Advanced Client-Side JavaScript

COMP2112

Georgian

Tom Tsiliopoulos

- Teaching technology courses to Adult students since 2011.
- Background in software engineering
- Loves Playing and Programming Games
- Currently sharing over 2000 videos on YouTube
- Connect with me at tom.tsiliopoulos@georgiancollege.ca

Coolest thing I've made: An asymmetrical multiplayer VR Game

Favourite game(s): Dungeons and Dragons (TTRPG), Warhammer 40K

Current project: Masters in VR Security: Password Memorability and Interactions

Course Evaluations

Evaluations	Weight	Due
In-Class Exercises (Best 8 / 10) - 4% Each	**32%**	Weekly
Assignment 1	**8%**	Week 5
Assignment 2	**10%**	Week 8
Assignment 3	**10%**	Week 10
Assignment 4	**10%**	Week 13
Mid-Term Test (Practical)	**15%**	Week 7
Final Exam (Practical)	**15%**	Week 14
Total	**100%**	

Course Schedule

Topical Outline (Revised - Subject to Change)

Week	Dates	Topics	Assigned	Due
1	Sep 05, 2022	**Labour Day - No Lecture**		
2	Sep 12, 2022	- Tool Setup - Intro to Git and GitHub - Progressive Enhancement Revisited - JavaScript Review - What is an IIFE? - Using Bootstrap and Font Awesome		ICE 1 (4%)
3	Sep 19, 2022	- Intro to TypeScript - AJAX Revisited / Asynchronous JavaScript - Just Enough jQuery to make AJAX easier - Creating a 5-Page Site with AJAX	Assignment 1	ICE 2 (4%)
4	Sep 26, 2022	- Full CRUD with localStorage - Intro to Model-View-Control (MVC) Design Pattern - Deploying to GitHub Pages		ICE 3 (4%)
5	Oct 03, 2022	- What is a Single-Page Application (SPA) - Creating a SPA with AJAX, localStorage and the history API - Adding Authentication	Assignment 2	ICE 4 (4%) Assignment 1 (8%)
6	Oct 10, 2022	**Thanksgiving - No Lecture**		
7	Oct 17, 2022	**- No Lecture** **- Mid-Term Test** (Practical)	Mid-Term Test	Mid-Term Test (15%)
	Oct 24, 2022	**Mid-Semester Break (No Classes)**		

Course Schedule (continued)

Week	Dates	Topics	Assigned	Due
8	Oct 31, 2022	- Web Frameworks Overview - Intro to React - React Components - React and TypeScript	Assignment 3	ICE 5 (4%) Assignment 2 (10%)
9	Nov 07, 2022	- React Hooks - The React Router - Rebuilding our SPA		ICE 6 (4%)
10	Nov 14, 2022	- Full CRUD with React - The Fetch API and Web Workers - AXIOS - a Better Fetch API?	Assignment 4	ICE 7 (4%) Assignment 3 (10%)
11	Nov 21, 2022	- Authentication with React - Getting Ready to Connect to an API		ICE 8 (4%)
12	Nov 28, 2022	- Intro to NodeJS - NodeJS and TypeScript		ICE 9 (4%)
13	Dec 05, 2022	- Intro to ExpressJS - Express Middleware - Recreating our 5-Page site with Express		ICE 10 (4%) Assignment 4 (10%)
14	Dec 12, 2022	- No Lecture **- Final Test (Practical)**	Final Test	Final Test (15%)

Agenda

- Tool Setup
  - Visual Studio Code
- Version Control
  - Git and GitHub
- Tool Setup
  - Git (local install) and GitHub Account
- Tool Setup
  - **nvm** and **NodeJS**
- Progressive Enhancement Revisited

- JavaScript Demo
  - JavaScript Review
  - What is an IIFE?
  - Using Bootstrap and Font Awesome

Installing Visual Studio Code

Tools – Visual Studio Code

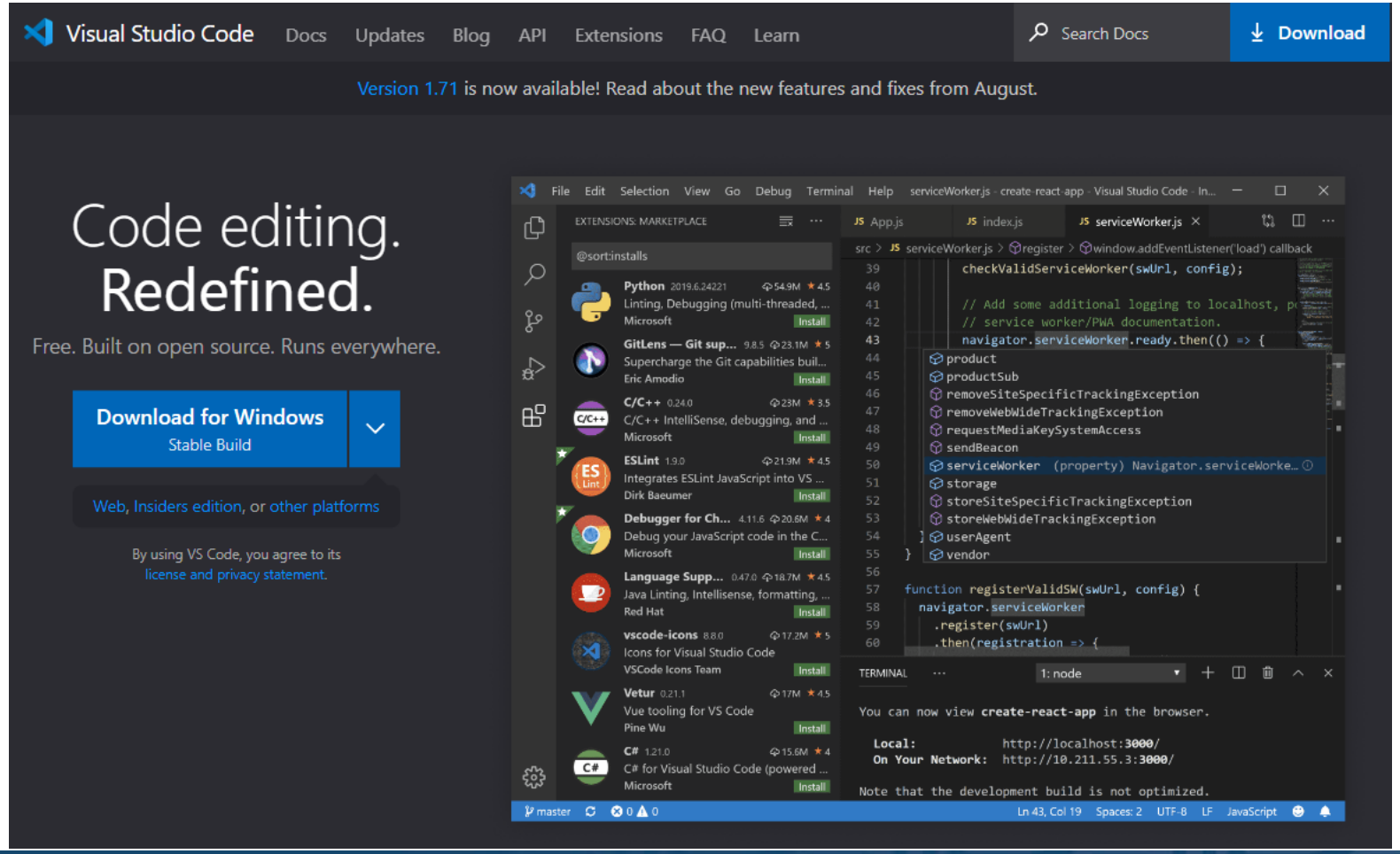
- To get started, you need an **editor** or an IDE to work with **JavaScript** and **TypeScript**.
- For a developer, choosing a code editor is a personal choice.
- And… there are many choices, like **Sublime Text**, **Atom**, **Webstorm**, or even a simple text editor like **Notepad** in Windows.
- Some Developers also choose a full-fledged IDE like Microsoft **Visual Studio (current version is Visual Studio 2022)**.
- Our recommendation is **Visual Studio Code**. First, it is free, lightweight, and open source.
- The features set is driven by Microsoft which is updated monthly with features and enhancements.

Tools – Visual Studio Code -continued

- **Visual Studio Code** is great for working with JavaScript and TypeScript (which we'll be using starting next week).
- It quickly shows warnings and errors as you type code.
- There are easy-to-peek-into definitions for the **functions**.
- It also has a good ecosystem and **extensions** created by developers from around the world.

Tools – Visual Studio Code -continued

- To install Visual Studio Code, download it from https://code.visualstudio.com



Version Control

What is Version Control?

- As the name implies, **Version Control** is about the management of multiple versions of a project.

- To manage a version, each change (addition, edition, or removal) to the files in a project must be **tracked**.

- **Version Contr**ol records each change made to a file (or a group of files) and offers a way to undo or roll back each change.

- For an effective Version Control, you have to use tools called **Version Control Systems**.

What is Version Control? - continued

- **Version Control Systems** help you navigate between changes and quickly let you go back to a previous version when something isn't right.

- One of the most important advantages of using Version Control is **teamwork**.

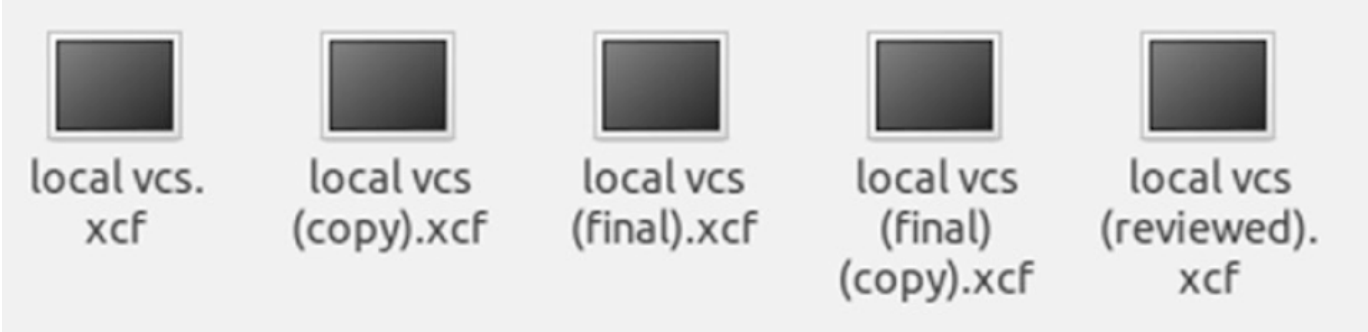
- When more than one person is contributing to a project, **tracking changes** becomes a nightmare, and it greatly increases the probability of overwriting another person's changes.

- With **Version Control**, multiple people can work on their copy of the project (called branches) and only merge those changes to the main project when they (or the other team members) are satisfied with the work.

Why do you need to use Version Control?

- Have you ever worked on a text project or on code that requires you to recall the specific changes made to each file? If yes, how did you manage and control each version?

- Maybe you tried to duplicate and rename the files with suffixes like “review,” “fixed,” or “final”?



- The figure above demonstrates what many people do to deal with file changes.

- As you can see, this has the potential to go out of hand very quickly.

- It is very easy to forget which file is which and what has changed between them.

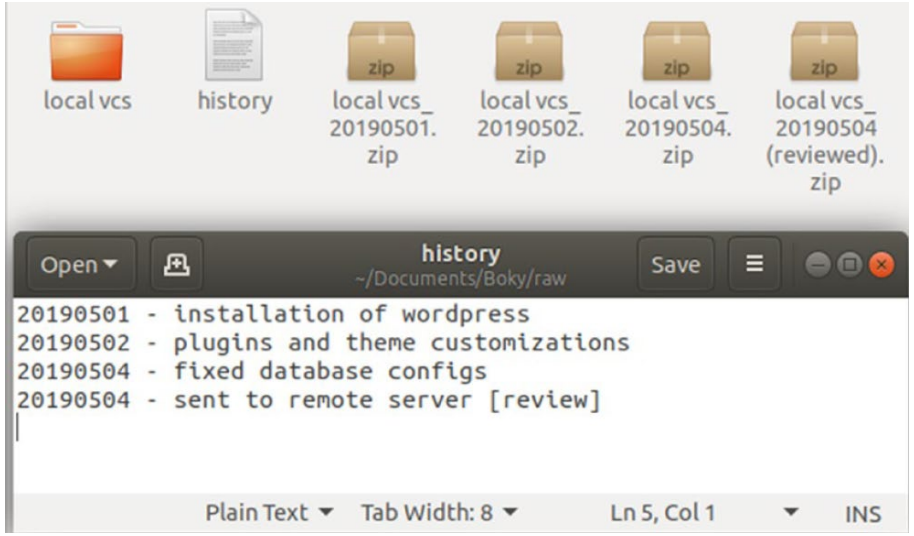
Why do you need to use Version Control? – continued

- To track versions, one idea is to compress the files and append timestamps to the names so that the versions are arranged by date of creation.

- The solution shown in figure above appears to be the perfect system until you realize that even though the versions are tracked, there is no way to know what the contents and descriptions are of each version.

Why do you need to use Version Control? – continued

- To remediate that situation, some developers use a solution like the one showed in the figure below, which is to put the change summary of each version in a separate file.



- That should do it, right? Not quite, you would still need a way to compare each version and every file change.

- There is no way to do this in that system; you just need to memorize everything you did. And if the project gets big, the folder just gets bigger with each version.

Why do you need to use Version Control? – continued

- What happens when **another developer** joins your team?
- Would you email each other the files or versions you edited? Or work on the same **google drive** folder? In the last case, how would you know who is working on which file and what changed?
- And lastly, have you ever felt the need to undo a change you made years ago without breaking everything in the process? An unlimited and all-powerful CTRL-Z?
- What about if your files became **corrupted** or you had a code-base that was recently working but some unknown breaking changes broke your build?
- All those problems are solved by using a Version Control System or VCS.
- A VCS **tracks** each change you made to every file of your project and provides a simple way to compare and roll back those changes.

Why do you need to use Version Control? – continued

- Each **version** of the project is also accompanied by the **description** of the changes made along with a list of the new or edited files.

- When more people join the project, a VCS can show exactly who edited a particular file on a specific time.

- All of that makes you gain precious time for your project because you can focus on writing code instead of spending time tracking each change.

- So to sum up, the main reasons to use a Version Control System are:
  - **Backup** (with the ability to roll back any changes)
  - **Collaboration** (teamwork)
  - **Version Control** (tracking changes)

History of Version Control

- Version control software has seen an evolution of **three generations**:
  - **First Generation:**
    - Version control software utilized a **locking mechanism** on files to allow a change to occur on the file.
    - A file could only be worked on by **one individual** at a given point in time.
    - During this period, **SCCS** (Source Code Control System) and **RCS** (Revision Control System) were the common version control software in use.

History of Version Control - continued

- **Second generation:**
  - Used a **merge-before-commit** mechanism to support concurrent editing of a file by multiple users.
  - To incorporate changes into a file, you would be are required to merge changes made by others to the same file.
  - Once done, you would then proceed to **commit** your change to the file of interest.
  - This generation of software introduced the use of **centralized repositories**.
  - **Subversion** is an example of a second-generation version control system (still in use today).

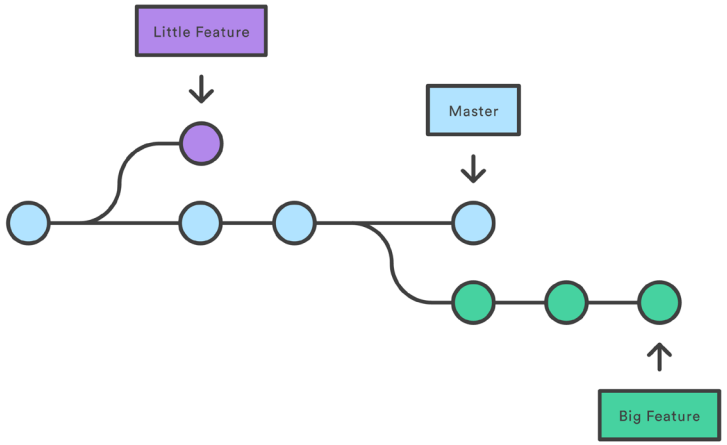
History of Version Control - continued

- **Third generation:**
  - **Decentralized** in nature.
  - Each developer obtains a copy of the **repository**.
  - Changes to the **remote** repository are introduced by way of a **merge**.
  - To allow multiple individuals to work on the same file, a **commit-before-merge** mechanism is used.
  - Here, you make changes to the **local repository,** to incorporate changes made to the **remote repository**, you **commit** the local changes, after which you are able to merge changes made by other individuals.
  - **Git** is a third-generation version control tool.

Common Terminologies

- **Repository (or repo)**
  - A unit of storage and change tracking that represents a directory (or container) whose contents are tracked by **Git**.

- **Branch**
  - A version of a **repository** that represents the current state of the set of files that constitute a repository.
  - In a repository, there exists a default or **main branch** (AKA: the “master branch”) that represents the **single source of truth**.



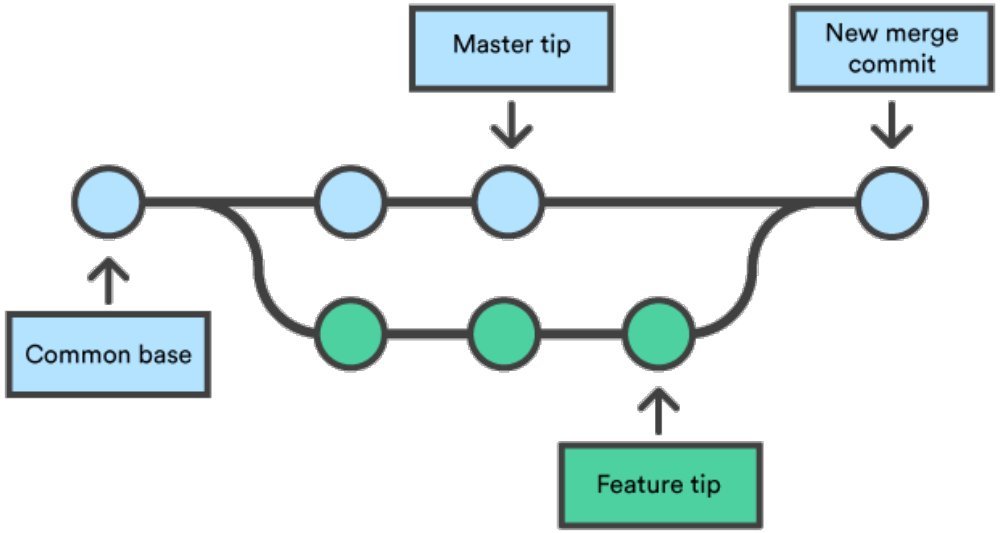
Common Terminologies – continued

- **Commit**
  - This is an entry into Git's history that represents a **change** made to a set of files at a given point in time (think of this as a ”snapshot” of your project)
  - A commit references the files that have been **added** to the index and **updates** the HEAD to point to the new state of the branch.

- **HEAD**
  - A reference to the **most recent commit** on a branch. The most recent commit is commonly referred to as the **tip** of the branch.

Common Terminologies – continued

- **Merge**
  - The process of incorporating changes from one branch to another.



Git Basic Workflow

1. **Create a project**, typically in a folder on your computer.

2. Tell your version control system of choice to **track the changes** of your project/folder.

3. Each time your project is in a working state, or you're going to walk away from it, tell your version control system of choice to **save it as the next version**.

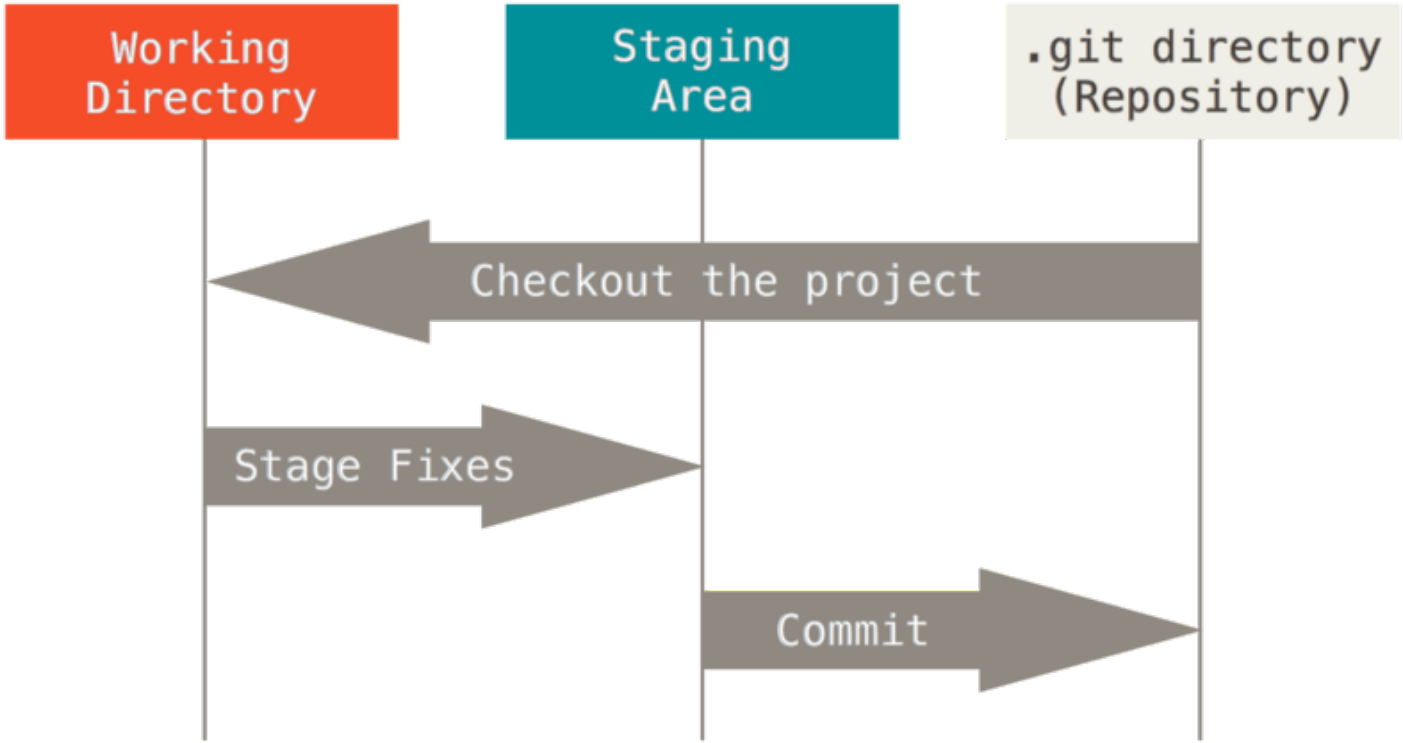
4. If you ever need to go back to a previous version, you can ask your version control system to **revert** to whichever previous version you need.

The three stages of making a commit

- Git has three main states that your files can reside in: **modified**, **staged**, and **committed**:
  - **Modified** means that you have changed the file but have not committed it to your database yet.
  - **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
  - **Committed** means that the data is safely stored in your local database.

The three stages of making a commit - continued

- The following diagram describes the three stages of creating a commit and the commands used to move between them:



The three stages of making a commit - continued

- The **working tree** is a single **checkout** of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

- The **staging area** is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git parlance is the “index”, but the phrase “staging area” works just as well.

- The **Git directory** is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you **clone** a repository from another computer.

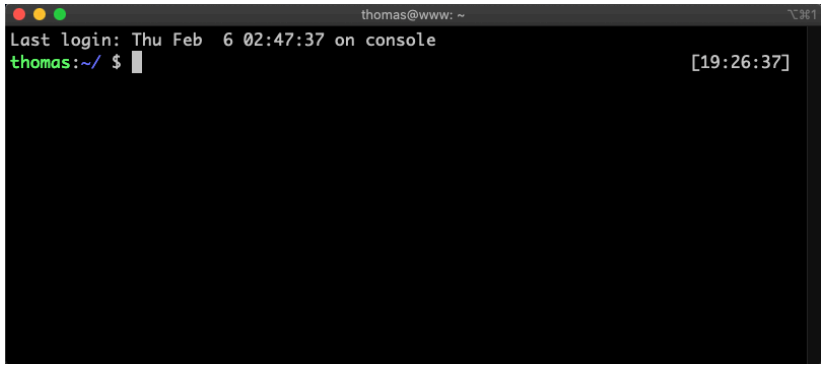
The Command Line

- There are a lot of different ways to use Git.
- There are the original command-line tools, and there are many **graphical user interfaces** of varying capabilities.
- The command line is the only place you can run **all** Git commands — most of the GUIs implement only a **partial subset** of Git functionality for simplicity.
- If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true.
- We'll go over various GUI options later in this lesson

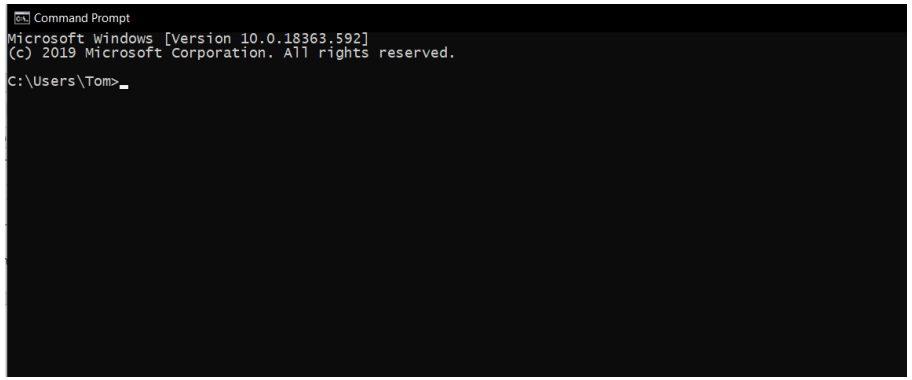
The Command Line – continued

- While your choice of graphical client is a matter of personal taste, **all** users will have the command-line tools installed and available.

iTerm MacOS



Command Prompt Windows 10



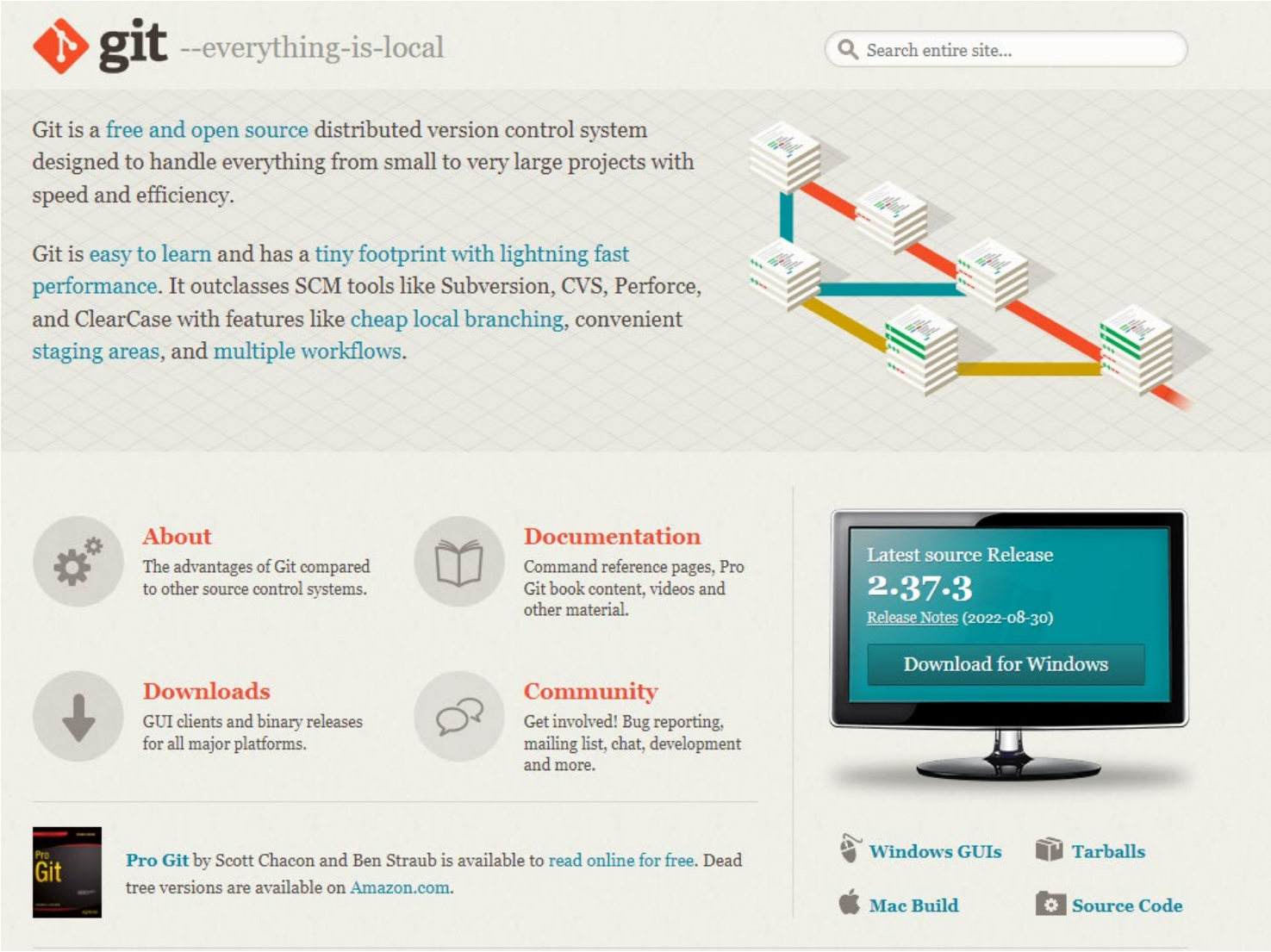
Installing git

Installing Git

- Before you start using Git, you have to make it available on your computer.
- Even if it’s already installed, it’s probably a good idea to update to the latest version.
- You can either install it as a package or via another installer or download the source code and compile it yourself.

Installing Git – continued

- The most official build is available for download on the Git website. Just go to https://git-scm.com/ and the download will start automatically.



Installing Git – continued

- Verify your installation by opening a command prompt.

- Enter the command: `git --version` and press enter

```
Command Prompt
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tsili>git --version
git version 2.37.1.windows.1
```

First Time Git Setup

Your Identity

- The first thing you should do when you install Git is to set your **user name** and **email address**.

- This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating.

- Open a command prompt and enter your information:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

- The email address you enter should be the same one that you will use to register for a new GitHub account.

Git Basics

- **Step 1.** Navigate to your project folder on your computer
- **Step 2.** Initialize your local git repo by typing **`git init`**

```
thomas@www: ~
Last login: Thu Feb  6 22:30:17 on ttys000
thomas:~/ $ git init                                    [22:31:30]
```

- **Step 3.** add all your files to the **staging area** by typing **`git add .`**

```
thomas@www: ~
thomas:~/ $ git add .                                   [22:34:51]
```

Git Basics - continued

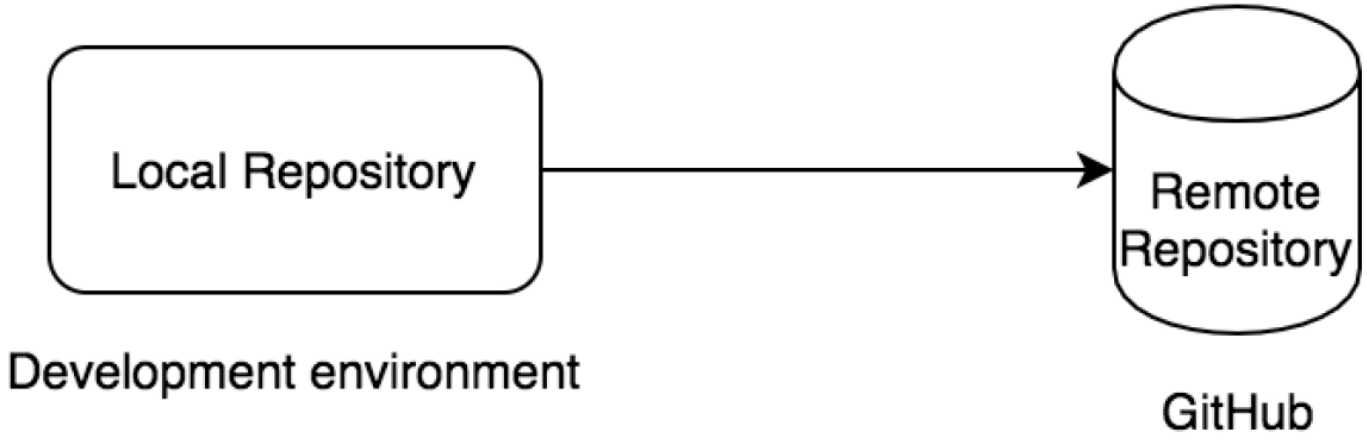
❖ **Step 4.** Create your first commit by typing **`git commit –m “initial commit”`**

```
thomas@www: ~
thomas:~/ $ git commit -m "initial commit"                    [22:34:51]
```

Navigating GitHub

- Version control with Git takes on a **distributed nature**.
- The code resides on each **local computer** where the code base is being worked on, as well as on a **central remote point** where every individual who wishes to work on the code base can obtain it.
- **GitHub** is one such **central remote point**.

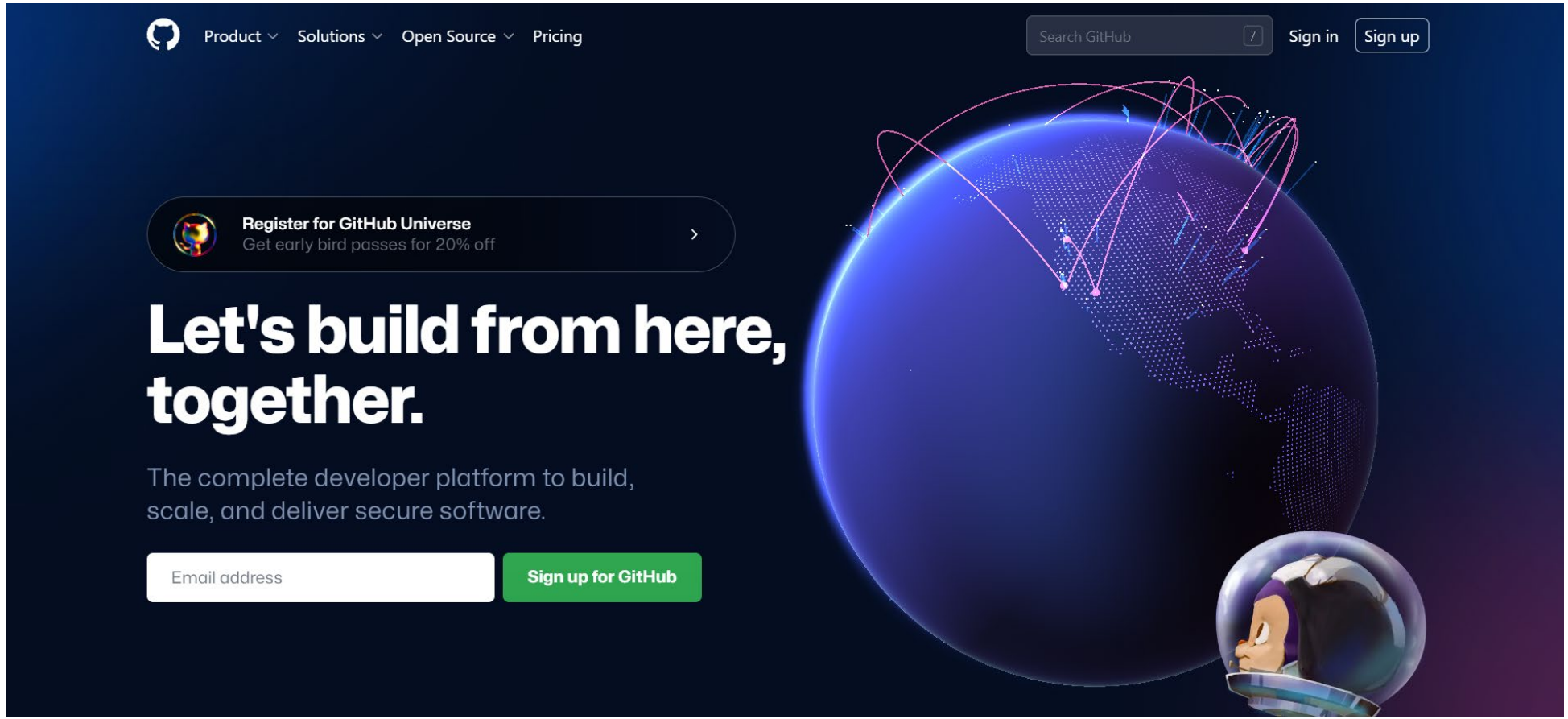
- **GitHub** hosts repositories and enables users to obtain, alter, and integrate changes to a code base through Git:



Setting Up a GitHub Account

Setting Up a GitHub Account

- Navigate to https://github.com/. Enter your user details, as shown in the following screenshot, and press the Sign up for GitHub button:



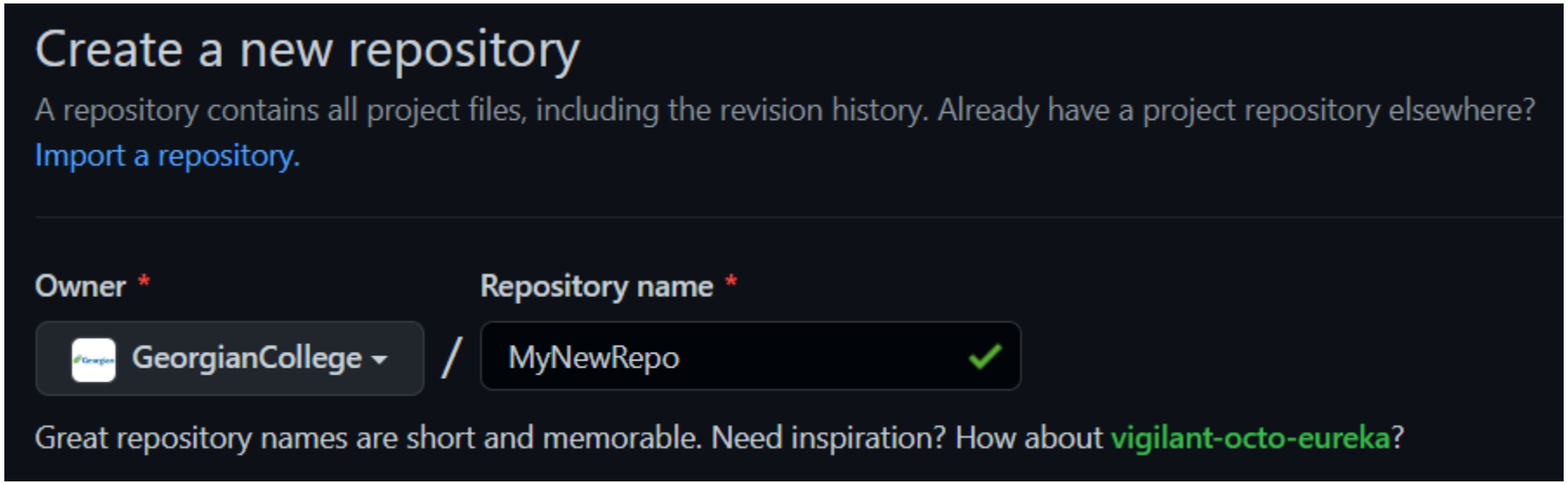
Creating your first repo on GitHub

- **Step 1:** On GitHub, you can start creating your repo by either clicking on the New button:

- Or the **+** symbol next to your avatar at the top right corner of the page

Creating your first repo on GitHub – continued

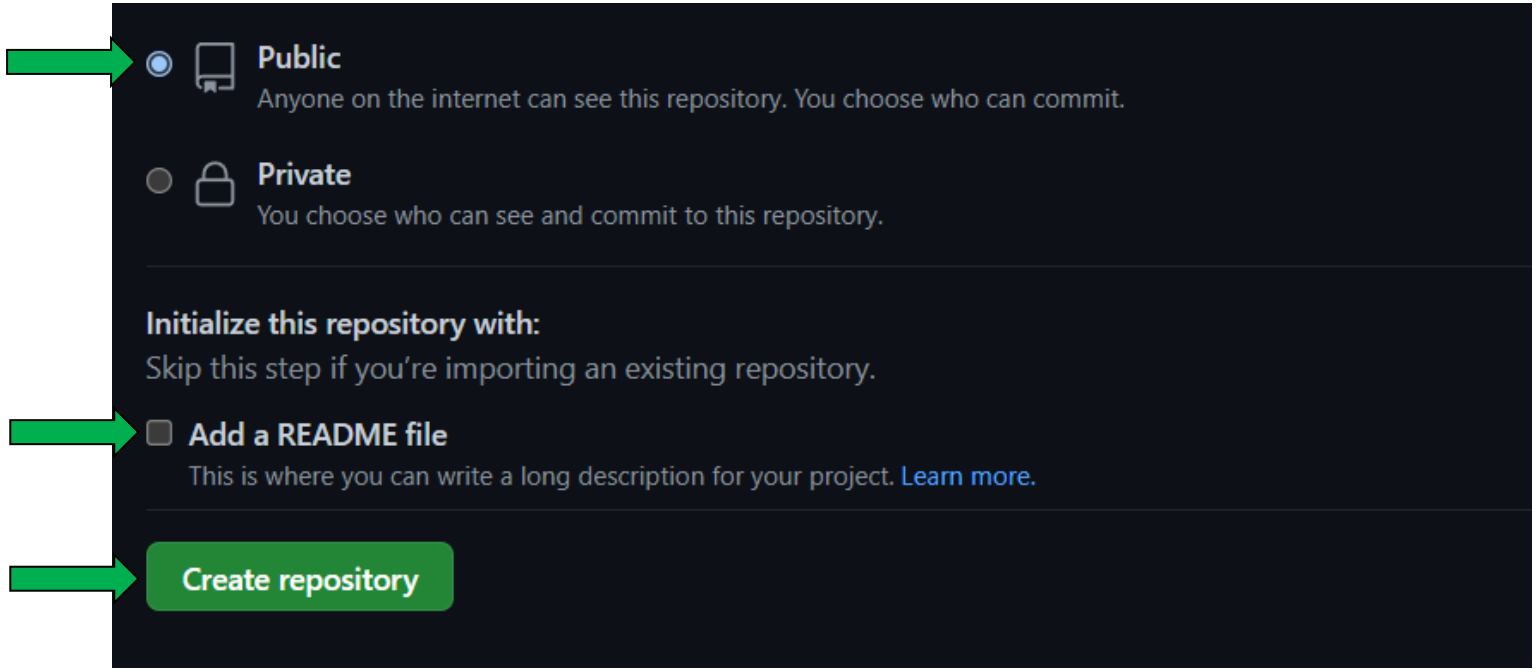
- **Step 2:** Select a **repository name** next to your account or team name. The name of the repo must be **unique** for your account:



Creating your first repo on GitHub – continued

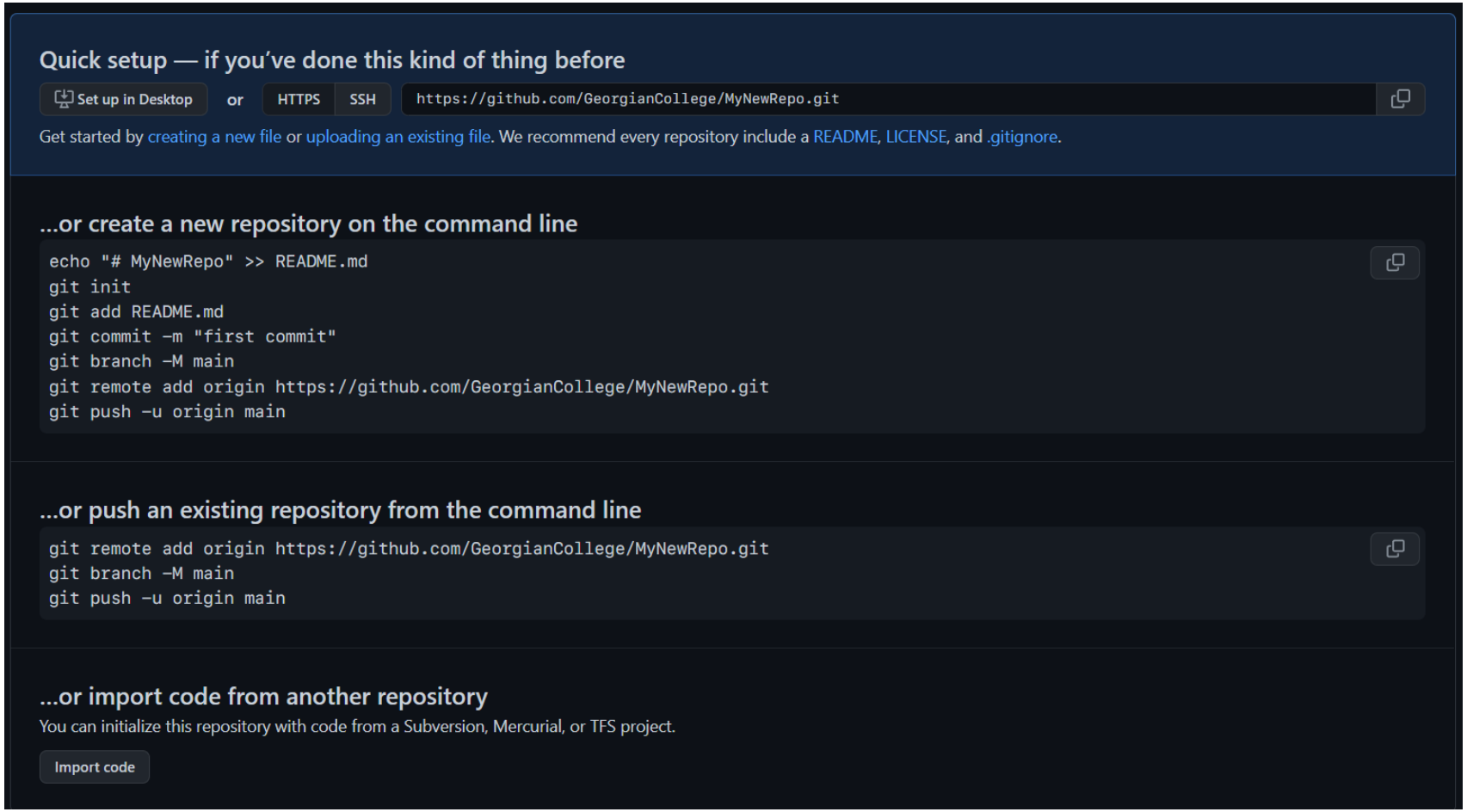
- **Step 3:** You can leave the repo as **public** for now. Also, you can leave initialize this repository with a README **unchecked**.

- **Step 4:** Click the **Create repository** button.



Connecting your local repo to your GitHub repo

- GitHub will create the new repo and display a page with the following information:



Connecting your local repo to your GitHub repo - continued

- To link your **GitHub repo** with the **local repo** you created earlier you need to copy the following three lines of code:

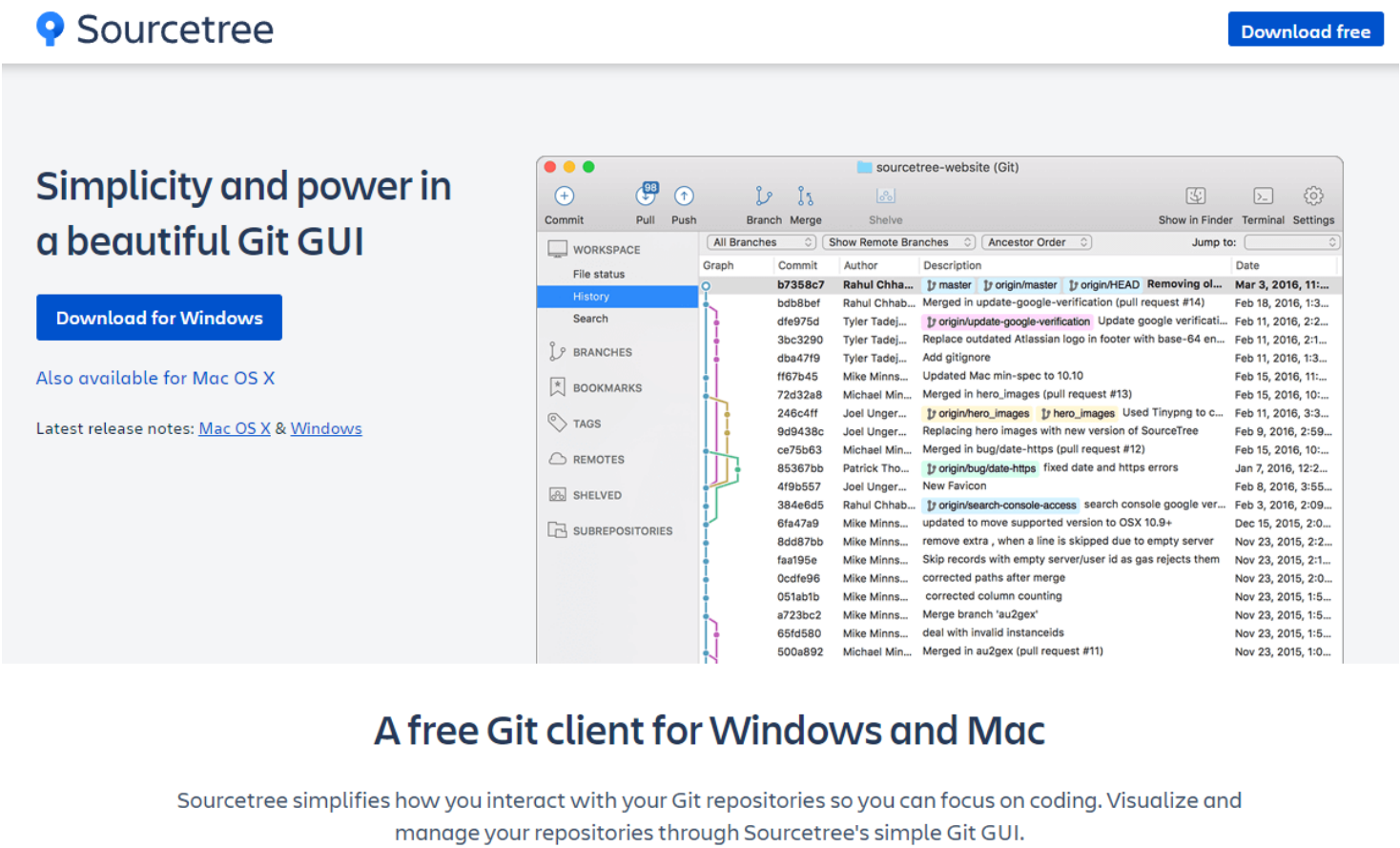
...or push an existing repository from the command line

```
git remote add origin https://github.com/GeorgianCollege/MyNewRepo.git
git branch -M main
git push -u origin main
```

- Then paste into your Command Line window

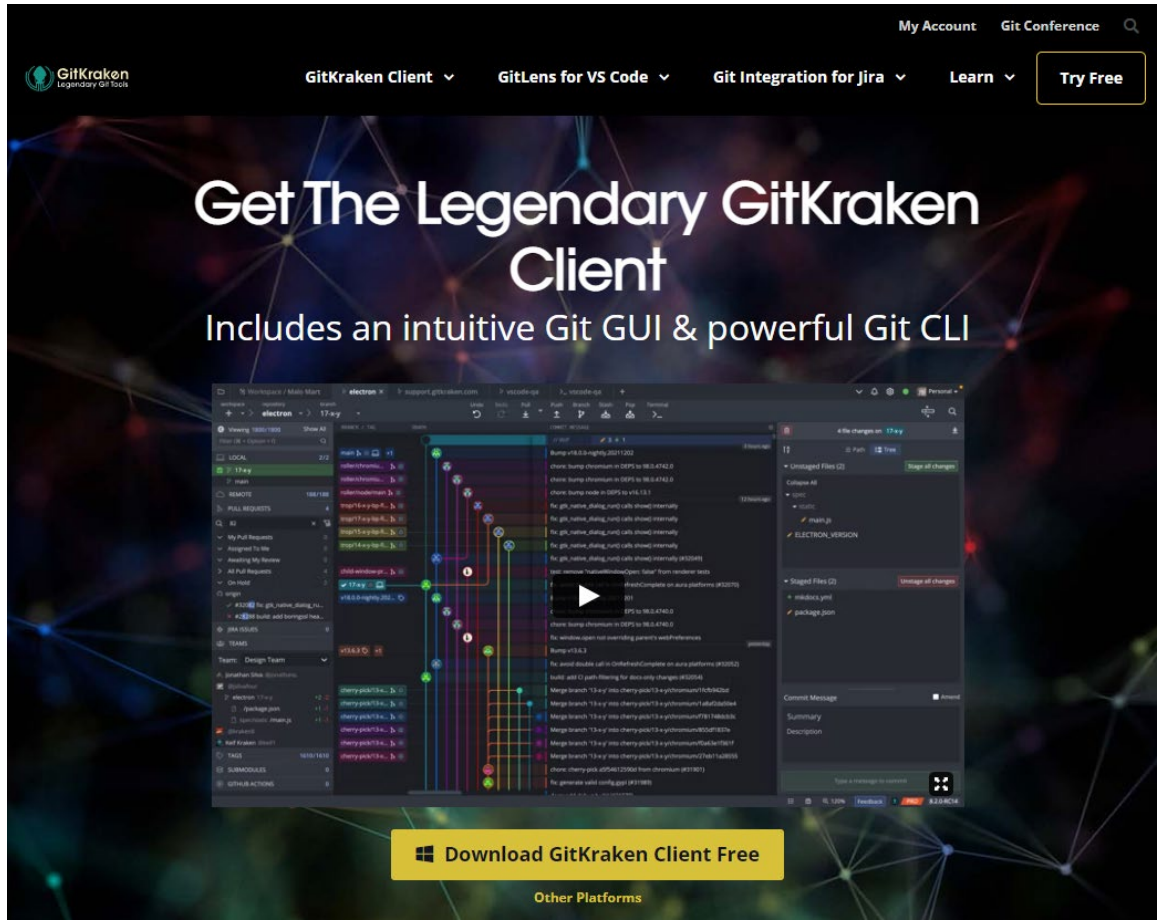
Common GUI Clients

- Sourcetree: https://www.sourcetreeapp.com/



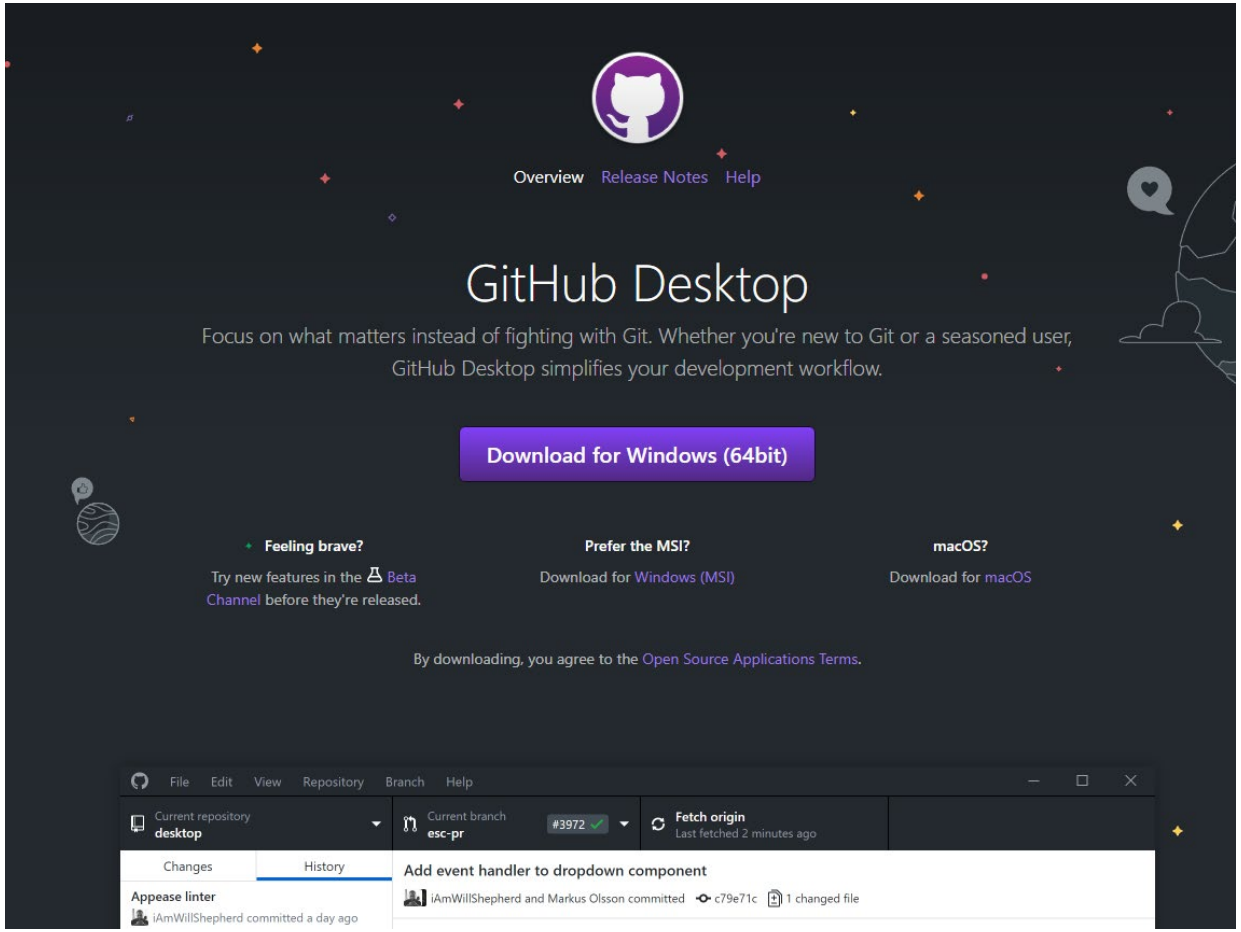
Common GUI Clients – continued

- GitKraken: https://www.gitkraken.com/



Common GUI Clients – continued

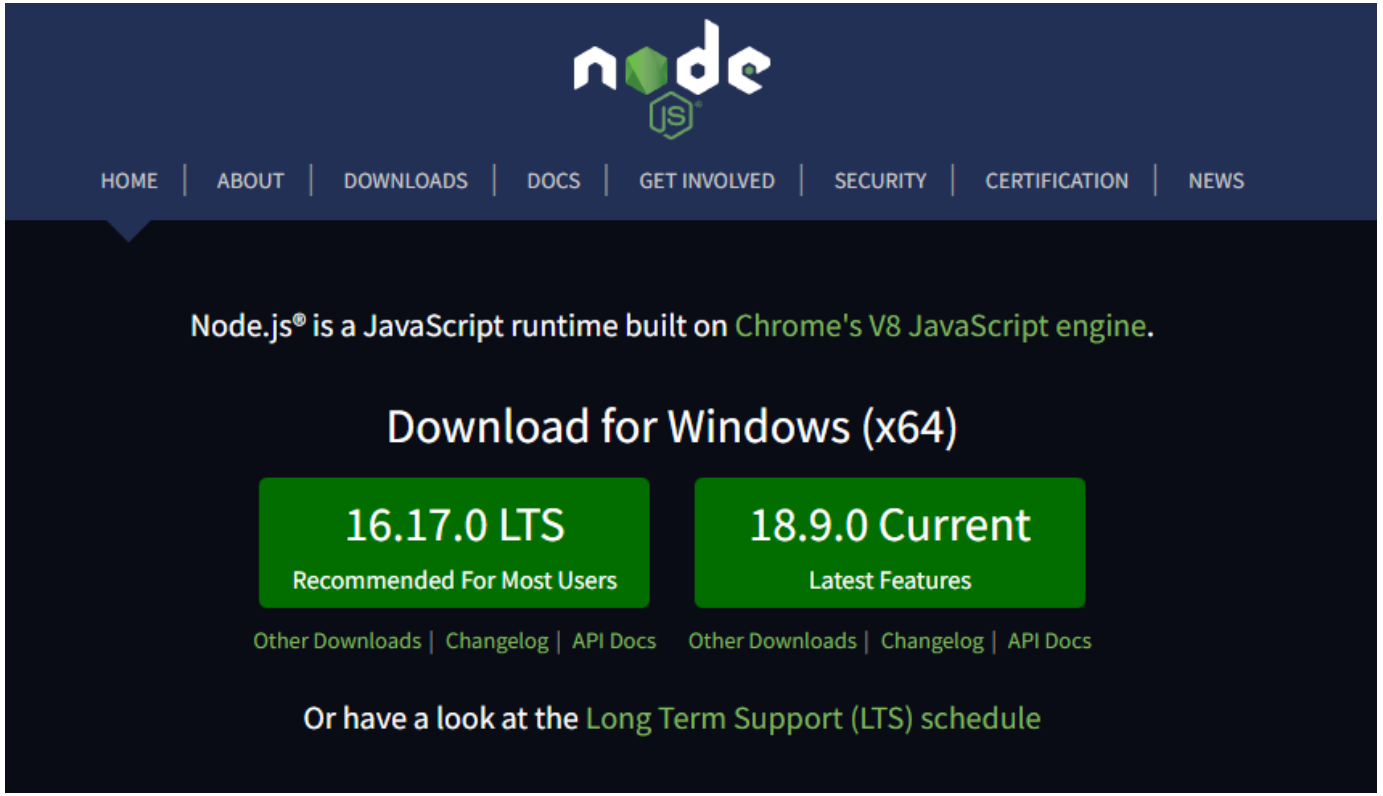
- GitHub Desktop: https://desktop.github.com/



Installing nvm and NodeJS

Installing nvm and NodeJS

- This semester we will be using **NodeJS** and **npm** (the node package manager) to acquire JavaScript frameworks and other software packages created by 3rd party developers (https://nodejs.org/en/)



Installing nvm and NodeJS – continued

- Downloading and installing NodeJS ***directly*** for Windows and MacOS is an option.
- However, **I don't recommend it**.
- The better method is to download and install the node version manager (**nvm**) and use **nvm** to install **NodeJS**

Installing nvm and NodeJS – continued

❖ Installing **nvm** for Windows: https://github.com/coreybutler/nvm-windows

▾ Assets 9

nvm-noinstall.zip	3.64 MB	Dec 15, 2021
nvm-noinstall.zip.checksum.txt	34 Bytes	Dec 15, 2021
nvm-setup.exe	4.64 MB	Apr 27, 2022
nvm-setup.zip	4.14 MB	Dec 15, 2021
nvm-setup.zip.checksum.txt	34 Bytes	Dec 15, 2021
nvm-update.zip	3.45 MB	Dec 15, 2021
nvm-update.zip.checksum.txt	34 Bytes	Dec 15, 2021

Installing nvm and NodeJS – continued

- Use `nvm install latest` to install the latest version of NodeJS

- Confirm your version of **nvm** and **NodeJS**

```
Command Prompt

C:\Users\tsili>nvm version
1.1.9

C:\Users\tsili>node --version
v18.9.0
```

Installing nvm and NodeJS – continued

- For MacOS installations, **homebrew** is the best method to install **nvm**

- See the following link for a “how to”: https://tecadmin.net/install-nvm-macos-with-homebrew/

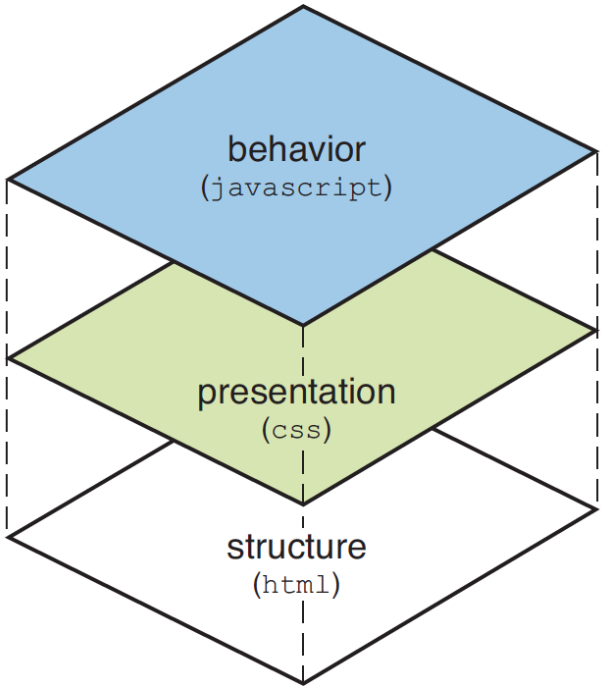
Progressive Enhancement (Revisited)

Progressive Enhancement (Revisited)

- **Progressive enhancement** is the fundamental base for all front-end development.
- At its most basic level, it is creating a **functional separation** between HTML, CSS, and JavaScript.
- Progressive enhancement is a **layered approach** to Web design, where focus is put on **content**, the **user**, and **accessibility**.
- The first step is keeping your HTML, CSS, and JavaScript separated into three “layers”.
- We refer to these three “layers” a **structure**, **presentation**, and **behaviour**
- It is a bottom-up or inside-out building model for a **website** or **application**.

Progressive Enhancement (Revisited) - continued

- You first focus on the **content** and mark it up with semantic and meaningful HTML – this is the first layer, **structure**.
- After the content is properly marked up, we can move onto layer two, **presentation**. On the presentation layer, we deal with CSS.
- The third layer of progressive enhancement, **behaviour**, we deal with last.
- This is where we will be spending a lot of time because this is where the **JavaScript** lives.



And now…the JavaScript Demo