Working with Git Bash

JSC270 Lab 4: Part I

Matthew Edwards

mr.edwards@utoronto.ca

Feb 3, 2021

Last Week: Git

Recall last week, we learned about **version control**

- **Git** is a version control system used to track changes in sets of files (called repositories, or *repos*)
- It was originally designed for computer programmers to collaborate during software development
- Today, almost all data science teams (in both academia and industry) use it to track their work
- The system includes a small (but very powerful) set of commands for editing and tracking files, including:
  - `push`
  - `pull`
  - `add`
  - `commit`
  - `clone`
  - `branch`

Last Week: GitHub

If the goal is collaboration, these file collections (repos) should be stored somewhere easily accessible

- **GitHub** is an internet hosting service designed to store Git repositories. It allows for easy interaction with Git
- It is not the only service that does this (e.g. GitLab, BitBucket, SorceForge), but it is by far the most popular (and free)
- Git and GitHub can track multiple branches (versions) of the same repository, so **contributors can make changes in parallel** without affecting the `main/master` branch
  - This is particularly useful if you're experimenting (ie you're not sure if your changes will actually work)

A Basic Git Workflow

Here is a basic step-by-step process for managing repos:

1. You have a new repo you'd like to work with. Either you've created it yourself, or it is someone else's **public** repository.
   - Recall from last week: Only you (the repo creator) and contributors you allow can see a private repo.
2. You **Fork** or **Clone** that repo to your local machine (e.g into your local `gitrepos` folder). More on this later.
3. You edit files in the repo (or create new ones). If you're experimenting, you would first create a new **branch**, so your changes do not affect `main`.

A Basic Git Workflow

4. You submit any changes you've made so they can be tracked by Git. This is a two-part process:

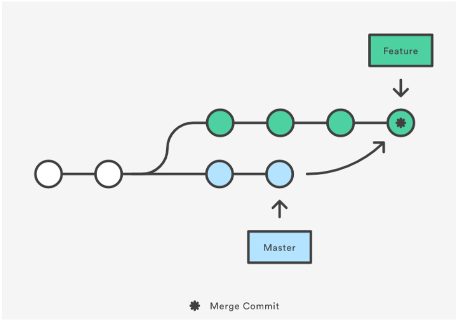
   i) **Use the `add` command** to move changes to the staging area. This tells Git that you have changes to be included in your next commit.

   ii) **Use the `commit` command** to save these changes to your local repo. This effectively takes a 'snapshot' of your files.

Note:
Committed snapshots are safe - Git will never change them unless you tell it to.

Every commit you make is recorded and tracked, so you may revert back to past versions of your project at any time.

A Basic Git Workflow



5. (Optional) If you were experimenting on a different branch, you can `merge` your newer version back into the main branch.

6. You use the `push` command to push your commit(s) from your local repo to the remote repo (on GitHub). Again, more on this later.

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

A Basic Git Workflow

7. If other contributors are following a similar process, it's likely they will `push` changes to the remote (GitHub) repo that you do not have locally. **Use the *pull* command** to to update your local repository to the most current version.

Note:
In this course, you will be the only ones editing your assignment files (although we need contributor access to read them), so steps 5 and 7 may not be necessary.

But in practice, there could be many contributors. So a good rule is to **always `pull` before you make changes**.

What is Bash?

The **B**orn-**a**gain **Sh**ell, or **Bash**, is a **command line interface**

Exercise 1: Installing Bash

Navigating Your File System

Creating, Removing, and Renaming Directories

Exercise 2: Creating a `gitrepos` Folder

Forking or Cloning Existing Repos