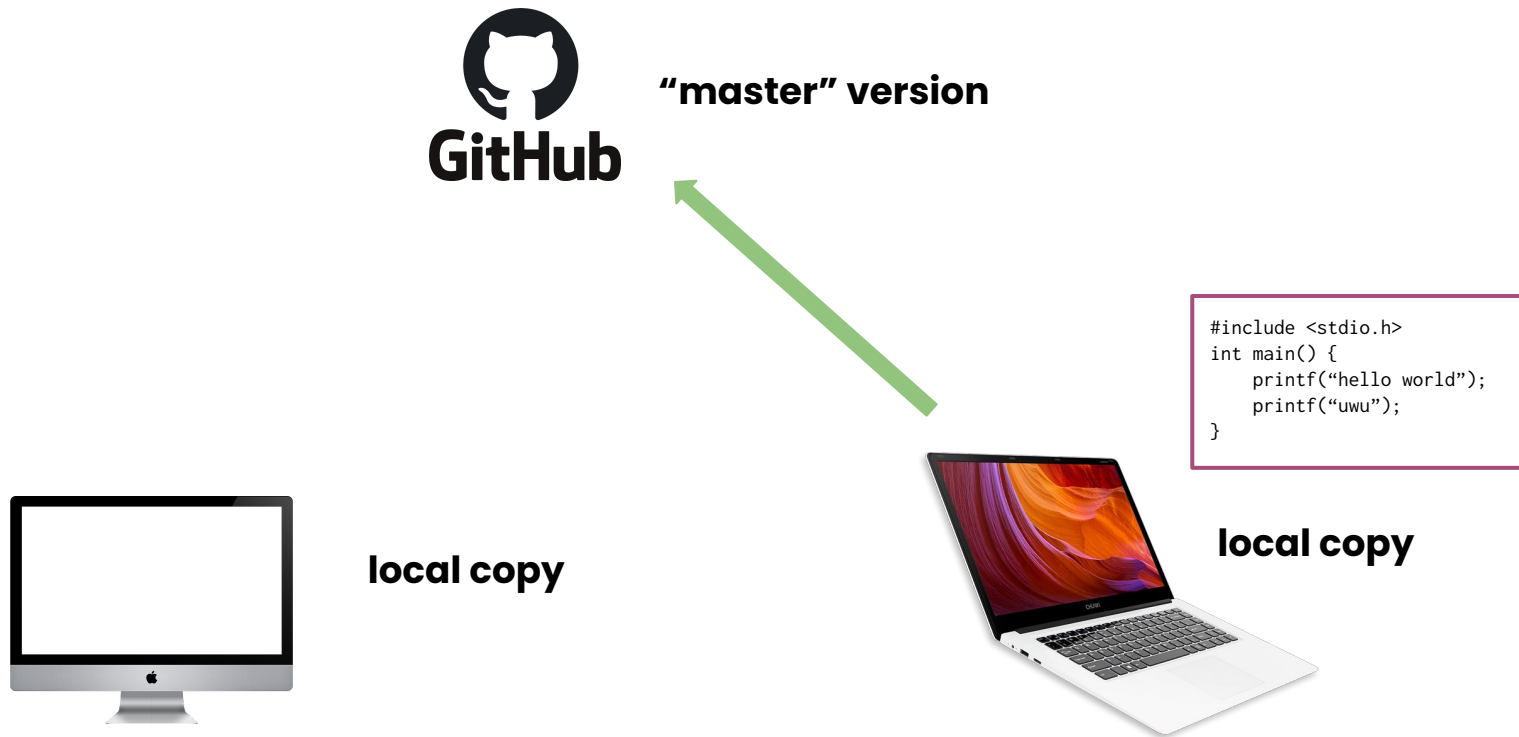


your friend



you

A visual representation



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



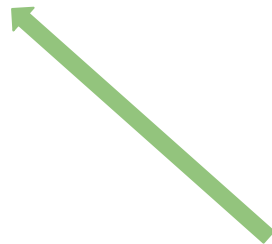
acm.hack

A visual representation

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



"master" version



```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```

local copy



local copy



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

A visual representation

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



"master" version

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



local copy

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



local copy

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

A visual representation

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



"master" version

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
    printf( oh no )
}
```



local copy

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



local copy

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

A visual representation

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



"master" version

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
    printf( oh no )
}
```



local copy

```
#include <stdio.h>
int main() {
    printf("hello world");
    printf("uwu");
}
```



local copy

README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

So how do I get my code from my laptop onto GitHub?



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>

Setting up the Hub of the Git

Everyone on your team:

- Make an account at <https://github.com/>
 - I recommend using the same email you used earlier when configuring git on your personal device



Setting up the Hub of the Git

Note: [square brackets] are placeholders
Eg. git remote add origin https://github.com/katelynsyu/Test.git

One person on your team:

- Make a github repo at <https://github.com/new>
 - Make sure it's a public repo
- Create a git repo in the folder of your choice on your device (`$ git init`)
- In that repo:
 - Create a commit
 - `$ git remote add origin [github repo url]`
 - `$ git branch -M main`
 - `$ git push -u origin main`
- Now you have connected and synced the github repository to your personal repository. You can push and pull between the two repos now.



Setting up the Hub of the Git

Note: [square brackets] are placeholders
Eg. `git clone`
`https://github.com/katelynsyu/Test.git`

Everyone else:

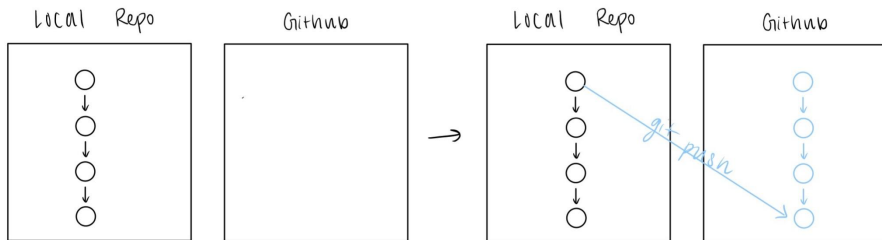
- Find the url of the repo your teammate just created
 - You can find it under the repositories tab when you search up your teammate's user on github
 - It's also the same as the url your teammate used on the last slide to connect their local repo to the github repo
 - The creator of the github repo can invite you to collaborate
- `$git clone [github repo url]`
 - This gets a copy of the github repo onto your personal device
 - You can push and pull between the your personal repo and the github repo now, fully able to collaborate with other people on the code



How do we get code to Github?

`$ git push`

- Copies the changes you made to your local repo to the online repo
 - Now, collaborators can see the most updated version of the code



**"master"
version**

git push


```
#include <stdio.h>
int main() {
    printf("hello
world");
    printf("uwu");
}
```



**local
copy**



How do we get code to Github?

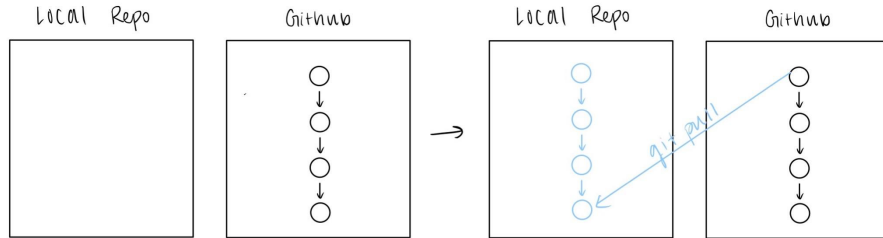
- Create a commit
- `$ git remote add origin [github repo url]`
- `$ git branch -M main`
- `$ git push -u origin main` 

- Copies repo on local device to remote repository
 - Before this line, there was a remote repository that was linked to the local one, but it was empty
- Pushing means that you are ready for everyone to see what you have changed and contributed to the project

How do we get code from Github?

`$ git pull`

- Syncs your local repo to the repo on github
 - Copies any changes to the github repo to the repo on your local device



```
#include  
<stdio.h>  
int main() {  
    printf("hello  
world");  
  
    printf("uwu");  
}
```



**"master"
version**



**local
copy**

```
#include  
<stdio.h>  
int main() {  
    printf("hello  
world");  
  
    printf("uwu");  
}
```



Demo Part the Third

- Create a github repo
- Link the local repo to the remote repo



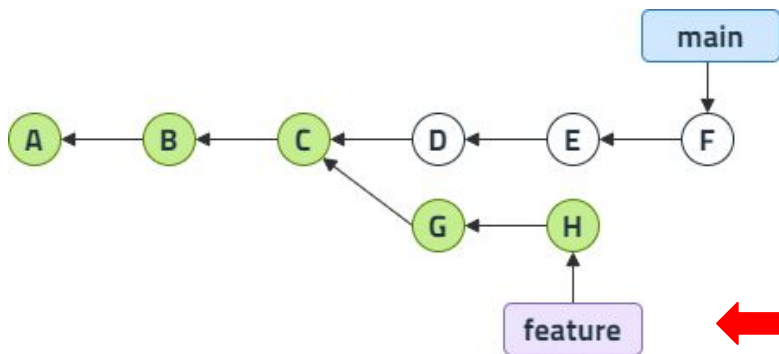
Branching

- Create a commit
- `$git remote add origin [github repo url]`
- `$git branch -M main`
- `$git push -u origin main`

It's a whole topic.



It relates to...



This thing (whaaaa.....?????)



Branching: An Analogy

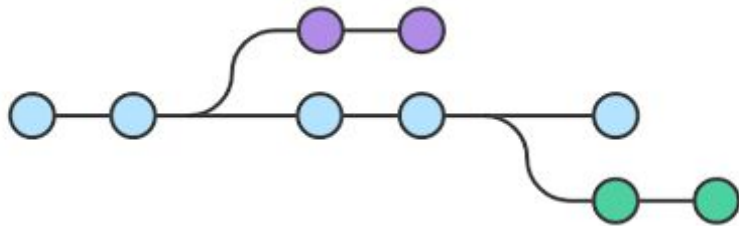
Back to video games - multiplayer this time

How do we speedrun it? Division of labor

- Have different people go off and do different side quests

This is branching!

- You can do the same when coding your project!
Have different people develop different features of your app to get everything done in a timely manner.

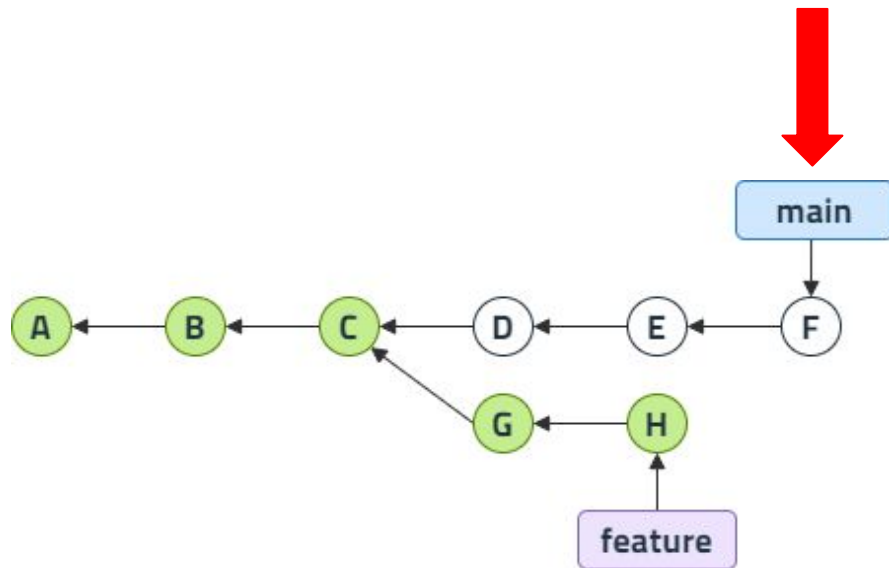


The main branch

Every repo has at least one branch, the main branch

This is the branch that

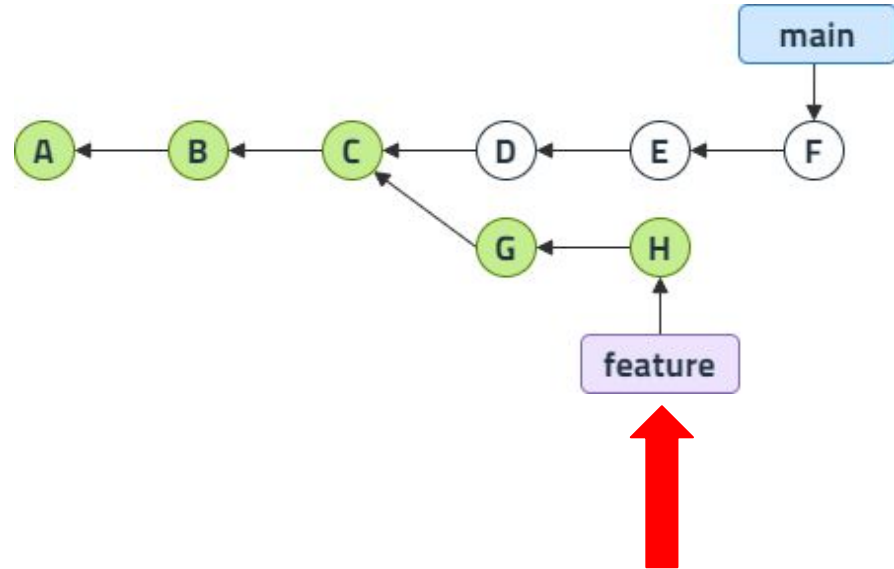
- Everyone looks to for the latest version of the working product
- Try not to submit commit to main
 - Main should only be edited we know the code is flawless and we want to include it in the final version.



What's the feature branch?

Branches in general are

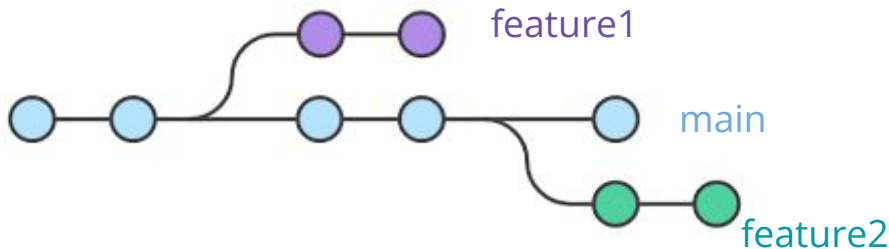
- Isolated development areas where we can commit changes without bothering the main branch
- Allow you to track changes when developing a feature of the app
 - Does not introduce buggy code to main



How do you list your branches?

`$ git branch`

- Lists all the branches on your local repo
- `$ git branch -r`: lists all branches pushed to the Github repo
- `$ git branch [branch name]`: creates a new branch called [branch name]



For a repo like this, ``git branch`` outputs:

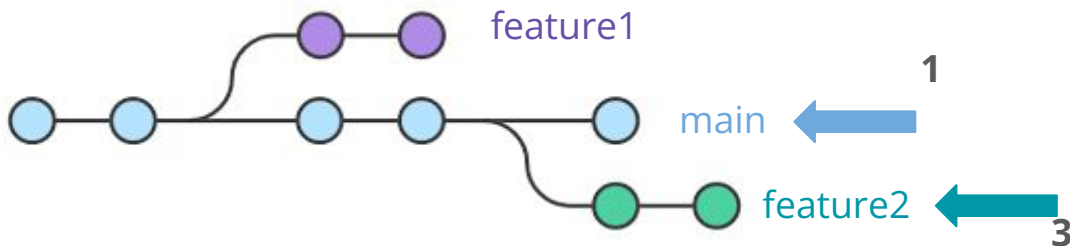
```
main
feature1
feature2
```



How do you switch branches?

`$ git checkout [branch name]`

- Switches you to the branch called [branch name]
 - Now the edits you make to the working directory will be committed to this branch's snapshot timeline
- `$ git checkout -b [branch name]`: creates a new branch called [branch name] and switches you to that branch



1. originally on **main**
2. `$ git checkout feature2`
3. Now on **feature2** branch

Branching Commands

Note: [square brackets] are placeholders
Eg. `git switch createFAQ` switches to a branch called 'createFAQ'

```
$ git switch [branch name]
```


- Switches to the branch called [branch name] if such a branch exists
- `$ git switch -c [branch name]`: git creates a branch named branch name and switches you to that branch



Pushing and Pulling Branches

Pushing:

```
$ git push -u origin [branch name]
```

- Pushes the branch you are on to the remote repository.
 - [branch name] should be the name of the remote repository branch
 - Recognize this?
- Create a commit
 - `$ git remote add origin [github repo url]`
 - `$ git branch -M main`
 - `$ git push -u origin main` 

Pulling:

If someone else created the branch

```
$ git branch -r
```

- All branches listed will be of the form: origin/[branch name]

```
$ git checkout [branch name]
```

Then just `git pull` as normal



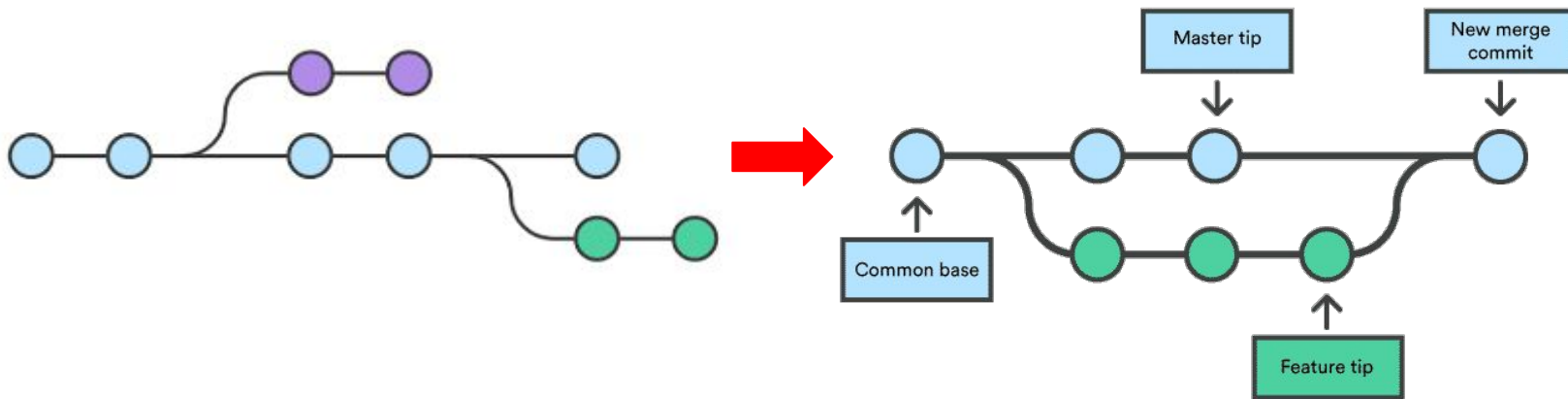


Demo Part the Fourth

- Create branch
- Push the branch to remote

Merging

How can we get development on our branches to show up in our finished product?



Merging: An Analogy

Back to the video games

- Everyone finished their individual quests.
- To finish the game, we need to bring everyone's progress together
 - ie. unite all the work everyone has done to show how much work has been done in culmination

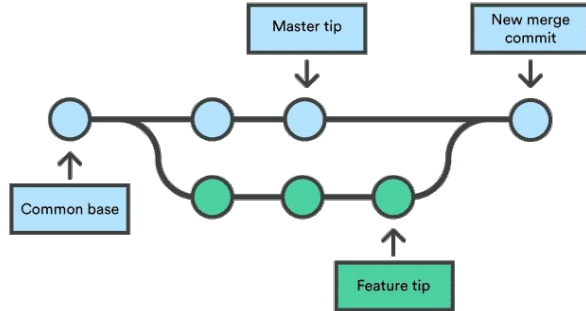
This is merging!



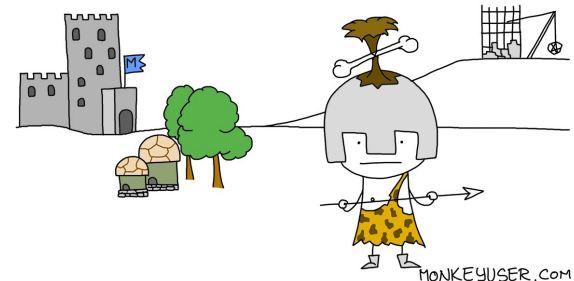
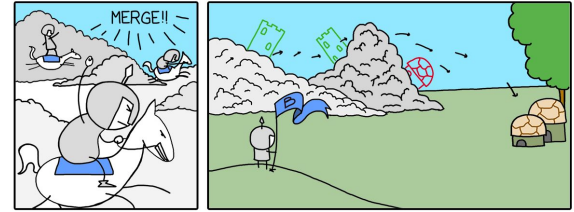
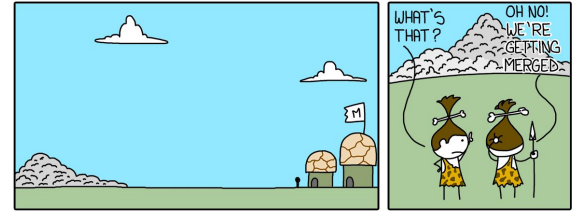
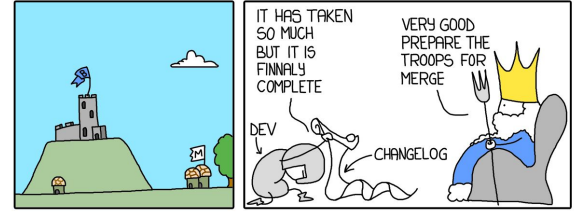
Merging

Merging is when you unite 2 or more branches

- We recommend only merging 2 branches at a time (otherwise it's messy)
- You end up mashing the features developed in both branches into one commit



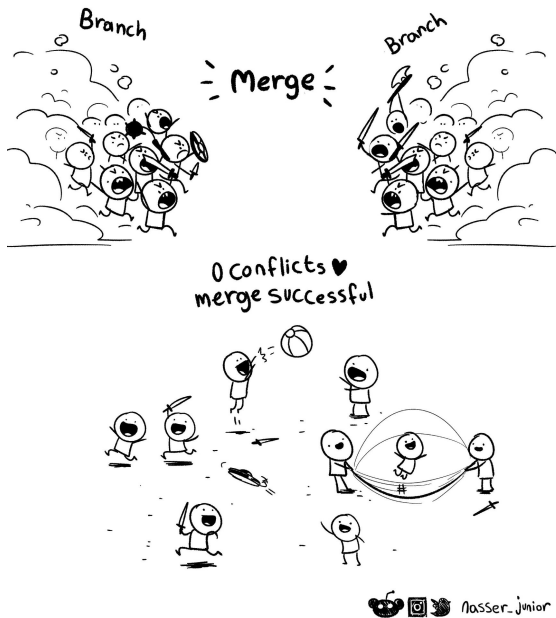
MERGING BRANCHES



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>

Merge Conflicts (may the battle for the best code begin... jk jk... unless...)

Ideally, merging looks like this:



In actuality, we kind of end up with this problem:

Two people ended up doing the same side quest (for reasons ok... it happens).

How do we merge now? There's progress that overwrites other people's contributions.

This is a **merge conflict**!

How do we merge?

You gotta figure out whose progress you want to take.

Merge Conflicts (Godspeed, friends)

Merge conflicts in git occur when trying to merge two branches that have different code in the same corresponding region

Don't worry, they rarely happen.

Merge conflicts tend to look like this:

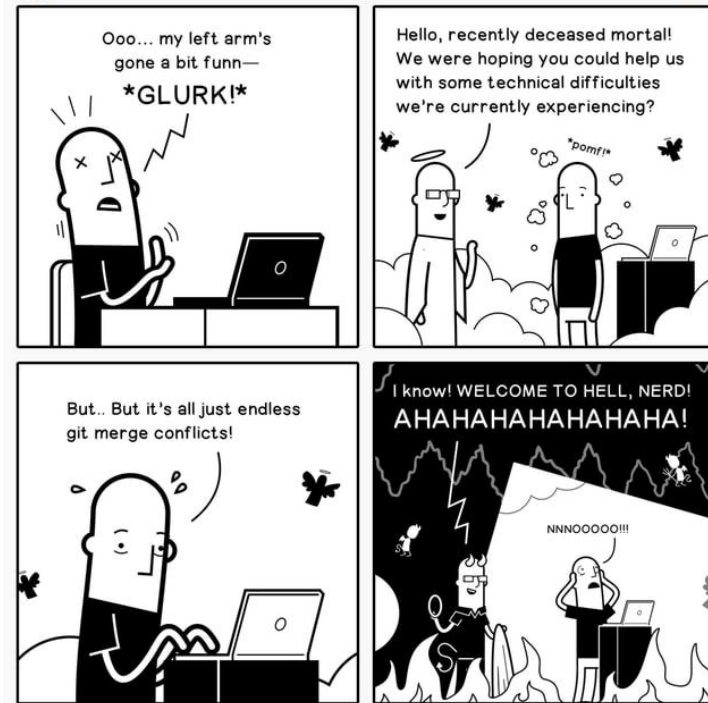
```
<<<<<< HEAD
```

```
We're all in a yellow submarine.
```

```
=====
```

```
Not all cabbages are created equal.
```

```
>>>>>> new-branch
```



Merge Conflicts (Peace comes... after the war)

<<<<<< HEAD

We're all in a yellow submarine.

=====

Not all cabbages are created equal.

>>>>>> new-branch

This means that one commit has “Not all cabbages are created equal” where the other commit has “We’re all in a yellow submarine”. Git doesn’t know which version to take.

Solving merge conflicts in git:

- You gotta go through the commit, figure out where the conflicts are, and figure out whose code you want to take or how to integrate both people’s work. (Godspeed)
- To select the version of the code you want,
 - Delete the text I have colored purple
 - Delete the version of the code you do not want



Git commands

`$ git merge [branch name]`

- Mashes the latest commit from [branch name] into the branch you are on
- You never want to be on main and call this command because we only add commits to main that we are ABSOLUTELY 220% CERTAIN work.
 - Always merge on your feature branch in case something breaks
 - Once you resolve merge conflicts, etc etc, then you merge to main

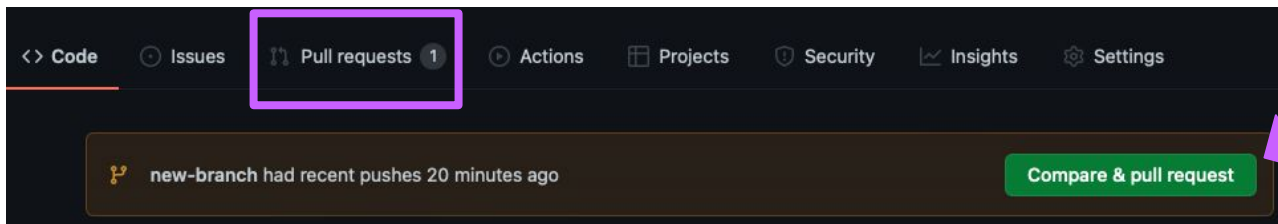
`$ git merge --abort`

- Aborts the merge



Steps to merge

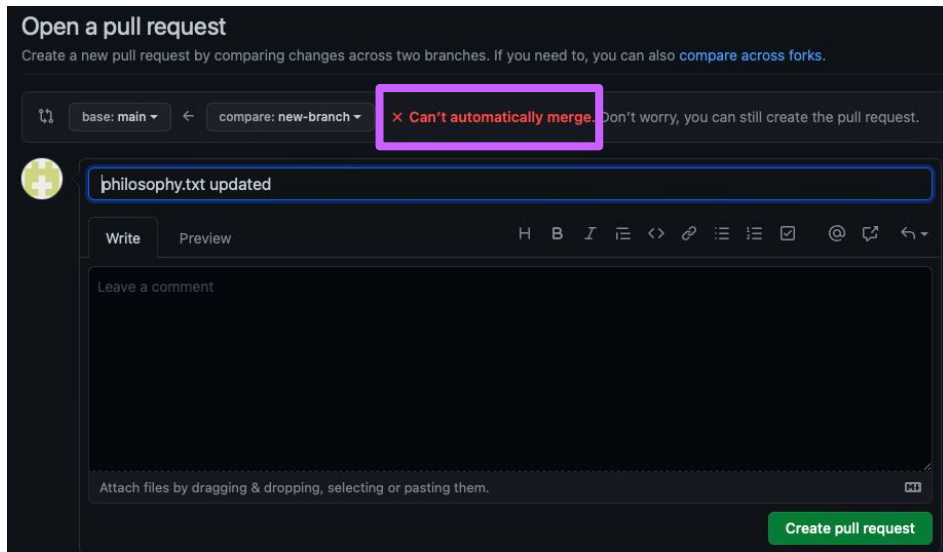
1. Switch to your feature branch
2. Push to remote branch: `$ git push -u origin [remote branch name]`
3. Go to the online github repo
 - a. Under the Pull requests tab, you'll see a notification that says the branch you created had recent pushes and would you like to 'Compare and Pull Request'
 - b. Yes, you would like to



Aside: what's a pull request (PR)

PRs are a safe and easy way to merge branches. They allow us to review code in github and look at the potential changes.

If you don't see the boxed text saying that you can't automatically merge, create the pull request and immediately merge.

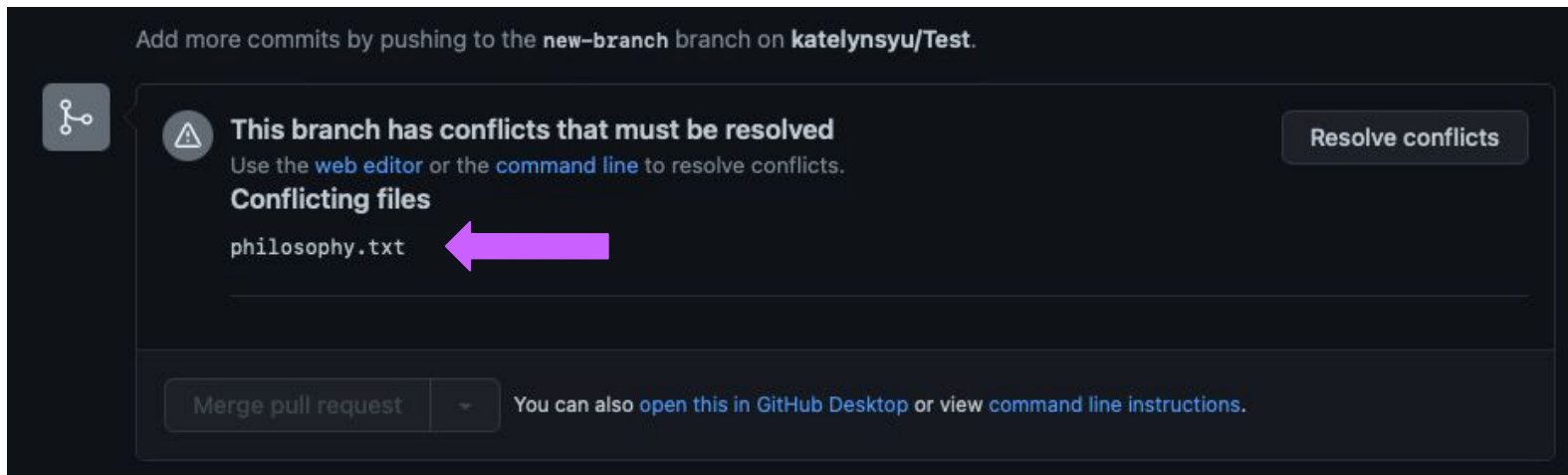


Looks like we've got a merge conflict that we'll have to settle. We can make a PR anyways.



Aside: what's a pull request (PR)

The PR will even tell us where there's a merge conflict (if there is one)



Steps to merge

4. Update local main branch

- a. On terminal, switch back to the main branch: `$ git checkout main`
- b. Pull from remote: `$ git pull`

5. Switch back to your feature branch: `$ git checkout [branch name]`

6. Merge main to your branch: `$ git merge main`

7. Fix the merge conflicts

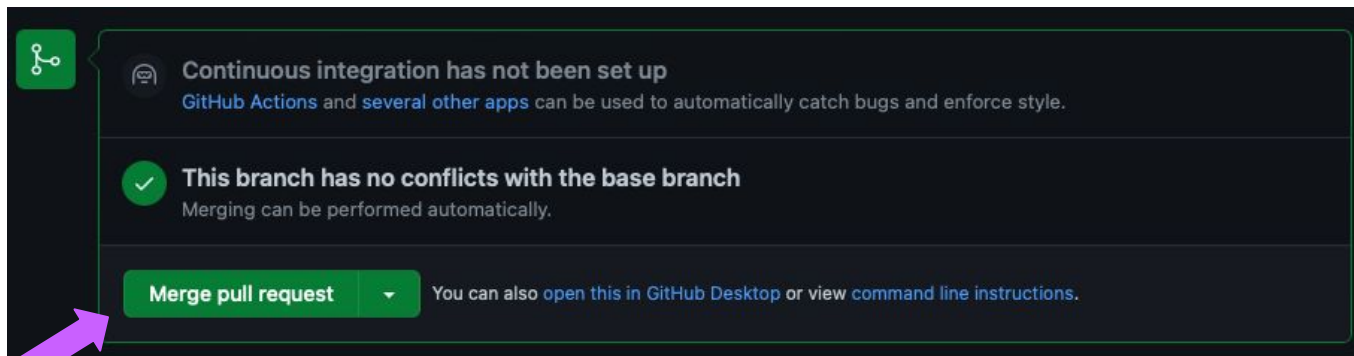
8. Add and commit the changes: `$ git add .` and `$ git commit -m "[message]"`

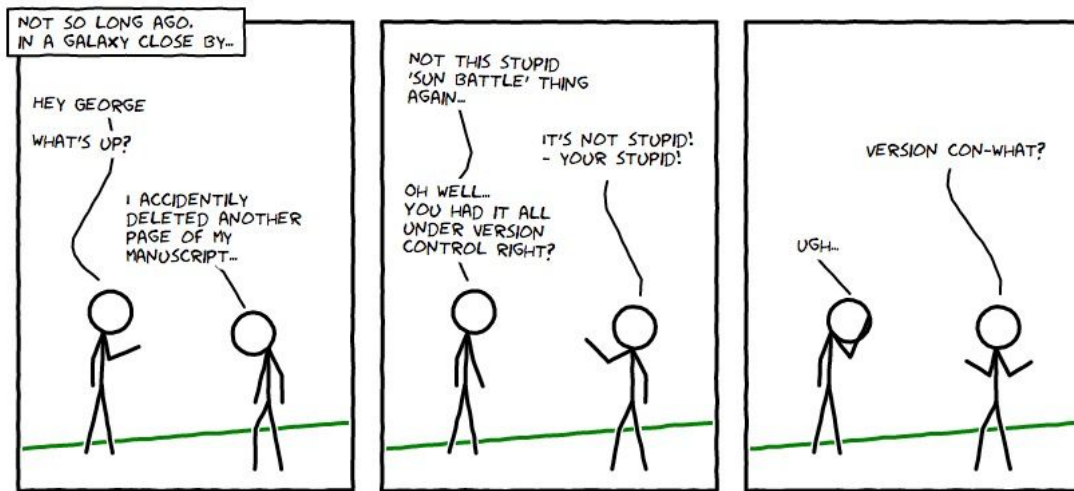


Steps to merge

9. Push to remote branch: `$ git push -u origin [remote branch name]`

10. Go back to github. You're PR should say that it's ok to merge. Just click merge.





- Merging with Merge Conflicts

Helpful Tidbits

.gitignore

- a file in the home directory of your project that tells git which files to never include in the commits
- Sample:

ignore-me.txt

#ignores the text file ignore-me.txt

*.log

#ignores all files that end with .log

some_folder/

#ignores the folder some_folder and all of its contents

This is a comment.
The file will not see
this



Helpful Tidbits

`$ git stash`

- temporarily shelves changes you've made to your working directory so you can work on something else

`$ git stash apply`

- This is how you re-apply stashed changes when you want to work on them again

`$ git diff --staged`

- Prints out all the differences between your staged files and the latest commit

`$ git reset --soft HEAD~1`

- Reverts the state of the repo to the previous commit without deleting the changes in the working directory that you have made since
 - You'll still see them if you do `$ git status`



Helpful Tidbits

`$ git blame [file]`

- For each line of the file, lists the line's author, the date they inputted the line, the commit that the author added it
 - This is how you can figure out how bugs were introduced



README: <https://github.com/uclaacm/hoth9-workshops/tree/main/intro-to-git>



acm.hack

Helpful Tidbits (Bash edition)

`$ cd [folder name/path]`

- Changes your current directory by going into the folder or path you name

`$ ls`

- Lists all folders and files in the current folder
- `$ ls -a`: lists all folders and files in the current folder, including hidden ones



Resources you might find useful:

- Git Cheat Sheet:
<https://www.git-tower.com/blog/git-cheat-sheet/>
- Useful Bash Commands:
<https://www.educative.io/blog/bash-shell-command-cheat-sheet>
- 'Git' help from mentors
 - We sometimes know things



Thanks a-minion for
listening, guys!

