Git Crash Course for Beginners

GITCRASHCOURSE.MD

Git Crash Course for Beginners

Preface

A good understanding of Git is an incredibly valuable tool for anyone working amongst a group on a single project. At first, learning how to use Git will appear quite complicated and difficult to grasp, but it is actually quite simple and easy to understand.

Git is a version control system that allows multiple developers to contribute to a project simultaneously. It is a command-line application with a set of commands to manipulate commits and branches (explained below). This tutorial will help you get started, and in no time you will be a Git Ninja!

Contents:

How Git Works

Branching and Commits

One of Git's biggest advantages is its branching capabilities. This allows someone to branch off of the master branch (where the production-quality code typically remains) and work on a feature or fix independently of the rest of the project. Once they finish their work, they can merge the branch back into the master branch with the click of a button.

```
FEATURE              o-----o-o---------o-----o
                    /                         \
MASTER    <---o----o------o--o-------------------o------>
```

Feature branches provide an isolated environment for the developer to work. They are also useful for organizing different levels of granularity. For instance, in what is known as the Git-Flow, there are three important branches: master, staging, and develop. Master is the parent branch, representing the state of the project that is released to the public (deployed). The staging branch is where pre-release QA is done. Develop is where developers work on the project. All feature branches are branched from develop.

```
FEATURE1              o-----o-o---------o-----o
                     /                         \
DEVELOP   <-------o-----o---o-----------o-------o------>
                        \               \
STAGING   <---------------o---------------o----o-------->
                          \                   \
MASTER    <------------------o-------------------o------>
```

In the diagram above, commits are represented by `o` in the branches. A commit is an individual piece of work. It's a bit like a milestone--when a small piece of the task is completed.

Workspace, Staging, Local, and Remote

A Git 
repository is also split into four levels. Each level represents a state of the codebase. Each state is isolated from each other, but git commands allow you to transfer data between them.

- The workspace is the folder on your computer. This is where all the work happens.
- The staging area is where changed files are placed before making a commit. It allows you to essentially select only the files you want committed.
- The local is your copy of the repository. No one else will see this except you.
- The remote is the public repository. Everyone sees the same remote.

So, evidently, moving changes from one workspace to another will involve passing from workspace1 > staging1 > local1 > remote > local2 > staging2 > workspace2. It is also important to note that all communication must pass through the remote repository.

Installing Git on your Machine

If you're using UNIX, you're in luck! Git come pre-installed on macOS and on most distributions of Linux. Type `git --version` in the terminal to check if you have Git installed.

Git does not come pre-installed on Windows (big surprise...), so you will need to install it. Git Bash is a great option; it emulates a UNIX shell environment so you don't need to use Windows commands. You can also download a version of Git here that you can use in the Command Prompt. There are also graphical Git clients, a quick google search will show a number of them.

Git Setup (Login)

Let's get you setup to use Git. First, you need to identify yourself. When you first install Git, it's important to set your user name and email address because every commit is signed with this information. In the terminal:

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

Cloning the Repository

If you have already set up your development environment through your IDE using a `Checkout From Version Control` option, then you can skip this section. Otherwise, you will need to clone the repository to your local machine. Fortunately, this is a very easy process.

Open the terminal (or Git Bash if using Windows) and navigate to the folder in which you wish to clone your repository. Now, run the following command: `git clone <REPO CLONE URL>` . You can get the repository clone url from the Github or Gitlab repository. You have the option of cloning with HTTPS or with SSH.

This should create a new child directory for the project sources. From the new project working directory, you are able to run your git commands.

Useful Git Commands

git status

> Shows the status of your current branch. It will tell you which files you have modified, which files are stages for commit and which files are not. It will also let you know if your local branch is behind, or not up to date with, its remote tracking branch. This command is particularly useful in combination with `git add` when you only want to commit certain files.

git branch

> Create a branch, or show a list of local or remote branches. With git branch, you can add options to do branch-related things:
>
> - `git branch new-branch-name` : Create a branch off of the current branch named `new-branch-name`
> - `-d or --delete` : Delete a branch. e.g. `git branch -d feature-branch-name`
> - `-D` : Force delete a branch. This is used when you want to delete a branch that has changes that weren't pushed to the server or merged. e.g. `git branch -D feature-branch-name`
> - `-a or --all` : Display all local and remote-tracking branches.

git fetch

> Fetches branches from the server, along with their history and information. You can also use `git fetch origin` to update your remote-tracking branches.

git pull

> Update your working directory by performing a git fetch, followed by a git merge. This is useful if you want to bring your branch up to date with the remote-tracking branch. You can also use the `--rebase` option to perform a rebase rather than a merge.

git checkout

> This command has a few functions. It can be used to create a new branch off of the current branch, it can be used to checkout a branch from the server, or switch to another branch.
>
> - `git checkout develop` : you will use this a lot. It will switch you from whatever branch you are currently in to the `develop` branch. If you have changes in your current branch, you will need to either stash them or undo your changes before you switch branches.
> - `git checkout branch-on-server` : If you want to work on another persons branch (assuming they already pushed their branch to the remote repository), you can use this command to create a copy of the branch on your local

machine. You will need to do `git fetch` first to grab the information about the branch.

- `git checkout -b new-branch-name` : create a new branch off of the current branch and switch to that branch. If you are in `develop`, this will create a new branch from develop. This command is synonymous to `git branch new-branch-name && git checkout new-branch-name`

git add

You made changes to your branch, and you want to stage your changes for committing. You will need to add them to what is known as the `index`. Here are two ways to do this:

- `git add /filepath/file.f` : this will add the single file.
- `git add .` : this will add all the changed files. Be careful here; make sure you aren't adding unwanted changes by reviewing your changes using `git status`. If you accidentally staged unwanted files, take a look at `git reset` described below.

git commit

You want to make a commit! Awesome! Here's how you do it:

- `git commit` : This will make a commit of the files you have staged using `git add`, and will open a VIM editor (or whichever editor you have configured in your .gitconfig) to enter your commit message.
- `git commit -a` : This will make a commit and add all changed files, and will open an editor to enter your commit message. This is the same as doing `git add . && git commit .`
- `git commit -m "commit message"` : Skip the editor and put your entire message in the command line!
- `git commit -a -m "commit message"` : Combine them all! Fab.

git push

Once you make a commit, you will need to push your changes to the server to make it visible to the world. Once you push, your branch and commit(s) will be visible to others.

- `git push branch-name` or `git push origin HEAD` : Push your branch upstream. You can use either version; using origin HEAD is simply a more verbose way of pushing. It signifies you are pushing the tip (HEAD) your branch to the same name on the remote `origin`.

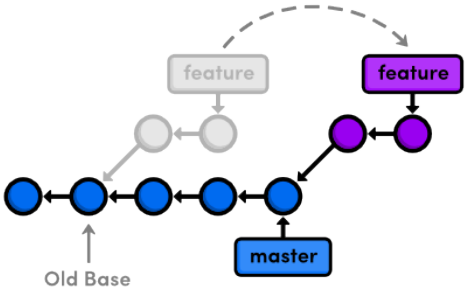
- `git push origin -f branch-name` or `git push origin +branch-name` : ❗ ❗ ❗ **BE CAREFUL: This is a potentially scary command, and you should only use it if you really know what you are doing. It will overwrite the branch you specified on the remote repository with your local copy of the branch.** ❗ ❗ ❗

git log

Show the history of all branches and their commits. Useful for seeing the most recent commit and getting commit hashes.

- `git log` : Show all commits on this branch, along with the commit hash, author, date and message.
- `git log --oneline` : Show a simplified version of the above command, showing only the commit hash and commit message.
- `git log --graph` : Show a graph of the commit history for your branch. (AWESOME)
- `git log --max-count <n>` : Limit the number of commits shown.
- `git log --oneline --graph --max-count 50` : Combined them all! Woo!

git rebase



Rebasing is a bit more advanced, but incredibly useful when you need it. When you perform a rebase, you are changing the base of your branch. In essence, a rebase will look at each commit on your branch and update the code to make it seem like you've been working off the new base all along. Sometimes, a rebase will barf if it encounters a situation in which it tries to update a piece of code that you have already modified. In this case, it doesn't know which version of the code to use, so it leaves it to you to resolve them manually. Although they have similar uses, a rebase differs from a merge in that a rebase updates your branch by tweaking each commit, and merge will update your branch by creating a new commit at the tip of your branch. Usually, there are standards and practices employed in a project or team around which method is preferred. Have a discussion with your team about which workflow they prefer.

git stash

Stashing allows you to save your current unstaged changes and bring your branch back to an unmodified state. When you stash, your changes are pushed onto a stack. This is especially useful if you need to quickly switch to another branch without having to commit your incomplete changes.

- `git stash` : stash your unstaged changes.
- `git stash pop` : unstash your changes.
- `git stash drop` : drop your stashed changes. Careful, you will lose your changes!
- `git stash push <path>` : stash a single file
- `git stash -m <message>` : add a message to your stash

git reset

Git reset is used to unstage files or remove commits. It does so by changing what the tip of the branch (HEAD) points to. Three `trees` are affected by a reset:

- HEAD: the tip of your branch
- Index: the staging area, which is like a proposed commit
- Working Directory: the files currently on your disk There are three reset modes: `--soft` , `--mixed` (default), and `--hard` :
- `--soft` : This is the very first step in any reset mode. For a soft reset,
- `--mixed` :
- `--hard` :
- Undo `git add` :
  - `git add .`
  - `git reset -- file.html` (unstage single file, or)
  - `git reset` (to unstage all)
- Undo most recent commit:
  - `git commit -a -m "commit message"`
  - `git reset --soft HEAD^` (undo commit that is at the tip of the branch)
  - `git commit -a -m "new commit message"`
- Undo a commit permanently:
  - `git reset --hard HEAD~2` (destroy the last two commits, i.e. HEAD and HEAD^)

Further reading

7.7 Git Tools - Reset Demystified

tsehayget commented on May 20

It helps me a lot

buchepalli-ramana commented on Jun 10

It is very useful and clean to understand basic commands

danlo9 commented on Jun 29

This is a great refresher!

alikhantareen commented 17 days ago

Great.