

# Version Control, Git, and GitHub

D2K Course Staff

Rice University

February 6th, 2019

# Intro to Version Control

- ▶ What is version control?
- ▶ Why do we use version control?
- ▶ How will we be applying it to our projects?

## Example

Say you are building a software product, but have a lot of different ideas for potential versions, features, analyses that you want to try out. How do we manage different versions of the same product? How do we keep track of changes over time? How do we keep ourselves from descending into chaos?

# What is Version Control?

- ▶ Version control is a system for keeping track of changes in records, files, code, etc. over time
- ▶ The most primitive version control method is to save different copies of your work locally with different names or timestamps every time a change is made (**BAD IDEA**)
- ▶ How do you collaborate with other people? What if multiple people make different changes at the same time (see below)? How do you reconcile everyone's work? How do you communicate changes?



Figure 1: Versions over Time

# What is Version Control?

- ▶ To answer these questions, developers first created **Centralized Version Control** systems to house the codebase on a central server called a *repository*
- ▶ These present potential problems since the code is only hosted on a single server
  - ▶ More error-prone since each commit updates entire codebase
  - ▶ Impossible to work if the single server goes down

## Centralized version control

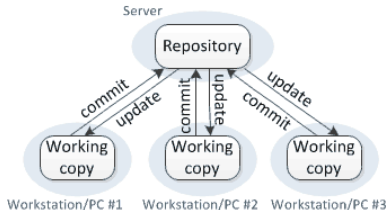


Figure 2: Centralized Version Control Structure

# What is Version Control?

- ▶ To address these issues and others, developers created what are known as **Distributed Version Control** systems
- ▶ These host the codebase or *repository* across multiple servers
- ▶ These are advantageous to centralized systems because they
  - ▶ create full backups of the repository across multiple servers/computers
  - ▶ allow for different people to work in different ways at the same time within the same project

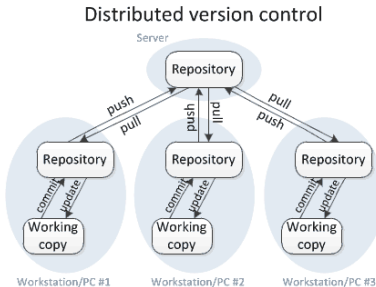


Figure 3: Distributed Version Control Structure

# Why do we use Version Control?

- ▶ You will need to work together!
- ▶ You will make mistakes!
- ▶ You will need to do different things at the same time!

## Bottom Line

Using version control systems is important and is how anyone who writes code as part of a team functions in the real world. It may be confusing at first, but it is better to deal with the initial friction than to realize you have messed up your entire codebase the night before it is due.

# How will we be applying it to our projects?



For your teams this semester, you will be using a code hosting platform called GitHub. GitHub is a free version control tool that is based on the distributed version control system Git. GitHub allows you to track your progress and collaborate on projects from anywhere.

# How to use GitHub

- ▶ Step 1: Create your repository (i.e. codebase)
- ▶ Step 2: Clone your repository to your local computer
- ▶ Step 3: Create a branch
- ▶ Step 4: Make and commit changes to your branch
- ▶ Step 5: Open a pull request
- ▶ Step 6: Merge your pull request

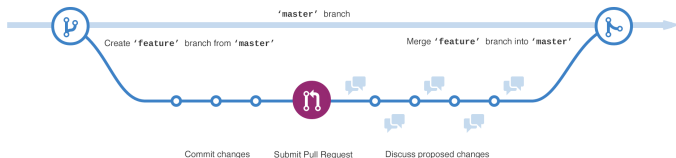


Figure 4: GitHub Workflow



# Step 1: Create your repository

- ▶ Create a new repository on github.com by clicking on the + sign in the top right corner of your screen and selecting **New repository**
- ▶ Select **Private** repository (This is important!)
- ▶ Name your repository, write a short description, and initialize your repository with a README.md file before clicking **Create repository**


## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 benochocinco

Repository name \*

hello\_world 

Great repository names are short and memorable. Need inspiration? How about [scaling-disco](#).

Description (optional)

This is an example for the D2K Learning Lab

☐ Public

Anyone can see this repository. You choose who can commit.

☒ Private

You choose who can see and commit to this repository.

☒ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

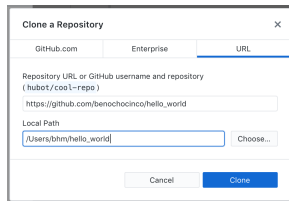
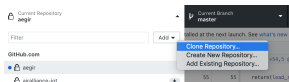
Add .gitignore: None

Add a license: None 

Create repository

## Step 2: Clone your repository

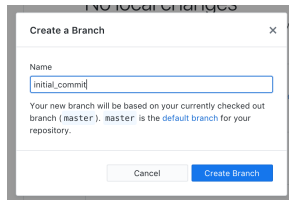
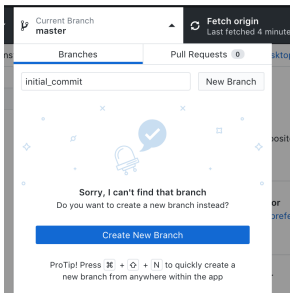
- ▶ Open GitHub desktop and click the top right corner where it says “Current Repository”, click “Add”, and then select “Clone Repository”
- ▶ Copy the github.com/user/repository URL as the Repository URL and select the local path you would like to use (where you want your repository to live on your local computer)



**Note:** you can use the command line if you prefer, but not required

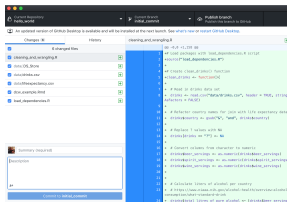
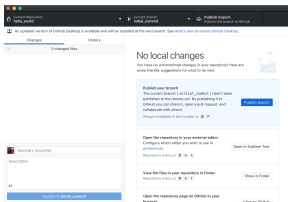
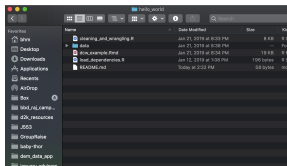
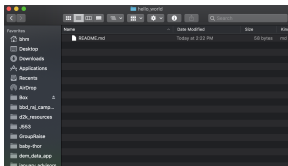
## Step 3: Create a branch

- ▶ Whenever you decide to start a new task or write new code you should create a new branch
- ▶ It is best practice to always start a new branch based on the master branch so that you are not building on top of outdated code
- ▶ The work you do on a branch will exist separately from the master branch until you *commit*, *push*, *pull* and *merge* (more on that later)



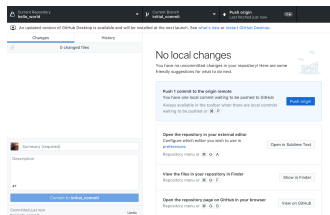
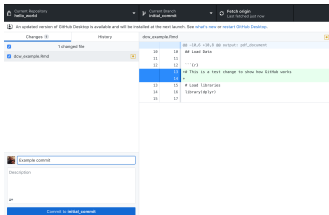
## Step 4: Make and commit changes to your branch

- ▶ Any change made in your local repository will be reflected in the GitHub Desktop interface
- ▶ Changes can be new files or additional lines of code added to old files



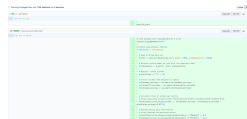
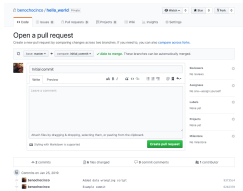
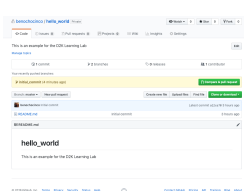
## Step 4: Make and commit changes to your branch

- ▶ Once you have made a change you are happy with, commit the change to “save” your progress locally
- ▶ Once you have complete your modularized task, you push your work to origin (the first push of a branch “publishes” it)
- ▶ After pushing your work, it is saved to the central repository on the GitHub website



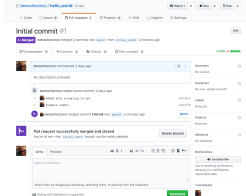
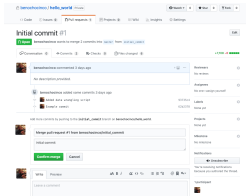
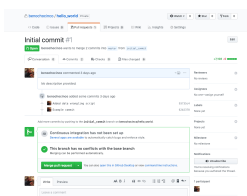
# Step 5: Open a pull request

- ▶ After pushing to origin, you are prompted to make a pull request for your work
- ▶ Making a pull request is making a request to merge your branch of code back into the master branch, or main codebase
- ▶ Before merging your pull request, all team members should review the changes and understand how new changes will affect the dependency structure
- ▶ Comment what changes you made on the pull request (important if you need to go back and review)



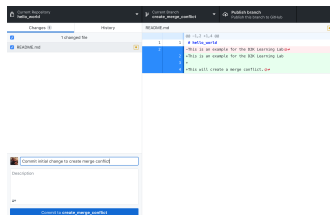
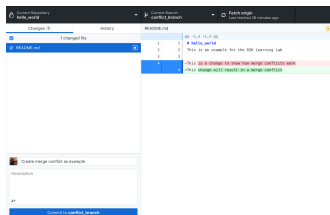
## Step 6: Merge your pull request

- ▶ Once you have opened a pull request you can merge it back into the master branch if there are no conflicts
- ▶ Merge conflicts occur when competing changes are made to the same line of a file (this could include deletion of the line)
- ▶ Merge conflicts are easy to avoid if you communicate with your teammates and always create new branches from the master branch when starting a new modularized task



# Resolving Merge Conflicts

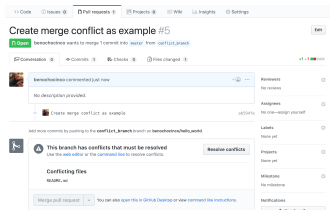
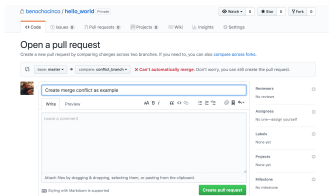
- ▶ Merge conflicts **will** happen at some point this semester, **DON'T FREAK OUT!**
- ▶ Most likely caused by someone committing, pushing, pulling and merging a change to the same files or lines you have changed on your branch
- ▶ Example:
  - ▶ Added a line of code on the **create\_merge\_conflict** branch then committed, pushed, pulled, merged with master branch
  - ▶ Changed same line of code on outdated **conflict\_branch** branch then committed and pushed





# Resolving Merge Conflicts

- ▶ After pushing the new branch that creates the conflict, you can still open a pull request to review the code
- ▶ Once the new pull request has been opened, click on **Resolve conflicts** to review the conflict and reconcile the differences



# Resolving Merge Conflicts

- ▶ The changes on the base branch (HEAD branch) will be reflected on the line after <<<<<<< HEAD and before =====
- ▶ The changes to the new feature branch which created the conflict will be represented after ===== and before >>>>>>> BRANCH – NAME
- ▶ To resolve the conflict, keep the change you want to be in your master branch and get rid of the <<<<<<< HEAD, =====, >>>>>>> BRANCH – NAME conflict markers



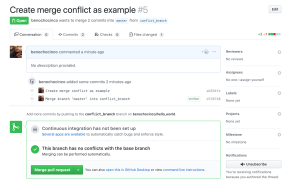
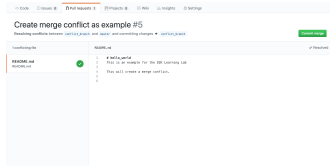
The screenshot shows a web-based interface for resolving a merge conflict. At the top, it says "Create merge conflict as example #5". Below this, it indicates the conflict is between "conflict\_branch" and "main" with conflicting changes in "conflict\_branch". The "Transferring file" section shows a file named "README.md" with a conflict. The conflict markers are: <<<<<<< HEAD, =====, and >>>>>>> conflict\_branch. The content between the markers shows a change from "I will be world" to "I will be world for the new learning lab".



The screenshot shows a web-based interface for resolving a merge conflict, similar to the one on the left. It also says "Create merge conflict as example #5". The conflict is between "conflict\_branch" and "main" in the file "conflict\_branch". The "Transferring file" section shows "README.md" with a conflict. The conflict markers are: <<<<<<< HEAD, =====, and >>>>>>> conflict\_branch. The content between the markers shows a change from "I will be world" to "I will be world for the new learning lab".

# Resolving Merge Conflicts

- ▶ After removing the conflict markers, you will be prompted to commit merge, do this
- ▶ Now you will be able to merge your original pull request without conflict
- ▶ Now that we have witnessed how much of a pain it is to resolve conflicts, we will do a better job in the future of not working from outdated branches!



# Overview

- ▶ Step 1: Create your repository (i.e. codebase)
- ▶ Step 2: Clone your repository to your local computer
- ▶ Step 3: Create a branch
- ▶ Step 4: Make and commit changes to your branch
- ▶ Step 5: Open a pull request
- ▶ Step 5.5: Resolve merge conflicts (if you have any)
- ▶ Step 6: Merge your pull request

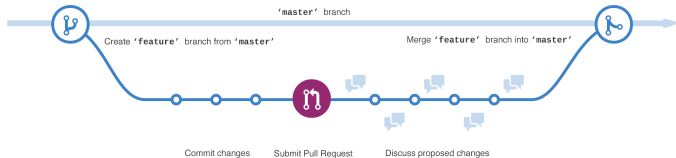


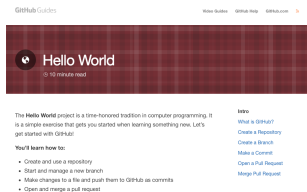
Figure 5: GitHub Workflow

## Overview: Branching Process

- ▶ Step 1: Create new branch from *master* for each new task
- ▶ Step 2: Make and commit your changes to the code until you have completed all of your sub-tasks for your modularized objective
- ▶ Step 3: Push the changes on your branch to GitHub
- ▶ Step 4: Open a pull request on [github.com/your-username/your-repo](https://github.com/your-username/your-repo)
- ▶ Step 5: Review all changes in codebase as a team and make sure there are no conflicts
- ▶ Step 6: Merge your pull request back into the master branch to update the codebase

# Now it's your turn!

- ▶ We will follow the GitHub tutorial at [guides.github.com/activities/hello-world/](https://guides.github.com/activities/hello-world/) together!



- ▶ Make sure you have set up your GitHub account and have downloaded and installed GitHub Desktop before starting
- ▶ *Note:* You can use the command line like they do in the tutorial, but we will be working with GitHub Desktop
- ▶ Additional documentation on how to use GitHub Desktop can be found here: [help.github.com/desktop/guides/](https://help.github.com/desktop/guides/)
- ▶ Additional documentation on how the GitHub workflow works can be found here: [guides.github.com/introduction/flow/](https://guides.github.com/introduction/flow/)