GitHub

Sajid Fadlelseed
P.h.D. Student

School of Engineering & Computing
University of Hertfordshire

June 14, 2023

Overview

What is version control

- keeps track of your creative output
- it tracks what is changed
- it tracks who made the changes
- it tracks why changes were made
- everyone needs it
  - developers / designers
  - writers / producers
  - artists / composers
- you can use it as part of team
- you can use it by yourself

Why version control

- history: change-by-change log of your work
- fearless experimintation: easy to create a "sandbox" to try new things
- synchronisation - easy to keep team members always up-to-date
- accountability - know who made each change and why
- conflict detection - keep the build clean every time

You already do version control
even if you don't use software

- `save as` . . . for backup individual files
- duplicate or compress directories
- sometimes you save them it on Dropbox, GoogleDrive etc . . .

Advantages

- backups - every verion is kept around
- change tracking - commit messages let you know why things changed
- rollback to previous versions - similar to undo
- labeling significant changes - tags/labels identify the state of the source that matches each release

Short history on version control

- centralised
  - 1972 Sourec Code Control System (SCCS)
    - ★ Unix only
  - 1982 Revision Control System (RCS)
    - ★ cross-platform,
  - 1986 Centerlised Version Control (CVS)
    - ★ first central repository
    - ★ file focused
  - 2000 SubVersioN (SVN)
    - ★ non text files
    - ★ track directory structure
- distributed
  - 2005 Git:
    - ★ created by Linus Torvalds after BitKeeper went commercial only
    - ★ broadly used in conjuncation with GitHub which offers free hosting for open-source projects

Centralised vs Distributed

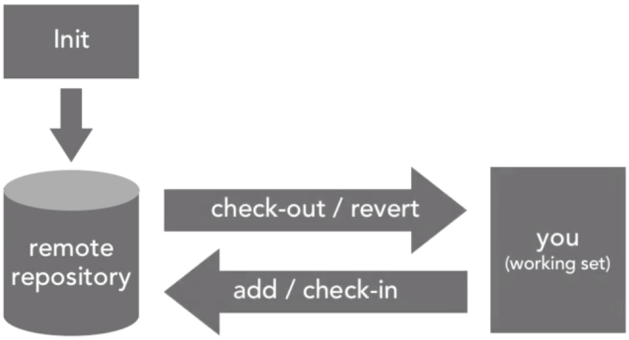

Figure: Centralised Version Control System

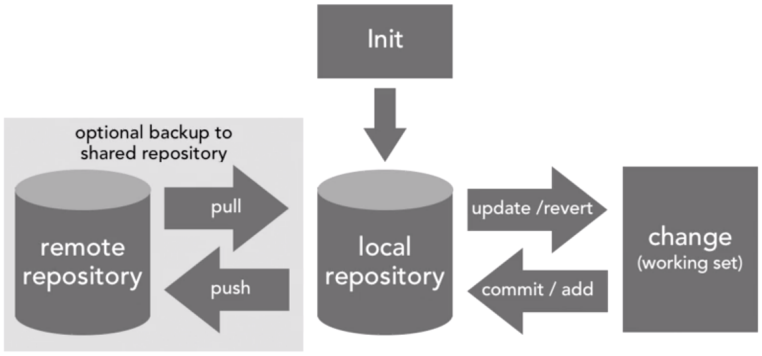

Figure: Distributed Version Control System

First time Git configuration

- setup your name
  ```
  git config --global user.name "your-name"
  ```
- setup your email
  ```
  git config --global user.email "your-email"
  ```
- makes sure that Git output is colored
  ```
  git config --global color.ui auto
  ```

Link Git with your text editor

- Sublime Text Setup
  Unix/Linux

  git config –global core.editor "'/Applications/Sublime Text 2.app/Contents/SharedSupport/bin/subl' -n -w"

  Windows

  git config –global core.editor "'C:/Program Files/Sublime Text 2/sublime_text.exe' -n -w"
- Atom Editor Setup
  Windows/Unix/Linux

  git config –global core.editor "atom –wait"
- VSCode Setup
  Windows/Unix/Linux

Create a Git repository

- Create brand new repository from scratch on your computer using command

```
git init
```

- Copy an existing repository from somewhere else to your computer using command

```
git clone
```

- Check the current state of the repository

```
git status
```

Create a Git repository cont.

- Running the `git init` command sets up all of the necessary files and directories.
- All of these files are stored in a directory called `.git` this directory is the "repo"! here is where git records the file changes and keeps track of everything!

WARNING

Don't edit any file inside the `.git` directory. This is the heart of the repository. If you change file names and/or file content, Git will probably lose track of the files that you're keeping in the repo, and you could lose a lot of work! It's okay to look at those files though, but don't edit or delete them.

Create a Git repository cont.

`.git` contents are:

description file this file is only used by the GitWeb program, so we can ignore it

hooks directory this is where we could place client-side or server-side scripts that we can use to hook into Git's different lifecycle events (Remember, other than the "hooks" directory, you shouldn't mess with any of the content in here. The "hooks" directory can be used to hook into different parts or events of Git's workflow, but that's a more advanced topic that we won't touch it in this workshop)

info directory contains the global excludes file

objects directory this directory will store all of the commits we make

refs directory this directory holds pointers to commits (basically the "branches" and "tags")

Useful Links:

https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain

https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

Clone an existed repo

- Create brand new repository from scratch on your computer using command

```
git clone <path-to-repository-to-clone>
```

- Running the git clone command will:
  1. takes the path to an existing repository;
  2. by default will create a directory with the same name as the repository that's being cloned;
  3. can be given a second argument that will be used as the name of the directory;
  4. will create the new repository inside of the current working directory.

Useful Links:

Cloning an Existing Repository:
https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository#Cloning-an-Existing-Repositoryhttps://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
git clone docs: https://git-scm.com/docs/git-clone

git clone Tutorial: https://www.atlassian.com/git/tutorials/setting-up-a-repository

Status update

- To figure out what's going on with a repository using command

```
git status
```

- At this point, we have two Git repositories:
  1. the empty one that we created with the `git init` command;
  2. the one we cloned with the `git clone` command.

Knowing the status of a Git repository is extremely important.

Review existed repository

- Display information about the existing commits using command

`git log`

- Display information about a commit using command

`git show`

Review existed repository cont.

- Running the git log command displays all of the commits of a repository. By default, this command displays:
  - SHA
  - author
  - date
  - message
- Git uses the command line pager *Less*, to page through all of the information. The important keys for Less are:
  - to scroll down by a line, use `j` or ↓
  - to scroll up by a line, use `k` or ↑
  - to scroll down by a page, use the spacebar or the Page Down button
  - to scroll up by a page, use `b` or the Page Up button
  - to quit, use `q`

Review existed repository cont.

`git log --oneline`

- lists one commit per line
- shows the first 7 characters of the commit's SHA
- shows the commit's message

`git log --stat`

- displays the file(s) that have been modified
- displays the number of lines that have been added/removed
- displays a summary line with the total number of modified files and lines that have been added/removed

Review existed repository cont.

Show a specific commit info using command

```
git show <SHA>
```

- the commit
- the author
- the date
- the commit message
- the patch information

Tagging, branching and merging

- Add tags to specific commits e.g. this is the beta release

  `git tag`
- Allows multiple lines of development

  `git branch`
- Switch between different branches and tags

  `git checkout`
- Combine changes on different branches

  `git merge`

The tag command

Used to add a marker on a specific commit and the tag does not move around as new commits are added. This command will:

- add a tag to the most recent commit

```
git tag -a beta
```

- add a tag to a specific commit if a SHA is passed

```
git tag -a beta a87984
```

The branch command

Used to interact with Git branches

- Lets you make changes that do not affect the master branch
- It can be used to
  - list all branch names in the repository

    ```
    git branch
    ```
  - create new branches

    ```
    git branch <new_branch_name>
    ```
  - delete branches

    ```
    git branch  -d <branch_name>
    ```
  - delete remote branches

    ```
    git push <remote_name> --delete <branch_name>
    ```

The checkout command

```
git checkout <branch_name>
```

It's important to understand how this command works. Running this command will:

- remove all files and directories from the Working Directory that Git is tracking (files that Git tracks are stored in the repository, so nothing is lost)
- go into the repository and pull out all of the files and directories of the commit that the branch points to

Switch and create branch in one command

```
git checkout -b <new_branch_name>
```

The merge command

Combining branches together is called merging.

```
git merge <name-of-branch-to-merge-in>
```

WARNING

It's very important to know which branch you're on when you're about to merge branches together. Remember that making a merge makes a commit. If you make a merge on the wrong branch, use this command to undo the merge:

```
git reset --hard HEAD^
```

Undoing changes

- Changing the last commit

```
git commit --amend
```

- Reverting a commit

```
git revert <SHA-of-commit-to-revert>
```

Working with Remotes

Difference between git and GitHub

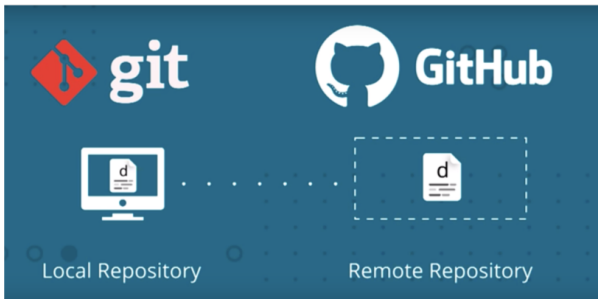

Figure: Git vs GitHub

- Git: tool to manage the repository
  - Git is a version control tool
- GitHub: Hosting version control repositories
  - GitHub is a service to host Git projects

Commands for remote

- Create & Manage remote repository

  `git remote`
- To send changes up to the remote

  `git push`
- Retrieve updates from the remote

  `git pull`

Add remote-repo to your local-repo

- To add a connection to a new remote repository

```
git remote add origin <repo-on-GitHub>
```

- To see the details about a connection to a remote.

```
git remote -v
```

Pull changes from a remote

- To pull changes from remote repo and merge to local repo implicitly

```
git pull origin master
```

- Do `pull` when:
  - Working with a team, and a co-worker has pushed new changes to the remote.
  - Working on the same project but from different computers.
- When git pull is run, the following things happen:
  - The commit(s) on the remote branch are copied to the local repository and the local tracking branch (origin/master) is moved to point to the most recent commit
  - The local tracking branch (origin/master) is merged into the local branch (master)

Fetch changes from a remote

- To fetch changes from remote repo but merge to local repo explicitly

```
git fetch origin master
```

- When git fetch is run, the following things happen:
  - The commit(s) on the remote branch are copied to the local repository and the local tracking branch (origin/master) is moved to point to the most recent commit
  - The local tracking branch (origin/master) is not merged into the local branch (master)

Working on another developer repo

- Forking a repo
- Reviewing another developer changes
- Knowing what changes to make

There fork command

There is no fork command!

- If a repository doesn't belong to your account, then it means you don't have permission to modify it;
- If you fork the repository to your own account then you will have full control over that repository.

There pull command

Pull request is a request for the source repository to pull in your commits and merge them with their project. To create a pull request, a couple of things need to happen:

1. you must fork the source repository
2. clone your fork down to your machine
3. make some commits
4. push the commits back to your fork
5. create a new pull request and choose the branch that has your new commits

Stay in sync with source project

If you want to keep up-to-date with the Repository, GitHub offers a convenient way to keep track of repositories - it lets you star repositories

- You can go to https://github.com/stars to list out and filter all of the repositories that you have starred.
- If you need to keep up with a project's changes and want to be notified of when things change, GitHub offers a "Watch" feature.

Stay in sync with source project cont.

Add connection to the remote source repository.

```
git remote add upstream <source-repo-on-GitHub>
```

Origin vs Upstream:

- One thing that can be a tiny bit confusing right now is the difference between the origin and upstream. What might be confusing is that origin does not refer to the source repository that we forked from. Instead, it's pointing to our forked repository. So even though it has the word origin is not actually the original repository.
- Remember that the names origin and upstream are just the default or de facto names that are used. If it's clearer for you to name your origin remote mine and the upstream remote source-repo:

```
git remote rename origin mine
git remote rename upstream source-repo
```

Stay in sync with source project cont.

- Get the changes from upstream remote repository

```
git fetch upstream master
```

- Then merge it with local repo

```
git merge upstream/master
```

- Then push it your remote repo account

```
git push origin master
```