Github Notes

Rohan Parmar

2023-02-18

Index:

1 - Core Concepts:

Git: Decentralised / Distributed Control System.

- Can still use a central server
- Scales well -> Created for Linux Kernel Project (Linus Torvalds)
- Local operations -> Super fast
- Open Source
- Active Community

Repository: Collection of files managed by Git.

- Files/ structure lives in **.git** folder

Commits and Files:

- Updates are saved in snapshots called *Commits*
- Git commits *files* **not** *folders*
- Use *dummy files* to "save" empty folders
- Commits are saved on a timeline, known as *the branch*
- At least one branch in a repo, called "master"
- Can create additional branches!

Github: Git hosting repository service provided by Github, Inc

- Most popular git hosting platform
- Comprehensive, strong features
- Claim to fame: Free unlimited public repositories
- Many open source projects!
- Extra features: issue tracking, web pages, etc.

2 - Basics - Foundational Commands:

Getting Started / New Repo:

Creating a new git repository locally in terminal:

```
# Print working directory
pwd
# List all files including hidden ones, and in long format
ls -al
# Create new directory -> We will use this as our base folder
mkdir projects
# Move into the newly created directory
cd projects/
# Create new repository within the projects folder that will be our current project folder
git init demo
# Move into the demo folder
cd demo/

# At this point, the terminal should have a (master) tag on the command line, indicating
# that you are on the default branch on the git repo, as follows:
(master) demo $
```

Git States:

Working Directory: All the files and folders for your application, which may or may not be managed by Git. Git is however, aware of those files.

Staging Area: Used to prepare for the next commit. Files are moved from the working directory to here, to the repository.

Repository: All the committed or saved changes to the git repository, stored in the ".git" folder. Everything here is part of Git's history.

Remote: 4th stage of the Git State, it is just the local repository saved on a remote server.

First Commit:

To check out what is happening on the repository, you can use:

```
git status
```

It indicates which branch you are in, what the current state of the repo is.

Now, we should create a READ ME file, which is a Markdown text file used to indicate what the repository is all about.

```
mate README.md
```

The *git status* command will now indicate that we have an untracked file in the repo.

To add the file from the Working Directory to the Staging Area:

```
git add README.md
```

The *git status* command will indicate that we now have changes to be committed.

Committing the staged file:

```
git commit -m "First file in demo repo"
```

The *git status* will now indicate we have nothing to commit.

Starting with an Existing Project:

Process:

- cd to the existing project folder
- use git init with the current folder
- add files to the repo
- commit files

```
cd projects/
git init .
git status
git add .
git commit -m "Initialising the repository"
```

Committing Details with Log and Show:

Git log displays the commit history.

```
(master) demo $ git log
```

Output:

```
commit 2g2vjgv4hgc22456hgvhfc23434hgv5jgv1jhvb5
Author: Rohan Parmar <rohanparmar@git>
Date: Sat Aug 15 00:16:12 2023 -400

  Initialising the repository
```

Anatomy of the log:

Commit ID: used to uniquely identify commits in the repo
Author: Person who made the commit
Date: Date and Time of commit
Message: The commit message

We can get the same information with more details about the diff using "git show"

Express Commit - Combining Commands:

To see which files git is tracking:

```
(master) demo $ git ls-files
```

Add a new file:

```
(master) demo $ touch new.file
```

The new file is not tracked, as proved by git status.

Remove the file:

```
(master) demo $ rm new.file
```

The express commit relies on tracked files to work.

```
(master) demo $ git commit -am "Updating Readme"
```

This stages all changes immediately and commits it.

Backing out changes - Unstaging:

```
(master) demo $ git reset HEAD README.md
```

To discard the changes entirely, revert back to last state in the repo

```
(master) demo $ git checkout --README.md
```

Sometimes, no news is good news

History and Alias:

```
(master) demo $ git log --oneline --graph --decorate --all
```

We can configure a new command for this to make it easier to call

```
(master) demo $ git config --global alias.hist "log --oneline --graph --decorate --all"
```

Checking if the entry made it in:

```
(master) demo $ git config --global --list
```

To use the new command:

```
(master) demo $ git hist
```

Can also pass files!:

```
(master) demo $ git hist --README.md
```

Rename and Delete files:

Rename:

```
(master) demo $ git mv example.txt demo.txt
```

Delete:

```
(master) demo $ git rm demo.txt
```

Exluding Unwanted Files:

In the command line:

```
(master) demo $ mate .gitignore
```

In the file that opens up, write the name of the files you want ignored

```
*.log # all log files
application.log # just this one
```

3 - Advanced Git:

Comparing Differences:

Compare what has changed.

```
(master) demo $ git diff {commit ID} {commit ID}
```

Branching and Merge Types:

Branch = Timeline of Commits.

More accurately, branches are labels we give to commits. Deletion removes label only.

Types of merges:

- Fast Forward Merge: Simplest case, no additional work has been detected on the parent branch. Can be disabled.
- Automatic Merge: Happens when Non-Conflicting Merge is detected, preserves both timelines and merges commit on destination.
- Manual Merge: Automatic merge not possible due to conflicting merge state. Conflicts must be resolved automatically before moving forward.

Special Markers:

Just like "pointers".

HEAD -> Points to Last Commit of Current Branch, and can be moved.

Simple Branching Example:

Creating a new branch and switching to it:

```
(master) demo $ git checkout -b {branch name}
```

To check the difference between the branches:

```
({branch name}) demo $ git diff {branch name} master
```

To integrate the changes on the branch:

- Switch to parent branch
- Merge with previous branch
- Delete previous branch

```
({branch name}) demo $ git checkout master
(master) demo $ git merge {branch name}
(master) demo $ git branch -d {branch name}
```