**Git** is the open-source software that actually manages the git commands as a VCS.

**GitHub** is the online hosting provider that can act as a machine connected to our local machine via the internet and host code in a repository for us.

**git init:**

* + This command initializes a Git Repository on your local machine.
  + You only need to run this command once per project.

**git status:**

* + This command will report back the status of your Git repository.
* Upon creating a repository with **git init** you will create a hidden .git file.
* The .git file is a hidden file that manages the versioning of the files inside the Git repository.
* Git inside a Folder/Directory:
  + Upon creating a Git Repository, all the folders/directories inside the top level Git Repository will also be part of that Repository, meaning all the changes are tracked.
* Create Personal Access Tokens (PAT) on Github.com

When using the **git clone** command, reference the PAT.

* Clone Syntax with Personal Access Tokens (PAT):

**git clone** [**https://token@github.com/account/repo.git**](https://token@github.com/account/repo.git)

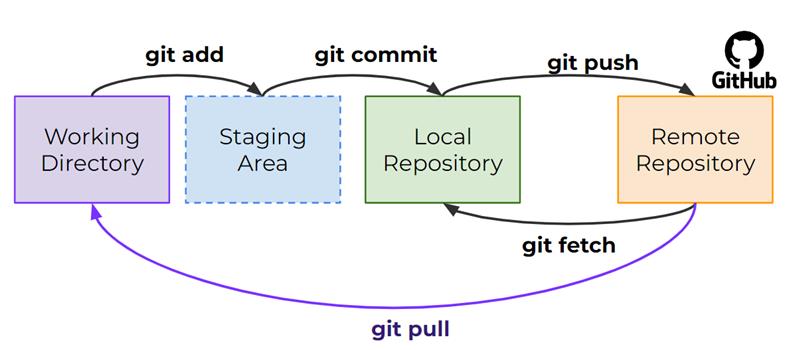
* Clone without Personal Access Tokens (PAT):

**git clone** [**https://github.com/account/repo.git**](https://github.com/account/repo.git)

* Changing code in a Repository to staging area

**git add** .

* Committing these changes
  + **git commit**
* Pushing or Pulling Changes
  + **git push** and **git pull**
* Checking Status, Logs, and Changes
  + **git status, git log, git diff**
* Changing code in a Repository & adding it to staging area
  + **git add**
* Committing these changes
  + **git commit**
* Pushing or Pulling Changes
  + **git push** and **git pull**
* Checking Status, Logs, and Changes
  + **git status, git log, git diff**
* to add a remote branch using the git remote command syntax:
  + **git remote add name** [**https://url.git**](https://url.git/)
* By convention, we call this remote branch the **origin** branch.
  + **git remote add origin** [**https://url.git**](https://url.git/)
* **git remote rename <old> <new>**
* **git remote remove <name>**
* Once we’ve connected to our remote branch on GitHub, we can **push** our code to the remote branch.
* We tell git to push to the remote main/master branch called origin with the command:
  + **git push -u origin main/master**

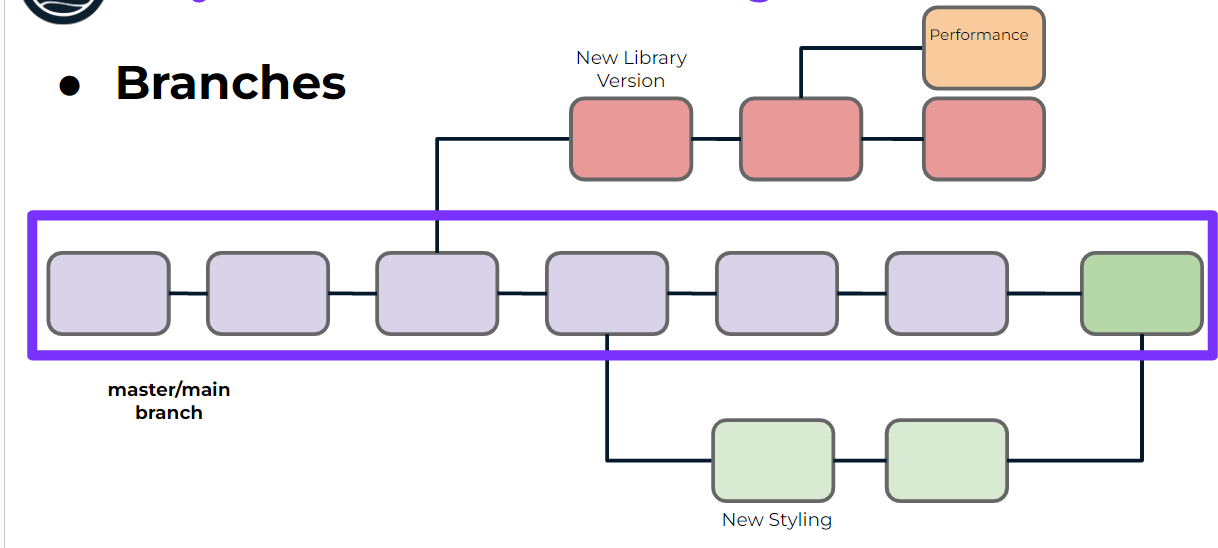


Using **git fetch** will download changes from the GitHub remote repository, however you will not see those changes be part of the files you have in the working directory.

* Fetch allows you to grab additional work done on the remote master branch, without needing to merge it in your working directory files.
* Using **git fetch** makes sense when you’re working with others and want to see what changes they’ve made but aren’t ready to overwrite your own files yet.
* Also if you are simply working by yourself, you may want to just fetch remote changes without overwriting your latest work (later we’ll discover branches are a better way of doing this).
* Using **git pull** makes sense when you want to directly grab changes from the remote branch and directly merge them into your current branch.
* This means you will literally update the files in your working directory to match up and merge with the remote branch.
* If you’re a solo developer working on a single master branch, you often skip using a combination of **git fetch** and **git merge** and go straight for a **git pull**.
* Branches allow us to organize a repository and split it apart so multiple people can work on it or so a solo developer can work on different aspects of a project on a separate work.
* Git Branch Commands:

**git branch**, **git switch**, **git checkout**

* **Branches**
  + A branch represents an independent line of development.
  + Branches serve as an abstraction for the edit/stage/commit process.
  + They are a way to request a brand new working directory, staging area, and project history.
* Branches are just pointers to commits.
* When you create a branch, all Git needs to do is create a new pointer, it doesn’t change the repository in any other way.



o Upon creating a new repo with **git init** you create a new branch called the **master branch** (or **main branch**).

o This is a branch just like any other, but it’s simply the first one created.

*§* *Should the code pushed to the master branch always be in working condition?*

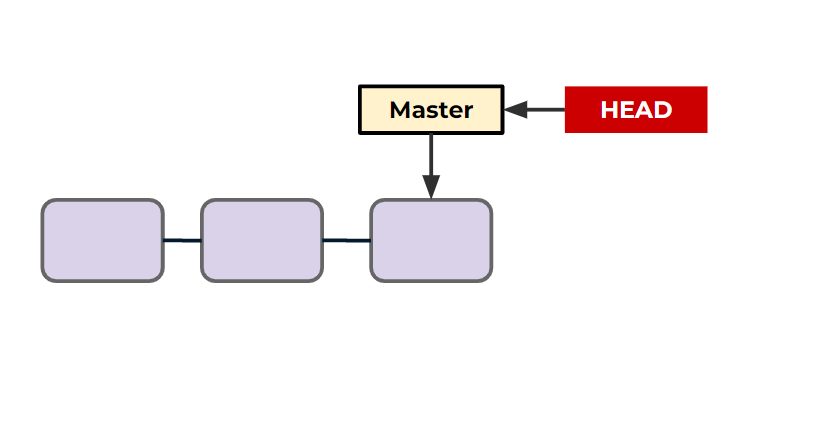
**Master:**

o While organizations and developers often treat this master branch as the official branch for things like deployment, this is not a requirement.

o You can use any branch for code deployment or code that’s actually “in-use”.

* Master vs. Main
* GitHub has changed the nomenclature for this initial branch to be **main branch** while Git is still using **master branch**
* Branches are just pointers to commits.
* When you create a branch, all Git needs to do is create a new pointer, it doesn’t change the repository in any other way.

* HEAD will help us understand what we are currently “viewing” or where we are “located” in regard to branches and commits.
* A HEAD is simply a reference to a commit object.
* We can think of HEAD as pointing to a specific commit in a branch that we are currently viewing.



o Create a New Branch

§ **git branch <branch\_name>**

o Report Branches

§ **git branch**

o Switch Branches

§ **git switch**

* **Deleting a Branch**
  + **git branch -d branch\_to\_delete\_name**
    - You can not delete a branch you are checked out at.
    - You also will get a warning if the branch is not merged.
      * You can confirm you want to do this anyways with **-D**

**Merges:**

When a new branch is created, but the original branch it stemmed(created) from has no additional commits. This is known as a “fast-forward” merge

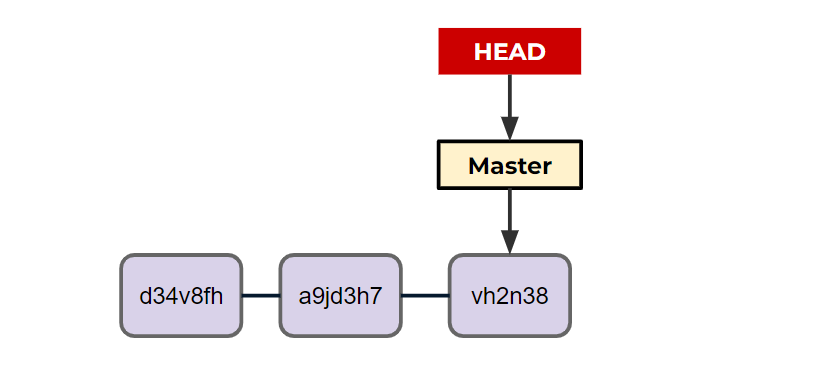
* **Merge in Practice**
  + Fast Forward Merge
  + Multiple Branch Commit Merge with No Conflict
  + Multiple Branch Commit Merge with a Conflict

**git checkout:**

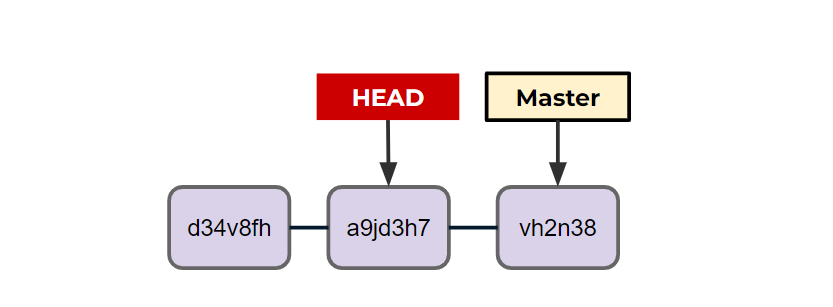
* + This is actually a very versatile command, so versatile in fact, that developers complained it was used for too many different actions, thus new git commands were created, such as **git switch**.
  + A "checkout" is the act of switching between different versions of a target entity.

The **git checkout** command can operate on three distinct entities: files, commits, and branches.

* + We can check out a particular commit by specifying its hash, we can get hashes from the **git log** command and we can also see the abbreviated hash using:
    - **git log --oneline**
  + Then we can provide the hash as:
    - **git checkout #######**
* Typically, our HEAD points to the branch which points to the latest commit.



Upon calling **git checkout a9jd3h7** we detach the HEAD to a previous commit



* You can think of this as traveling back in history to what your code looked like when you ran this commit.

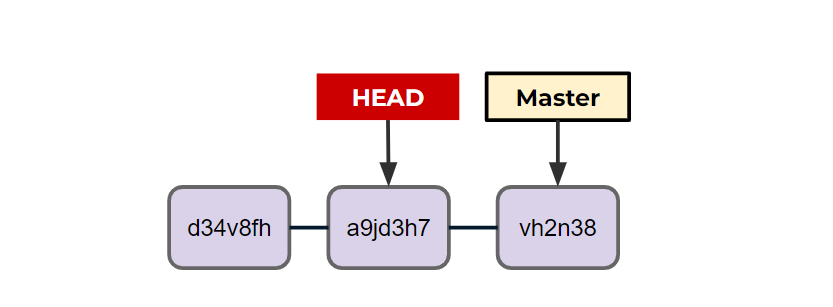
**git reset**:

Allows us to remove commits and “reset” the branch.

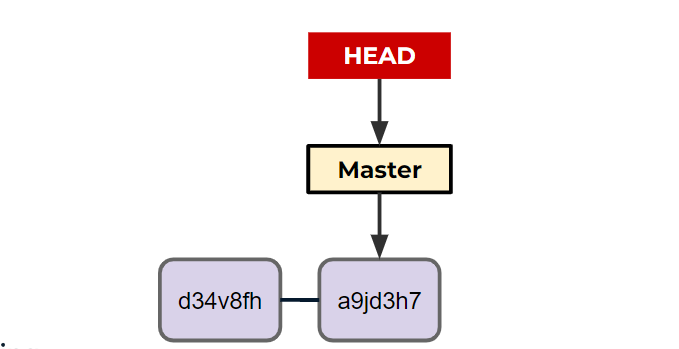
* There are two main types of **git reset** calls:
  + **git reset #######**
    - Removes commits in front of the specific hash called, files unchanged.
  + **git reset ####### --hard**

Removes commits *and* the changes in the files.

* We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)

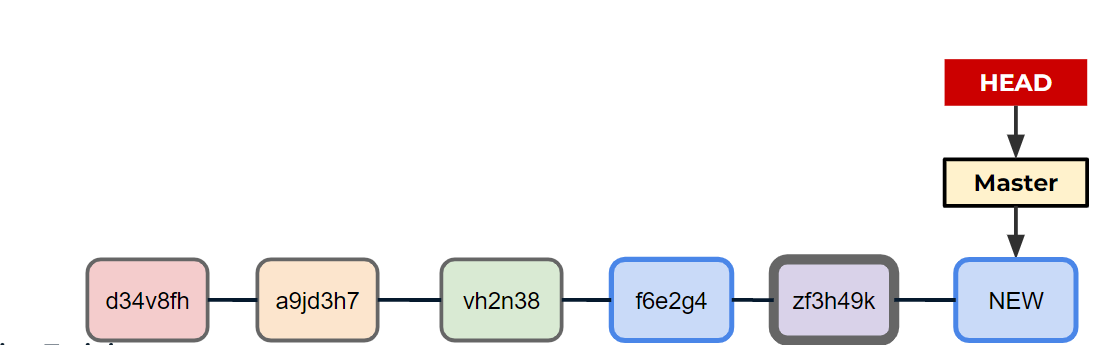
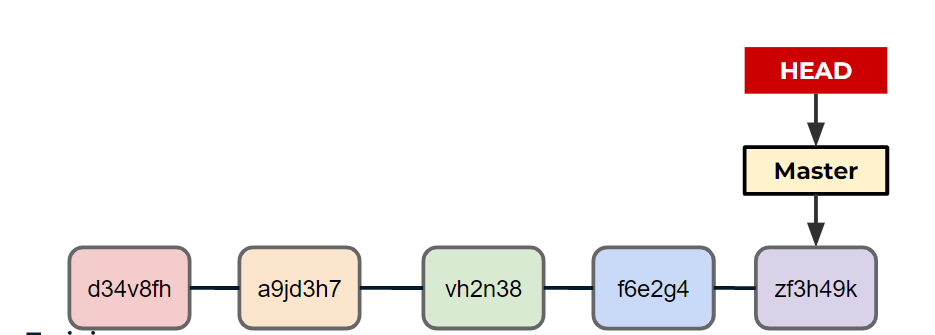
**git reset without hard**

**git reset with hard**



* A **git revert** creates a new commit that matches the historical state of a previous commit.

* The **git revert** command is a forward-moving undo operation that offers a safe method of undoing changes. Instead of deleting or orphaning commits in the commit history, a revert will create a new commit that inverses the changes specified.
* **Git revert** is a safer alternative to git reset in regards to losing work.



**git stash:**

* + Temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.
  + Stashing is useful when you find yourself needing to quickly switch branches to work on something else, but are in the middle of changing a file.

**GitHub Actions:**

Is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.

* You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.
* GitHub Actions are executed upon an event, and the “runners” of jobs can be stacked on top of each other:
* If you work on large projects with many users on GitHub, you’ll often see GitHub actions in use to help maintain the codebase.
* The actions themselves are YAML files with instructions for what GitHub should do upon an event.

Ways to view your remote URL:

* Use the **git remote -v** command. This will list all of your remotes, along with their URLs.
* Use the **git config --get remote.origin.url** command. This will print the URL of the remote named origin.
* Use the **git ls-remote --get-url origin** command. This will print the URL of the remote named origin, even if it is not the default remote.

The -M option in git is used to move a branch to a new location. For example, if you have a branch named master and you want to move it to a new location called main, you would use the following command:

git branch -M main

This will move the master branch to the main branch.