



Team Lead 1 Presentation

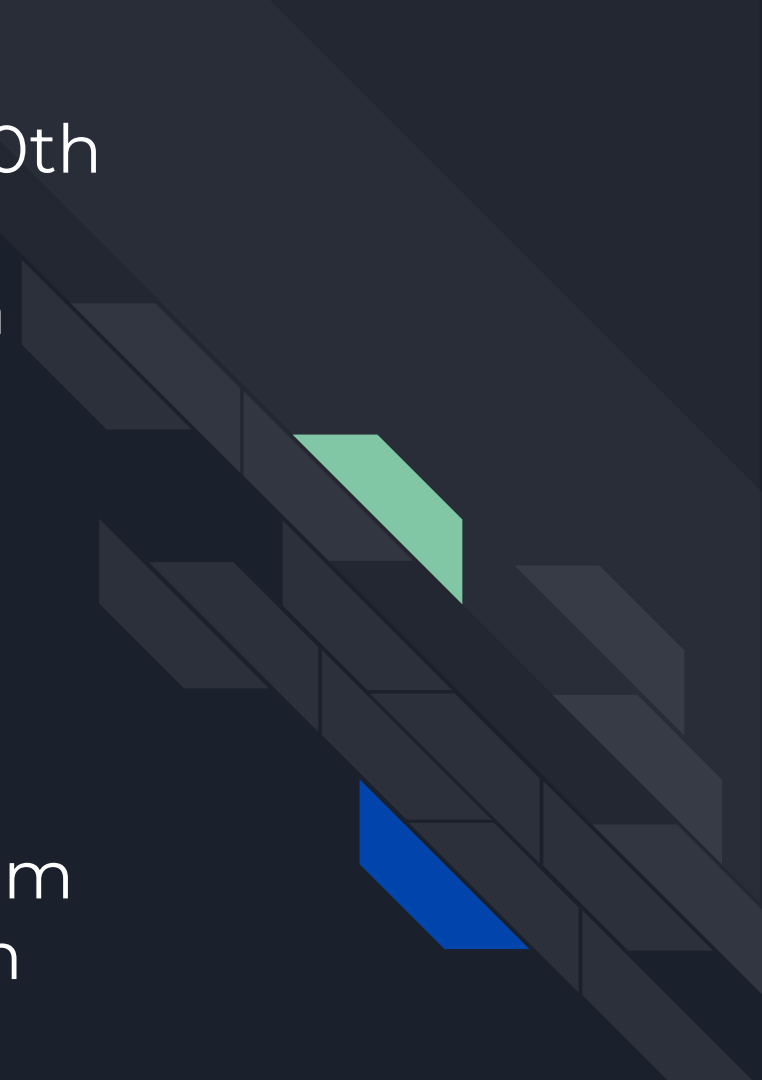


Presented by: Dawson Burgess, Bryce
Hendrickson, Cole Halvorson

Important Due Dates
coming up



- Platformer Game - Sept 20th
- Champion Doc - Sept 17th
- RFP - Sept 21st
- SA (systems analysis)
presentation - Sept 22nd
- TL2 presentation (minimum
viable product) - Sept 29th



Git: Overview - Dawson



What is GIT?

Git is a tool used for source code management.

It is a free and open-source version control system used to handle small to very large projects efficiently.

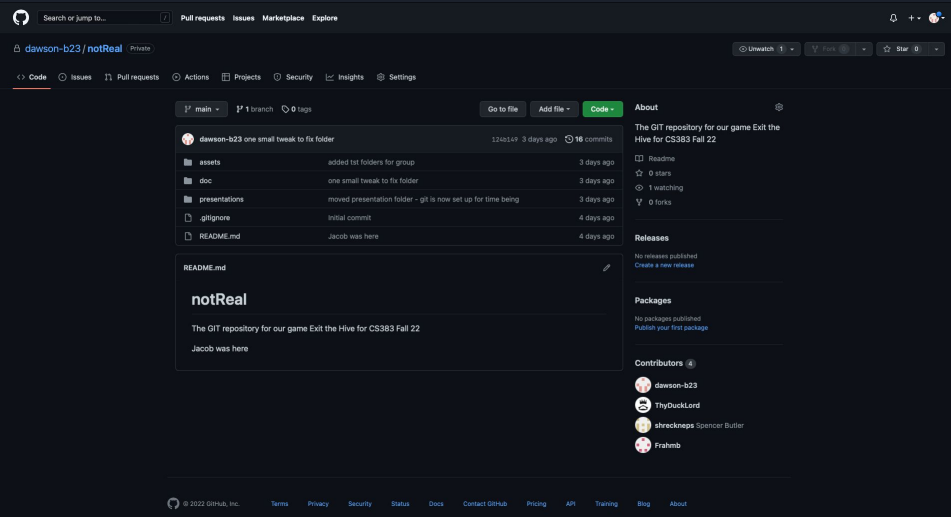


Git is used to track **changes in the source code**, enabling multiple developers to work together on non-linear development.

What is the difference between Github and GIT?

In simple terms, Git is a version control system that lets you manage and track your source code history.

GitHub is a cloud-based hosting service that lets you manage Git repositories. If you have any projects (open source) that use Git, then GitHub acts as a way to better manage them.






More on Version control

Version control, also known as source control, is the practice of tracking and managing changes to software code. **Version control systems are software tools that help software teams manage changes to source code over time.**

This will be very important to the class as time progresses in the projects. Later on, if there is a mistake and the code is suddenly not working, you can revert to a working version or until the changes that cause it to crash are found.



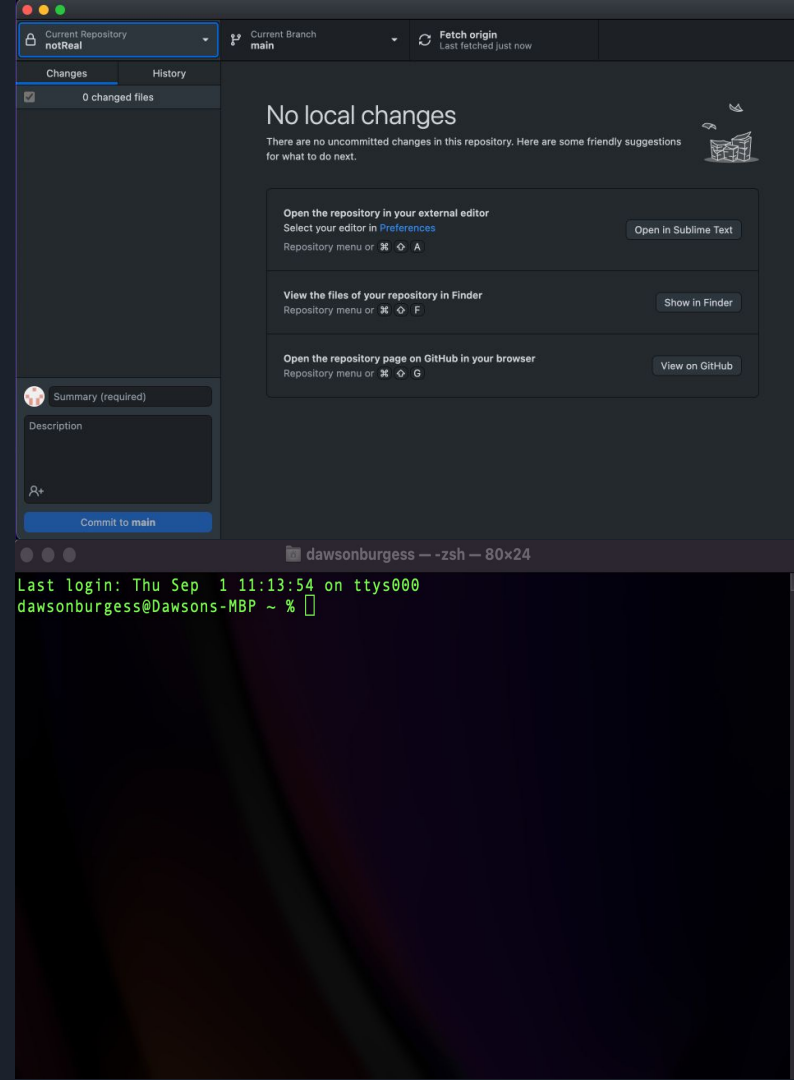
Why do we need to use it?

In theory, you could do the whole project with absolutely no GIT or version control at all - but it would be incredibly difficult, especially with a team.

Version control systems (GIT) allow multiple developers, designers, and team members to work together on the same project. It helps them work smarter and faster! **A version control system is critical to ensure everyone has access to the latest code and modifications are tracked.**

Command Line Vs GitHub Desktop

- GIT was originally designed and integrated using the command line, but that is often intimidating towards inexperienced users
- There is also a **Github desktop application** that will solve this if you are intimidated/uncomfortable with the command line.
- Specifics of how to use both will be covered in later sections of this presentation.



Git: Getting Started | Command line version - Bryce



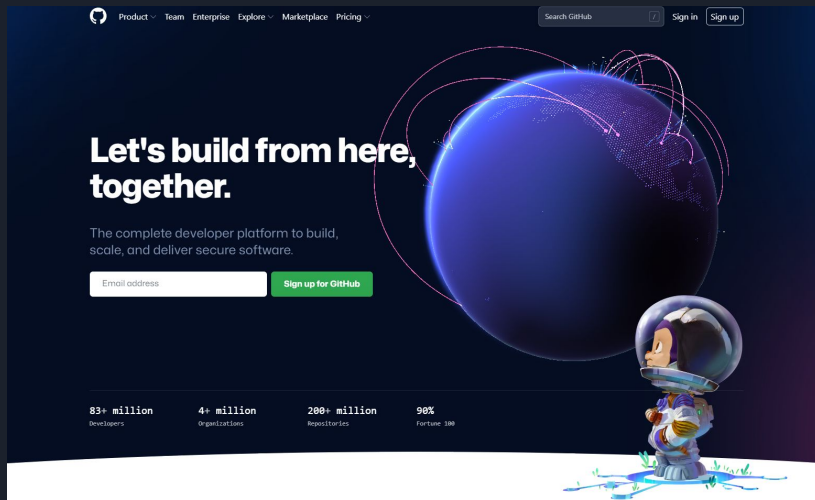


Git: Getting Started - Overview

- Account creation
- Creating a repository
- Installing Git access for PC (Git bash) & Mac (Command Line)
- Setting up an access tokens
- Git global variables

GitHub - Account Creation | github.com

- Start with creating an account for GitHub
- Enter email and create a password
- Enter a username



Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ bc.cs383@drbc.com

Create a password
✓

Enter a username
→ Dr.BC Continue



GitHub - Account Creation | github.com

- Verify email

You're almost done!

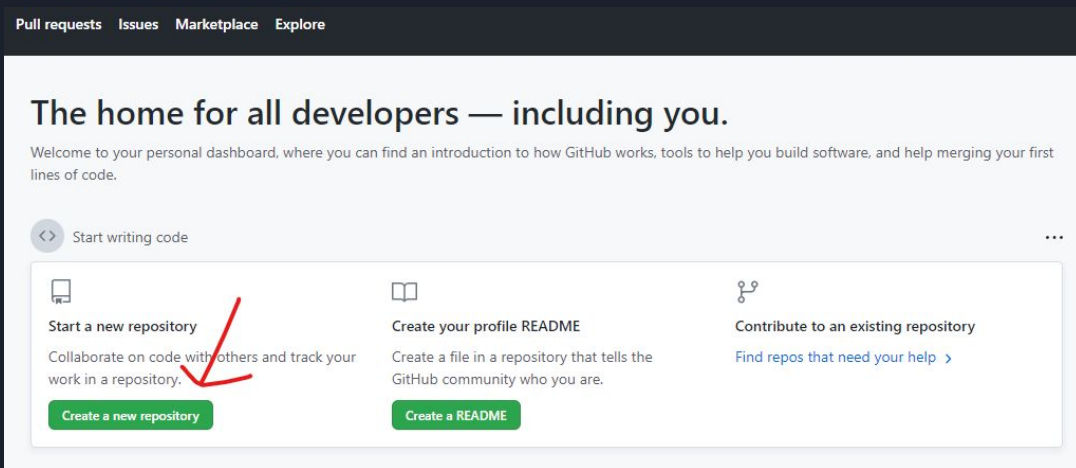
We sent a launch code to `bc.cs383@drbc.com`

→ Enter code

Didn't get your email? [Resend the code](#) or [update your email address](#).

GitHub - Creating a repository

- Sign in at github.com & click “Create a new repository”



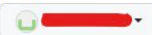
GitHub - Creating a repository

- Name
- Description
- Public / Private
- README?
- Gitignore?
- License?

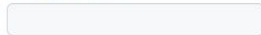
Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

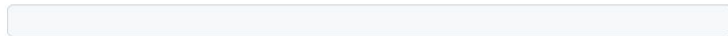


Repository name *



Great repository names are short and memorable. Need inspiration? How about **refactored-octo-robot**?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

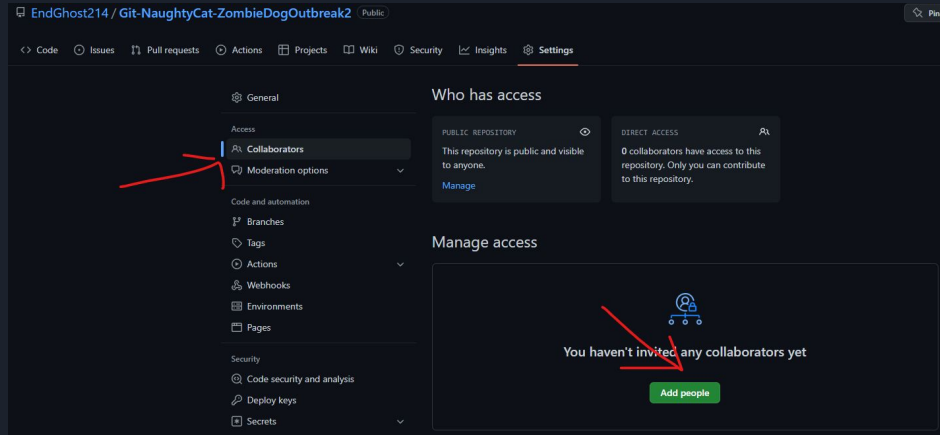
License: None ▾

You are creating a public repository in your personal account.

Create repository

GitHub - Set up Repository for Collaboration

- Navigate to repository settings
- Click on collaborators
- Click add people & add all usernames of collaborating members.



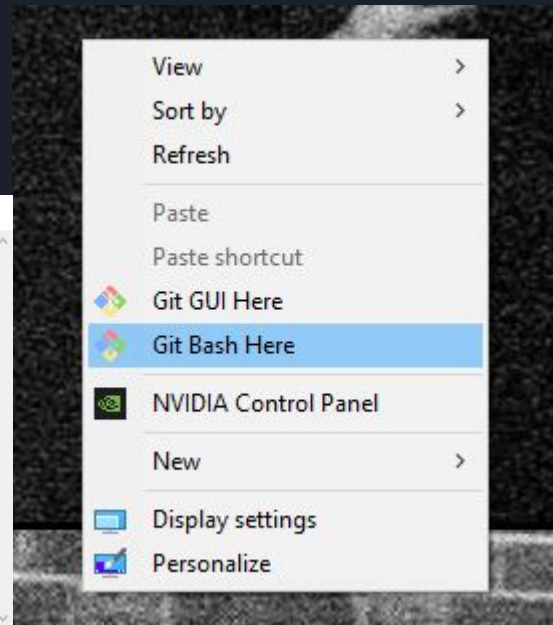
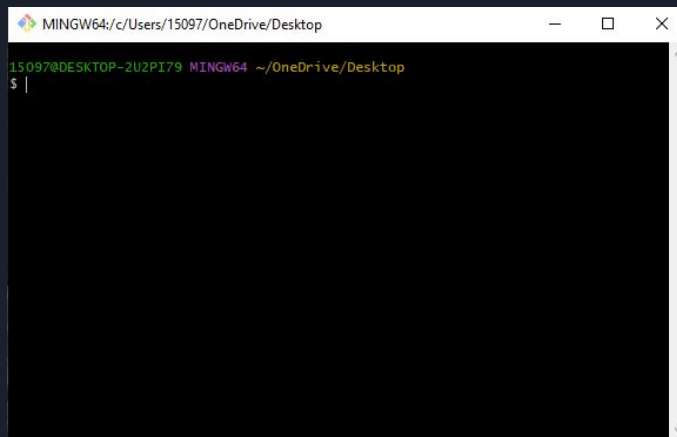


Installing Git on Local Machine

- Mac - Git comes preinstalled on macOS
- Pc - Must download Git
 - Most popular variant is Git Bash

Git - how to interact with GitHub

- PC - right click and bring up Git Bash
- Mac - go to command line





Git - Login

- Must login to git to access GitHub
- Must establish username, email & password
 - Do not establish account password for local git - use a personal access token instead
 - Why?
- Must configure login credentials
- Must proc access request
 - Usually from a pull request



Git - Personal Access Tokens

- Navigate to your GitHub account
 - GitHub -> Profile settings -> Developer settings -> Personal Access tokens -> Generate new token

Git - Personal Access Tokens Continued

- Note
- Expiration
- Access Scopes
 - We will just use repo for our purposes

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Temporary token for API access

What's this token for?

Expiration *

Custom...

09/06/2022

Select scopes

scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events

Personal access tokens

Generate new token

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

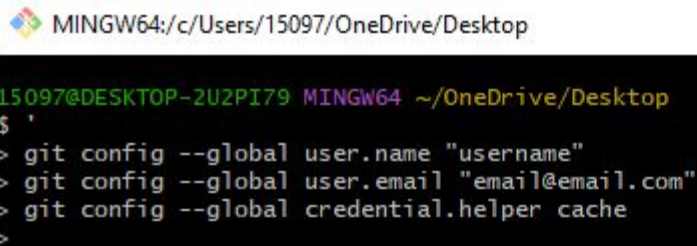
✓ ghp_1PJwQ9ockwusHvNRUzcuJkKCvbRnu20U0wJ

Delete

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Git - Login Configurations

- `git config --global user.name "username"`
 - Establish username
- `git config --global user.email "email@email.com"`
 - Establish email
- `git config --global credential.helper cache`
 - Saves password / access token from login
- `Git config -l`
 - Lists all config settings

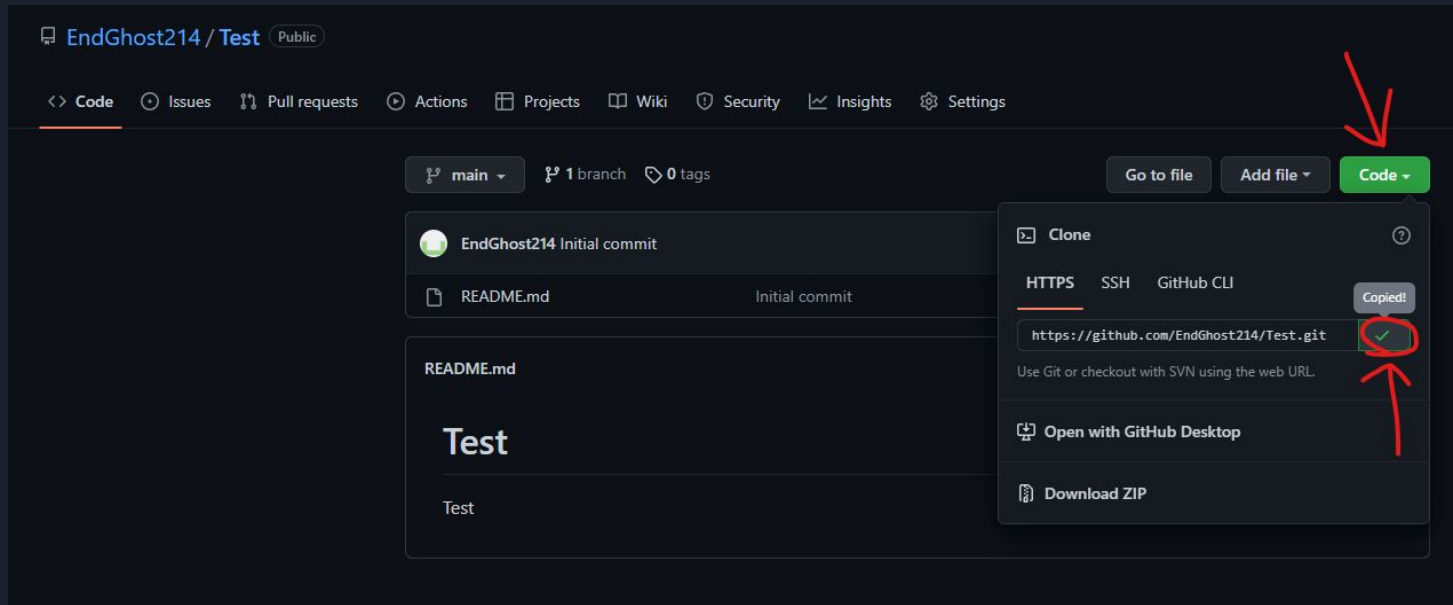


A terminal window with a dark background and light-colored text. The title bar at the top reads 'MINGW64:/c/Users/15097/OneDrive/Desktop'. The terminal content shows a user prompt '\$ ' followed by three lines of Git configuration commands: 'git config --global user.name "username"', 'git config --global user.email "email@email.com"', and 'git config --global credential.helper cache'. Each command is preceded by a greater-than sign '>'. The text is color-coded: the prompt is green, the user name is purple, the email is blue, and the credential helper is yellow.

```
MINGW64:/c/Users/15097/OneDrive/Desktop
15097@DESKTOP-2U2PI79 MINGW64 ~/OneDrive/Desktop
$ '
> git config --global user.name "username"
> git config --global user.email "email@email.com"
> git config --global credential.helper cache
>
```

Git - Let's login!

- Navigate to GitHub profile repositories
- Use the repository that was previously created & copy the HTTPS link to the repository



The screenshot shows the GitHub interface for a repository named 'Test' by user 'EndGhost214'. The repository is public and has one branch, 'main'. The 'Code' button is highlighted with a red arrow, and its dropdown menu is open. The dropdown menu shows the 'Clone' option selected, with the HTTPS URL 'https://github.com/EndGhost214/Test.git' displayed. A red circle highlights the copy icon next to the URL, and a red arrow points to it. The 'Copied!' status is also visible. The repository content shows a README.md file with the text 'Test'.

EndGhost214 / Test Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

Clone

HTTPS SSH GitHub CLI

Copied!

https://github.com/EndGhost214/Test.git

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

EndGhost214 Initial commit

README.md Initial commit

README.md

Test

Test

Git - Let's login!

- Clone repository
- Request a pull once in repository
- Password request will appear on line pointed to by the arrow below - enter personal access token and never login again until token expires!

```
MINGW64:/c/Users/15097/OneDrive/Desktop/Test
15097@DESKTOP-2U2PI79 MINGW64 ~/OneDrive/Desktop
$ git clone https://github.com/EndGhost214/Test.git
Cloning into 'Test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

15097@DESKTOP-2U2PI79 MINGW64 ~/OneDrive/Desktop
$ cd Test

15097@DESKTOP-2U2PI79 MINGW64 ~/OneDrive/Desktop/Test (main)
$ git pull
Already up to date.

15097@DESKTOP-2U2PI79 MINGW64 ~/OneDrive/Desktop/Test (main)
$
```


Using Git: The Basics - Cole





The Big Idea

Once we've set up our repository, it's time to start **versioning** our code

- First we'll need to clone our repository onto our local machine
- Next we'll pull to make sure we're up to date
- Next we'll make some changes or add some files locally
- Next we'll add our files to a new commit
- Then we'll commit our changes
- Finally we'll push our new commit onto the main branch



Git: Clone

What is cloning?

- When you clone a repository, you're downloading a copy of all of the files you have in that repository on github onto your machine
- Now you have **two** versions of your repository!
 - **Local**, on your machine. This is where we make changes
 - **Remote**, on github. This is where we save versions of our code



Git: Cloning a Repository Using Git Bash

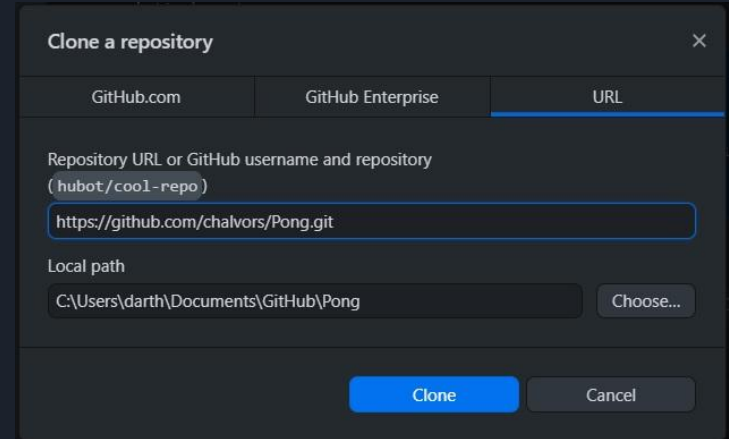
- 1) Navigate to your desired directory
- 2) Type “git clone <repository>”

```
MINGW64 ~/Desktop/School/CS383/Test
$ git clone https://github.com/chalvors/Pong.git
Cloning into 'Pong'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 13 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (13/13), 26.20 MiB | 8.97 MiB/s, done.
```

- 3) Now my Pong repository is on my local machine

Git: Cloning a Repository Using GitHub Desktop

- 1) Click on the “Current Repository” button in the top left
- 2) Click “Add”
- 3) Click “Clone repository...”
- 4) Paste repository github link
- 5) Specify desired path
- 6) Click “Clone”
- 7) Now you’ll see your repository in the list at the top left on the home page





Git: Pull

The pull command incorporates changes from a **remote repository** into the current branch

- This is how we get the latest changes from the repository onto our machines
- In order to see other people's changes to the project, we need to execute a pull



Git: Pulling Using Git Bash

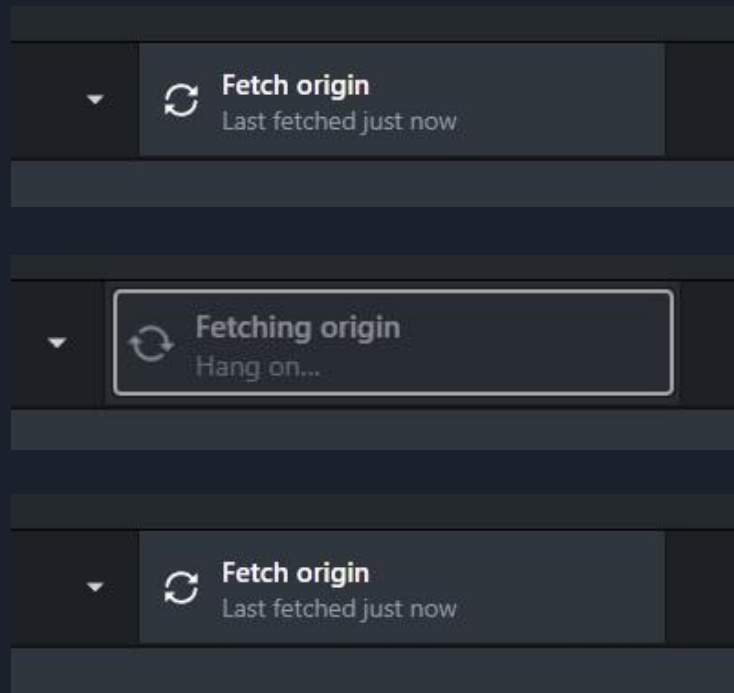
- 1) Navigate to the repository directory
- 2) Type “git pull”

```
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git pull
Already up to date.
```

- 3) We're up to date!

Git: Pulling Using GitHub Desktop

- 1) Click the “fetch origin” button in the top right
- 2) Wait
- 3) We're up to date!





Git: Status

The status command displays the state of the **working directory** and the **staging area**.

Working Directory: umbrella term for all files and folders for your project

Staging Area: files that are waiting to be **committed**



Git: Checking Status Using Git Bash

- Navigate to git repository directory
- Type "git status"

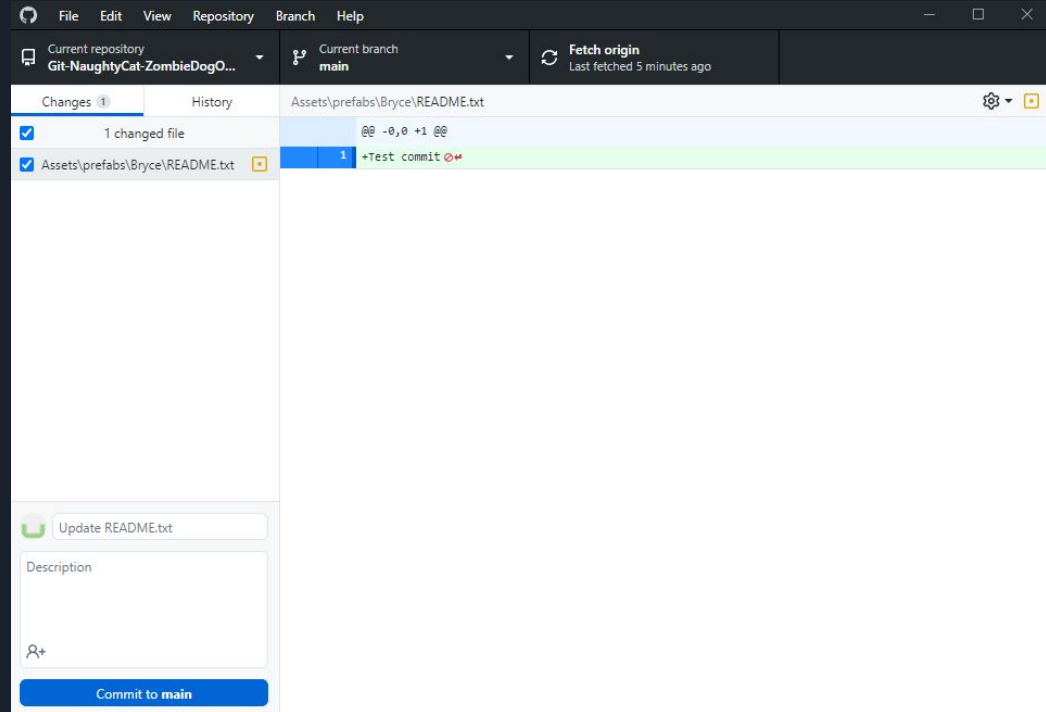
```
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

Git: Checking Status Using GitHub Desktop

- Status is automatic
 - Changes automatically appear in the left column
 - Click on the change to see a preview of the file and how it changed





Git: Add

The add command is one of the commands we will use most often, it adds a change in the **working directory** to the **staging area**

- These can be existing files we've made changes to or new files we've created since our last commit
- Any change in the working directory must be added before committing

Git: Adding using Git Bash

- 1) Navigate to the repository directory
- 2) Type “git add <file>”
- 3) Now our changed file has been moved to the **staging area** and is ready to be **committed**

```
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

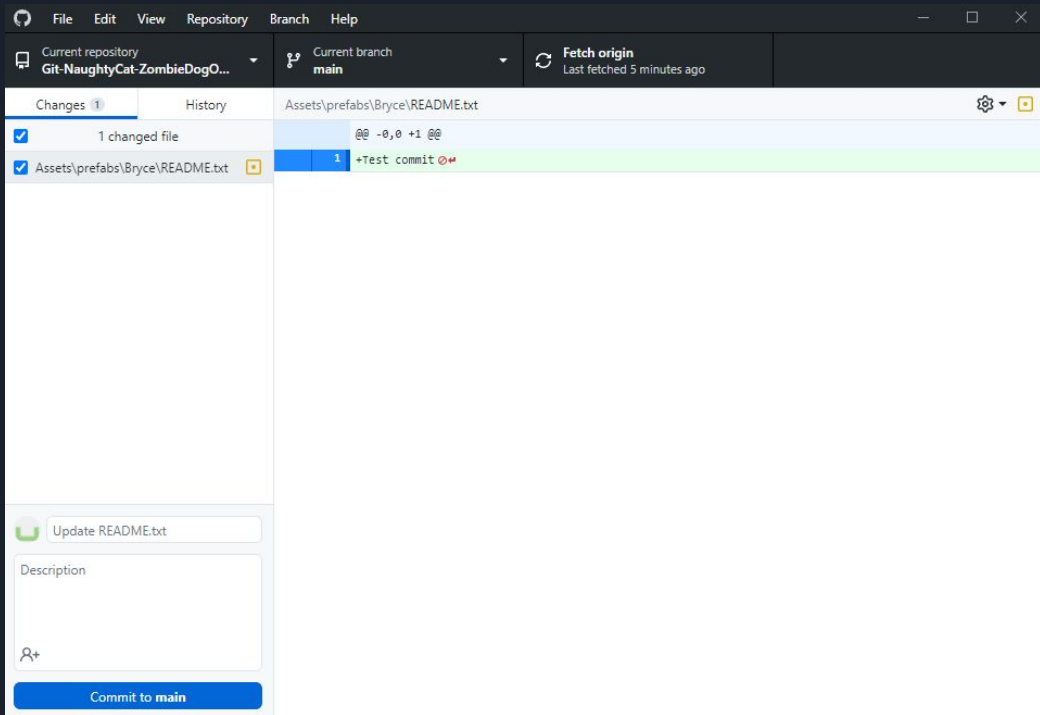
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git add test.txt

darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

Git: Adding Using GitHub Desktop

- Adding is super easy, just select the checkbox on the left for the files you want added to your **commit**
- Changes are automatically added to the next **commit** by default





Git: Commit

The commit command **records changes** to the repository

- Commits are how we record the history of our project
- Commits have log messages
 - It's very important to write good log messages for your commits, that way other people know what you changed
- When our project is done, we should see a long chain of commits, each with a clear log message describing what was changed



Git: Committing Using Git Bash

- 1) Navigate to the repository directory
- 2) Type “git commit -m <log-message>”

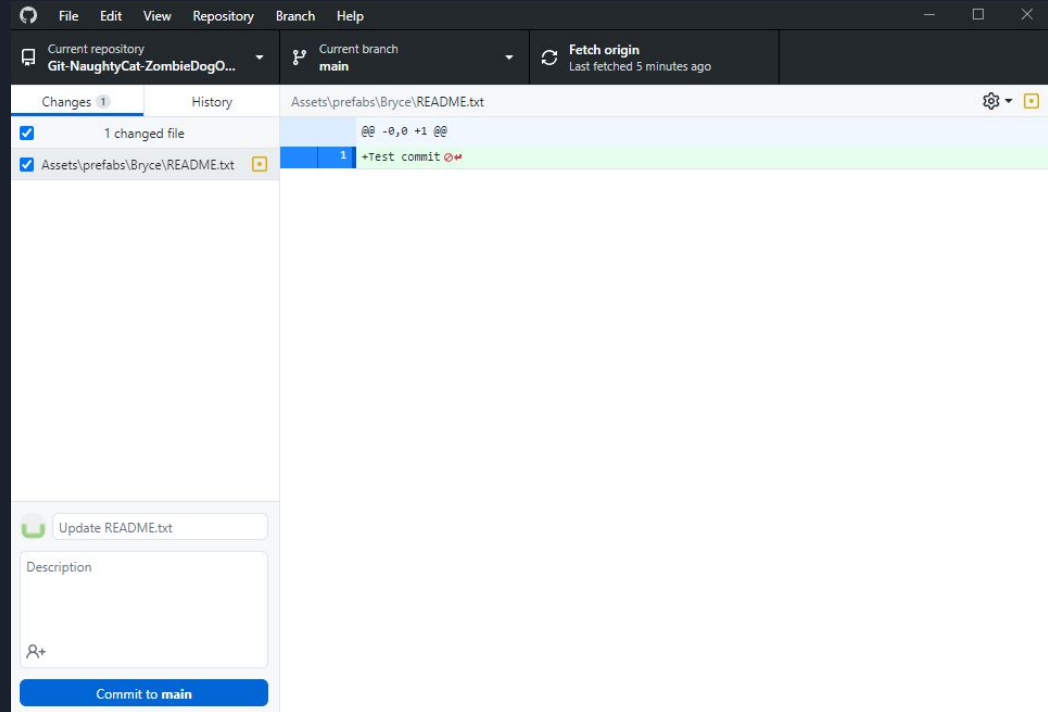
```
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git commit -m test
[main 43dcb26] test
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 test.txt
```

- 4) We've made a commit on the main branch!

Note: the “-m” option lets us enter a log message for our commit right in the command line. If you do not include a “-m” in your commit command Git Bash will open up a vi editor and make you write a log message in there. We recommend using the “-m” option, it's just easier.

Git: Committing Using GitHub Desktop

- 1) Make sure the changes you wish to commit are checked
- 2) Write a log message in the provided box
- 3) Click the “Commit to main” button in the bottom left





Git: Push

The push command is used to upload **local** repository content to a **remote** repository

- This is how we put our local changes onto GitHub
- You might also think of it as updating the remote repository to look like your local repository
- We **push** our local commits to the remote repository
- Kind of the opposite of **pull**

Git: Pushing Using Git Bash

- 1) Navigate to the repository directory
- 2) Ensure you have commits ready to be pushed
- 3) Type “git push”
- 4) We’ve pushed our first commit!

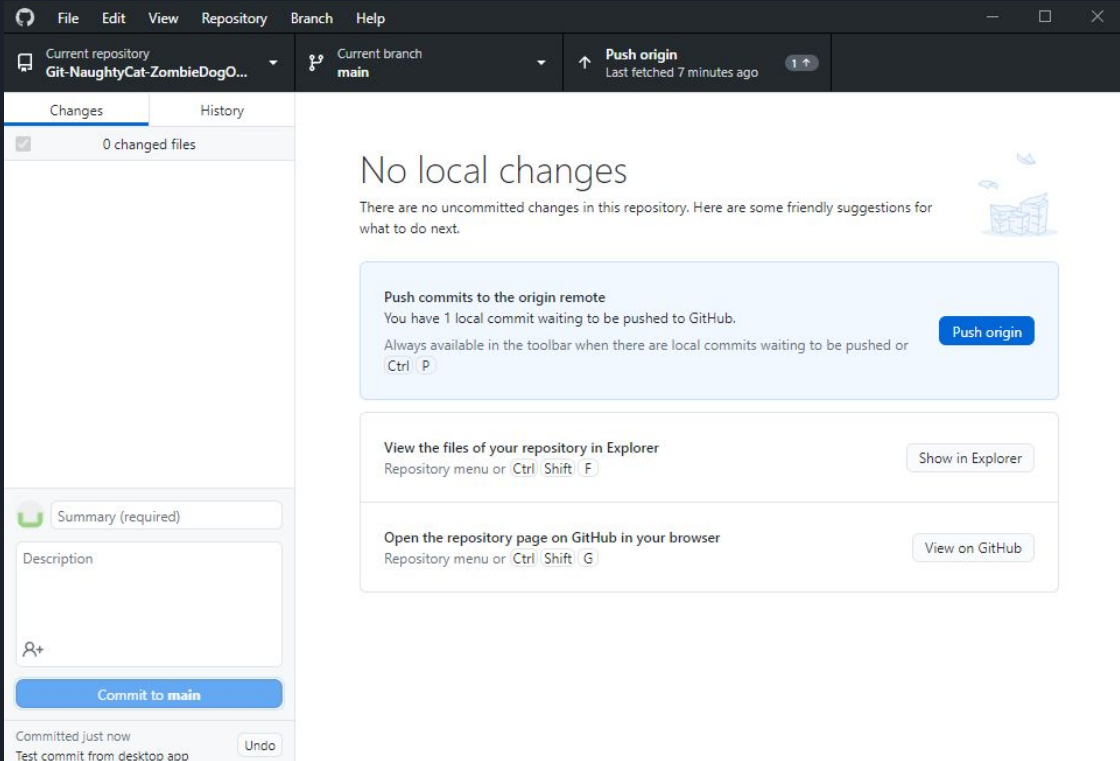
```
darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git add test.txt

darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git commit -m test
[main e48917d] test
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 test.txt

darth@MSI MINGW64 ~/Desktop/School/CS383/Test/Pong (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 645 bytes | 645.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/chalvors/Pong.git
0779f99..e48917d main -> main
```

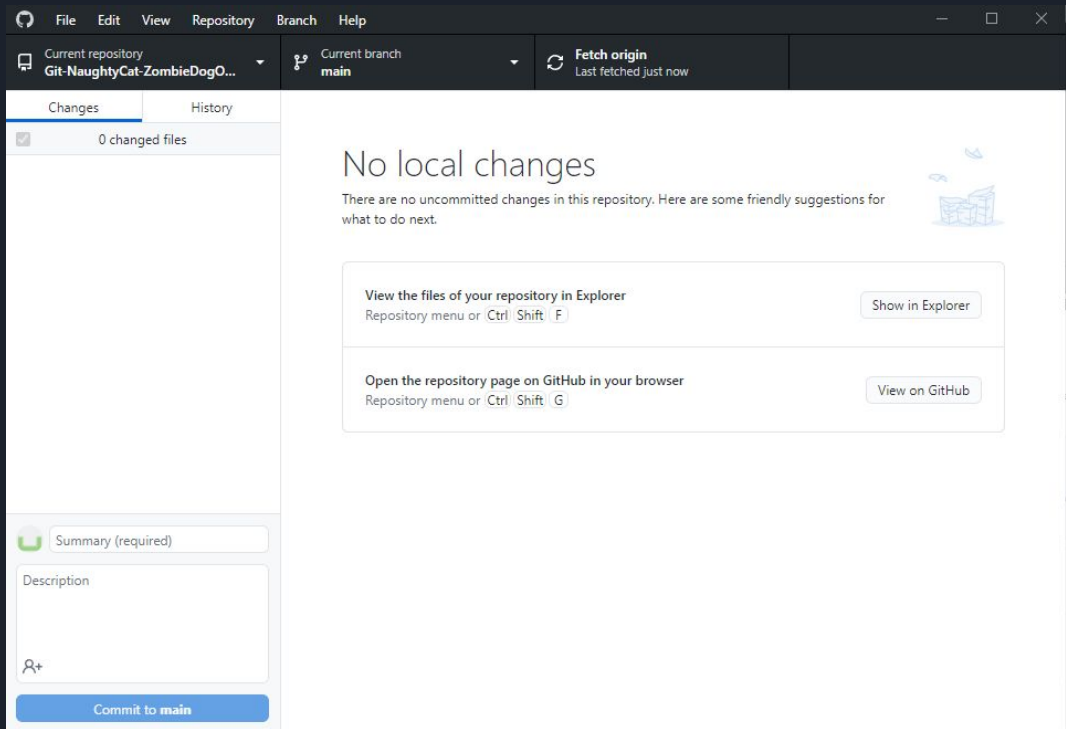
Git: Pushing Using GitHub Desktop

- Committed changes
- Prompting to push
- Click either of the “Push origin” buttons



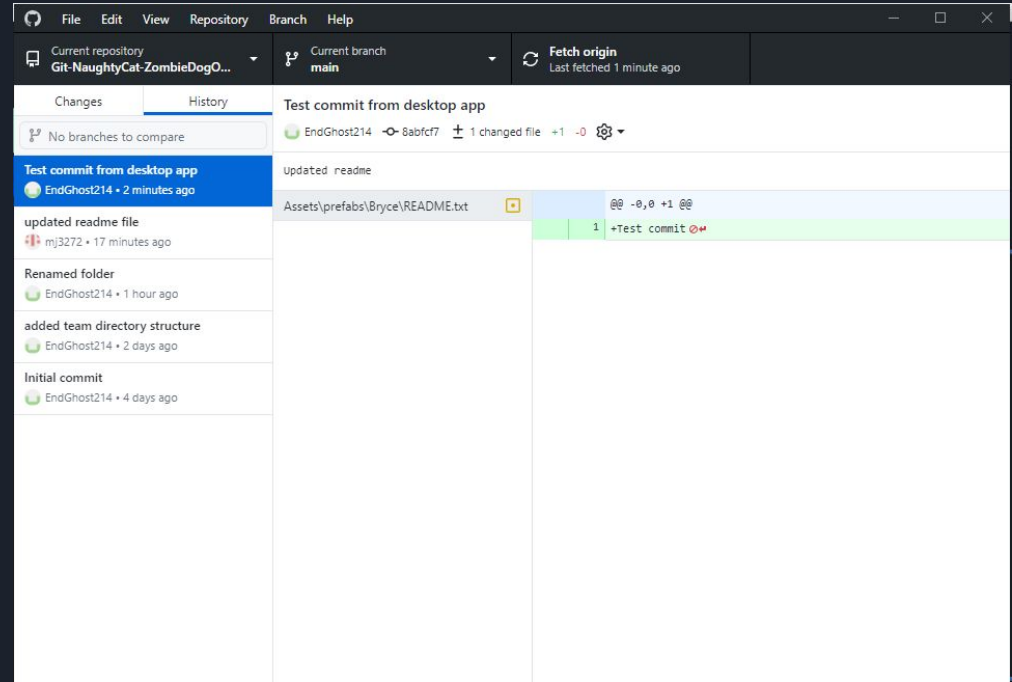
GitHub Desktop - Post Push

- Back to no changes



GitHub Desktop - History

- History tab shows all commits on current branch





A quick note on branches

We will **not** be making branches and merging them into main. Instead we will be committing directly to the main branch

- If you know how to make branches and do merges:
 - Good, you'll use that in the future
 - We won't be doing that for this class
 - "Never commit to master"
- If you don't know what any of that means:
 - Don't worry about it :)
- There are lots of really cool and fancy git commands
 - Feel free to learn them, knowing git is super important
 - Don't use anything crazy for your games, keep it simple!



Git Bash VS GitHub Desktop

GitHub Desktop makes everything really easy!

That's why we strongly encourage you all to use the command line :)

- The command line forces you to understand what you're doing
- We've done lots of Bash before, it shouldn't be too intimidating
- If you're really struggling to use the command line you can fall back on GitHub Desktop
 - At the end of the day they do the same thing but we think you'll learn a lot more using the command line

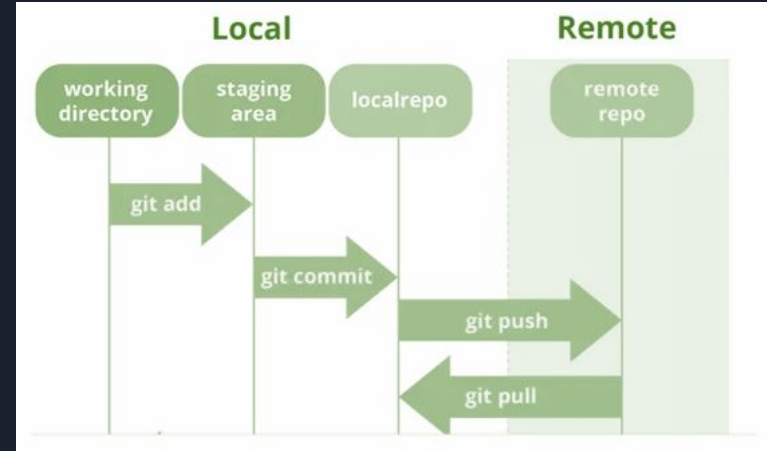
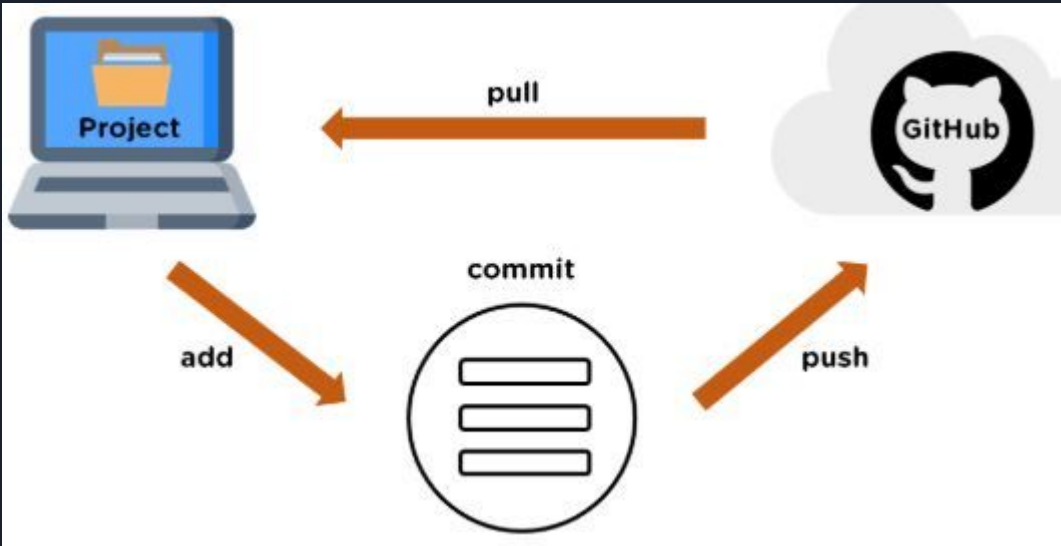


Git Bash + GitHub Desktop Downloads

- Git Bash: <https://git-scm.com/downloads>
- GitHub Desktop: <https://desktop.github.com/>

Git: Tying it All Together

Some Useful Pictures



Git - Live Tutorial - Bryce



Other deliverables - All



Platformer Game
due Sept 17th





Platformer Game - Description and goals

- Use Unity and C# to create a Platformer video game or an equivalent game. Your game must include the following elements:
 - An animated main player that moves in response to the user pressing keys on the keyboard or mouse (or in response to the controller moving if using Vive)
 - Move left
 - Move right
 - Jump
 - Health Bar
 - At least one item should remove health
 - Score Bar
 - Items picked up should add to score
 - Level Goal that needs the player to jump on at least three platforms to reach.
 - Sound.
 - Nice graphics.



Platformer Game - Deliverables

- Code (30 marks) You can either submit a .zip file, or give a link to a GIT repository where (BC) can clone your code, or use the delivery method you are most comfortable with.
- On the demo day show all aspects of the items above to the class (on one of your team's laptops).
- Note: This is a group assignment with the people who have the same TL number as you have.

Platformer Game - Marking Key Sept 17th

- Overall game (Sound, graphics, understandable goal) (10 marks)
- Main player (Movement, graphics (animated)) (10 marks)
- Other (Health Bar, Score Bar) (10 marks)
- MultiMedia:


Team Lead Number: _____

	1	4	7	10
Multi-media	Turned back to camera or lacked necessary equipment to present.	Came prepared and faced camera but spoke too softly or quickly to be heard.	Spoke up well so could be heard in all classrooms. Had some issues with the tech, but mostly knew how to use it.	Conscious of the location of camera and mic. And used them both to full advantage. Also knew how to adjust control the camera so it faced them, and set it back to facing the classroom when done. Knew how to connect their computer to Zoom.



Champion Document due Sept 17th

[Champion - OneDrive \(sharepoint.com\)](#)



RFP (request for
proposal) due Sept
21st

[RFP - OneDrive \(sharepoint.com\)](#)

SA (system analysis) presentation





Everything that will be due on presentation day:

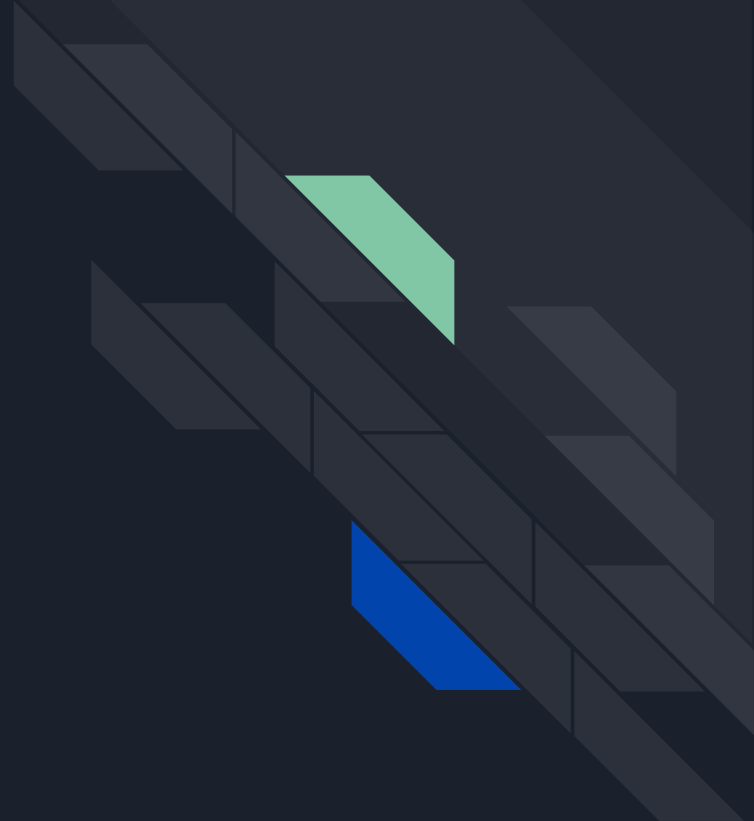
- RFP - due the day before the presentation
- Presentation
 - Storyboard
 - Global Use Case
 - Context diagram and Diagram 0
 - Individual Use Case Diagrams
- Champion
 - Use case diagram & scenario
 - Data flow diagram
 - Test plan (rough since you have not learned this yet)
 - Gantt and PERT charts
- Peer Evaluations


SA Presentation Marking key (available on Dr BC's Website)

Presentation Marking Key	Group_____	Mark_____/40
	Individual	Group
Give a brief introduction to the game <ul style="list-style-type: none"> What is the name of your game? What is the goal? Pretend your game is a story. Tell us the story. 		/4
Introduce Group Members (Have each person briefly describe their roll as you introduce them.)		/1
Storyboard slides (marked separately).		
Introduce project using Context Diagram and any sample screen shots.		/3
Break down project with diagram 0. <ul style="list-style-type: none"> Who is responsible for each sub-system? 		/6
Walkthrough of a global use case involving all members and showing which data is passed between sub-systems.		/6
Person_____ Individual mark ____/20 Re-introduce yourself (Have your asn 0 nickname on each slide as you talk) Introduce your feature (ensure you explain its role in the overall project) Tell how high of a priority your feature is Walk through your use case diagram Estimate the complexity of the work relative to the other features Answer any questions	 /1 /4 /2 /7 /2 /4	

SA: StoryBoard

- Needed for SA and RFP





Storyboard Marking key (separate marking key from SA presentation)

	1	4	7	10
Frames	<p>Overall storyboard is deficient.</p> <p>Frames, pictures, and written pieces are missing or deficient.</p> <p>Frames do not flow together and are messy.</p>	<p>Storyboard is missing frames.</p> <p>Frames are missing pictures and written pieces. Frames are not explained with written piece.</p> <p>Frames do not flow well and are messy.</p>	<p>Storyboard has six or more frames.</p> <p>Frames include pictures and a written piece explaining the frame.</p> <p>Frames mostly flow well and are mostly neat.</p>	<p>Storyboard has seven or more complete frames.</p> <p>Each frame includes a picture(s) and written piece explaining the frame.</p> <p>The frames flow well and are neat.</p>

Storyboard Marking key (separate marking key from SA presentation)

<p>Story & Writing</p>	<p>What game are you developing?</p> <p>Sentence structure and paragraphs are deficient.</p> <p>Grammar, punctuation, and spelling are deficient.</p> <p>Story does not have fluency and makes no sense.</p> <p>Game has no main player.</p> <p>Game has no goal.</p>	<p>I can picture this as being similar to an exist game, but I am not sure.</p> <p>Story contains Incomplete sentences and lacks paragraph form.</p> <p>Writing contains several mistakes in grammar, punctuation, and spelling.</p> <p>Story lacks fluency and is confusing in places.</p> <p>The main is character's description is weak.</p> <p>Game's goal is unclear.❑</p>	<p>Writing gives a good overview of the story beats of the game.</p> <p>Story has complete sentences and paragraphs with a few mistakes.</p> <p>Writing contains a few mistakes in grammar, punctuation, and spelling.</p> <p>Story mostly flows well and makes sense.</p> <p>Story includes a main character with a description of who he/she is.</p> <p>Game includes a goal.</p>	<p>Writing clearly conveys the story beats and intent of the game.</p> <p>Story is written in complete sentences and paragraph form.</p> <p>Writing contains correct grammar, punctuation, and spelling.</p> <p>Story flows well and makes sense.</p> <p>Story includes a main character (which could be who the player imagines himself to be while playing) with a strong description of who he/she is.</p> <p>Game includes a clear goal for the mission.❑</p>
-----------------------------------	--	---	---	---



Where does this fall in the SDLC?

- ★ Specification

- ★ Design and implementation

- ★ Validation

- ★ Evolution



What is a storyboard?

- Storyboards are visual representations that aid in the creation process of digital story writing.
- Storyboards lay out images in sequential order to create the flow of the production.
- They can also include technical aspects and explanations of design.
- A storyboard visually tells the story panel by panel, kind of like a comic book or slideshow presentation.



Why a storyboard?

- Creating a storyboard will help you plan your video game out shot by shot.
- You can make changes to your storyboard before you start animating and coding, instead of changing your mind later.
- You will also be able to talk about your game and show your storyboard to other people to get feedback on your overall ideas, and less about the technical coding side.
- This makes it ideal to pitch to a team of executives, potential investors, etc.




What makes a storyboard?

- **A storyboard conveys the following information:**
 - What characters are in the frame, and how are they moving/interacting with the game?
 - What are the characters saying/doing to each other, if anything?
 - How much time has passed between the last frame of the storyboard and the current one?
 - Where the “camera” is in the scene? Close or far away? Is the camera moving, still, following the player, or triggered by something?



Beat board vs. Storyboard

- A beat board is basically the same thing as a storyboard with the exception of having less overall detail.
- It has the idea in around 9-16 images but without all the additional detail of shot type, camera angle and speech/interactions.
- It is perfect for a short pitch as you can get the bulk of your general idea across without having to go into specific details.



Your Storyboard somewhere between a storyboard and a beatbox:



Image

- Action
- Dialog
- Notes

Far too short example from Dr BC:



Dialog:

- ✧ Oracle: To win this game you must build a boat and cross the ocean to the land of Cees. It will be a perilous journey.
- ✧ Start by cutting down trees for your boat

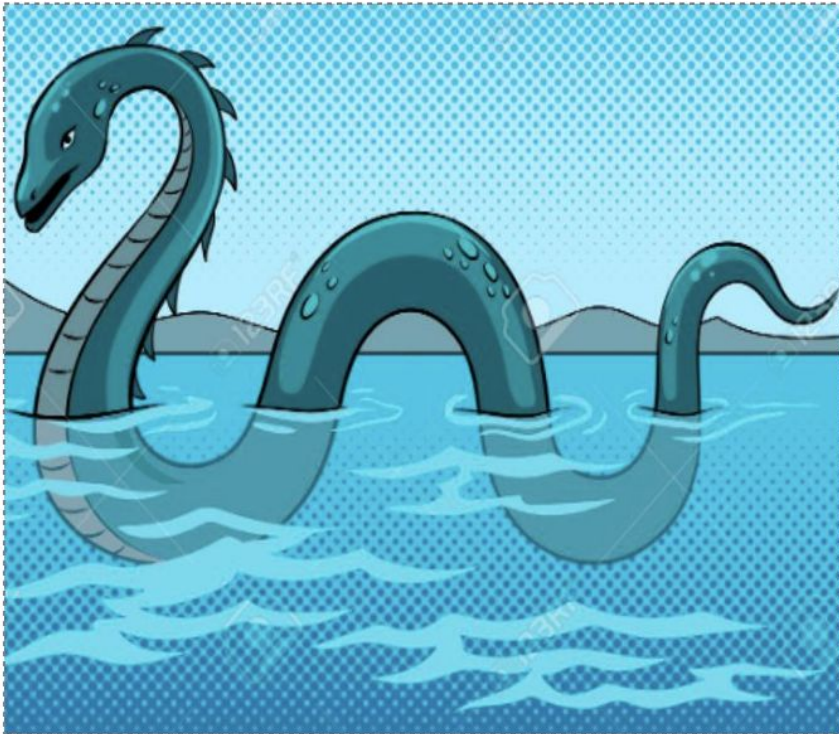
Far too short example from Dr BC:



Action:

- ✧ Player cuts down 15 trees and assembles a boat.

Far too short example from Dr BC:



Action:

- ✧ Sea monster attacks boat.
- ✧ Player dodges fireballs until sea monster is out of breath.

Far too short example from Dr BC:



Action:

- ✧ Player unlocks vault and gets trophy.
- ✧ The end.