
git

# Review

*git* is a type of version control system (VCS)

*git* is useful for coordinating work among multiple people

*git* tracks progress over time by saving “checkpoints”

GitHub is not *git*

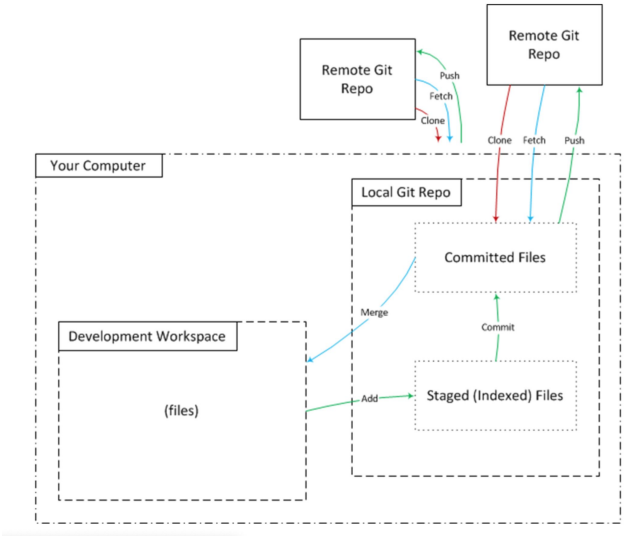
git makes it easier to track files (what changed, who changed it, and why it was changed)

git is useful for coordinating work among multiple people

git tracks progress over time by saving “checkpoints” or versions of a project

GitHub is not git. GitHub is a web services that hosts remote, distributed repositories for use with git.

# *git* Workflow




git tracks your local files by storing them in a central repository.

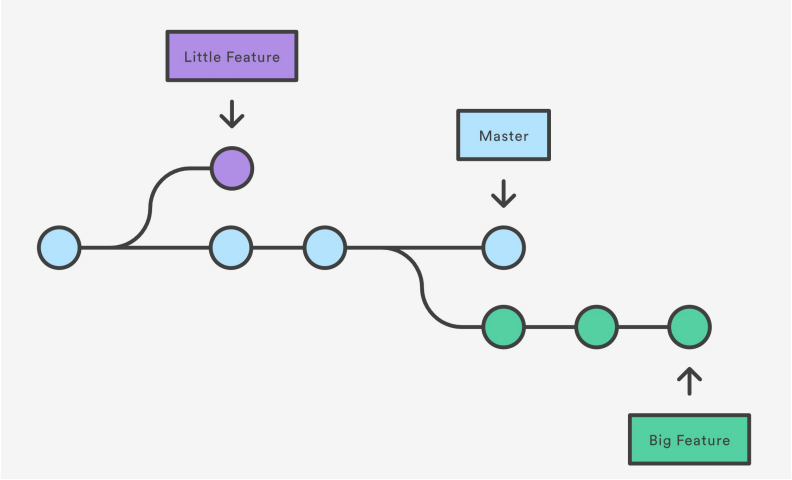
The central repository can be **remote, distributed, or both remote and distributed**. In all cases, the **workflow is the same**

When using git, you will always have a local repository and, if you are working with others, you have also have a central repository.

The local repository is what's on your system.

The central repository can be wherever. Most often it is hosted on some server somewhere, like a server that GitHub provides.

## *git* Workflow




A git repository, whether it is your local repository or the central repository, is structured like a tree.

In a git repository, you can have different branches. That is like having different concurrent versions of a project.

Similarly to a tree, the leaf nodes (or project versions) can share common parent nodes.

For example, Big Feature has everything in it that Master has in it up to a certain point. This is where they diverged.

How is this useful? Perhaps I want to create two versions of a website. I want to create two websites that have the exact same functionality, but look differently.

So here, the Master branch might be the original website and Big Feature might be the mobile version of the website where I had to change the layout.

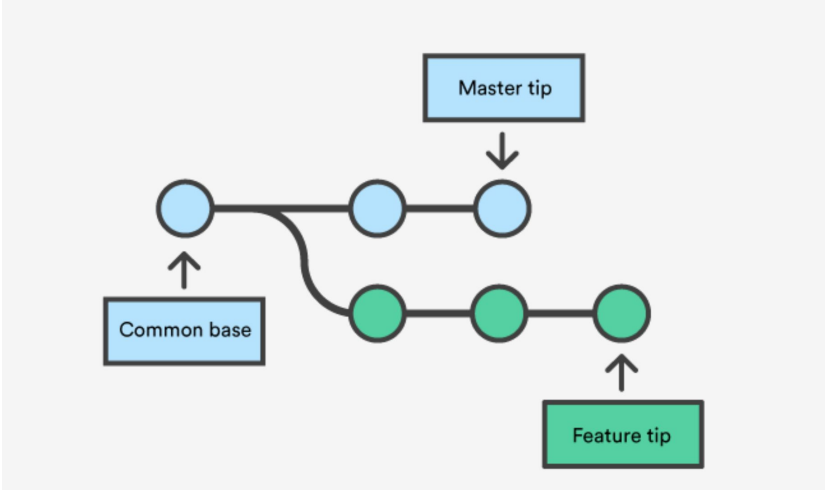
A repository only has a single Master branch. The idea is, that the Master branch is always the most current working version.

You can also think of it as the version that is currently released or will be released.

Whereas branches are new features or new code that have yet to be integrated into

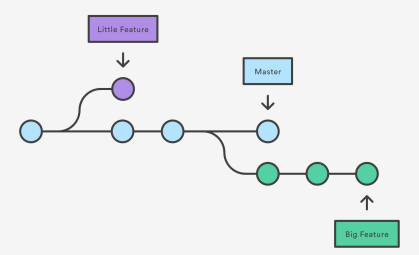
the Master version.

# git Workflow

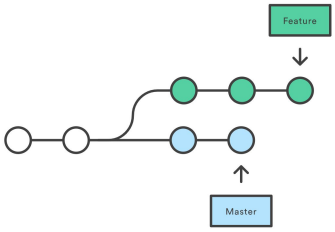



So again, Master and Feature have a common base.

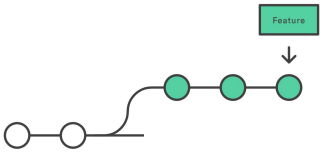
# *git* Workflow




Central repo




Local repo



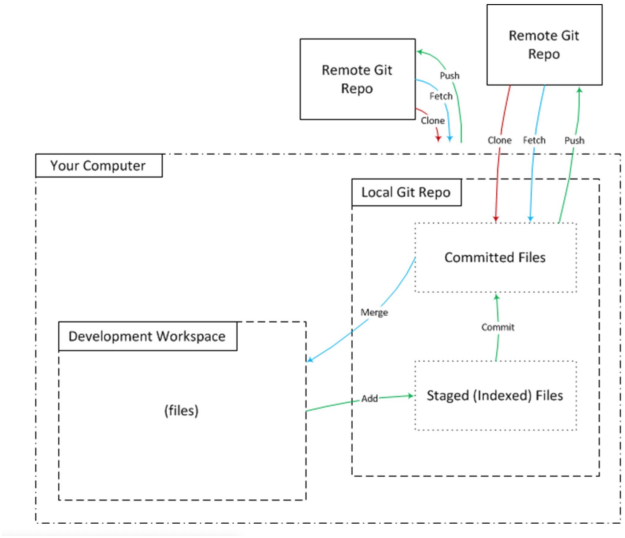

“working tree”

The central repository stores everything while your local repository may contain everything or just a subset of the files.

Because if someone else has saved changes to the central repository you need to download these changes to see them.

And your “working tree” only contains those files that you have checked out from your local repository.

# *git* Workflow



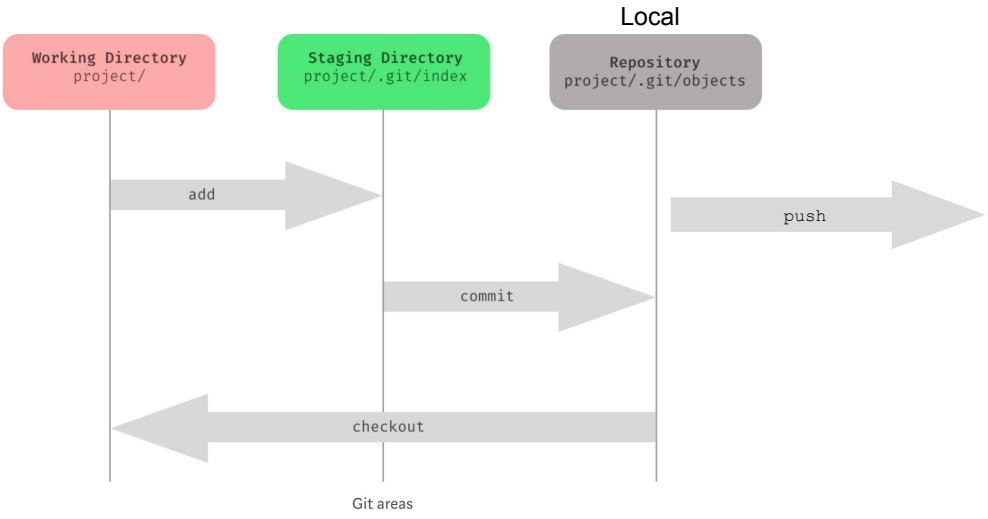

Let's go back to our diagram.

So you create and edit files in your working tree. Then you add and commit them to your local repo. Then you push them to your remote repo a.k.a. your central repo.

The other way around. You fetch the content from the remote repo. Then you checkout that content to place it in your working tree.

You can make changes to that content. Then add it to the staging area. Commit it to your local repo and finally push it to the remote repo.

## *git* Workflow




Let's talk about how this works exactly.

When you make changes in your working tree, those changes only happen in your working tree.

To tell git to store those changes, you have to add the files that you changed to the staging area.

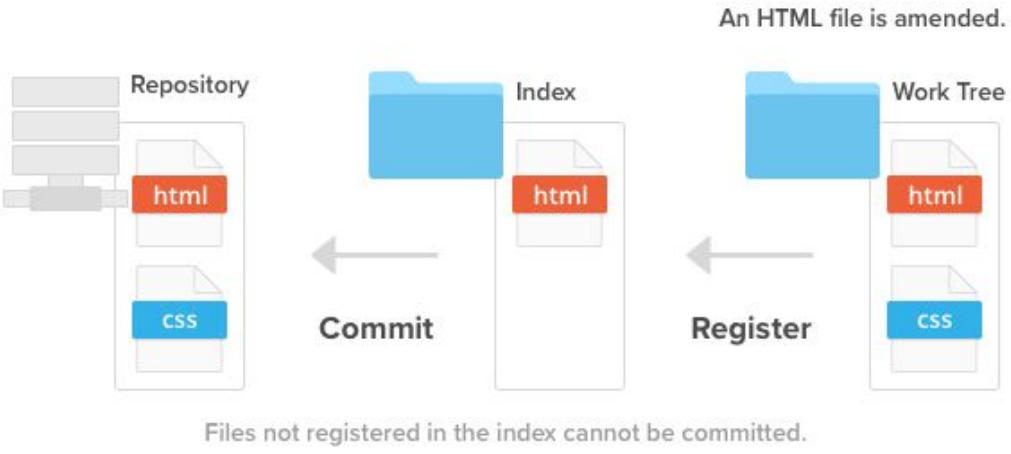
Although it is useful to think of the staging area as some real area like a directory where git puts the file changes, this is not true.

Git doesn't have a separate area or directory for staging.

Instead git has a file called the index where it keeps track of the file changes in all three areas: working directory, staging area, and the localrepository.

This is why you have to stage all of your new files and all of the files that you have made changes to.

## *git* Workflow




Here is an alternative view.

You have files in your working tree.

You need to register new files with the index and you have to tell the index when you modified a file.

Then you commit those files to the local repository. Committing files is like saying “yes, I approve those changes”.

When you commit, all the changes you have made in your working tree and staged, will then be made to the central repository.

Any file, modified or new, that has not been registered with the index, has not been staged, will not be committed.

You don’t have to stage all files in order to commit. You just have to stage the files you want to commit.

# git Commands

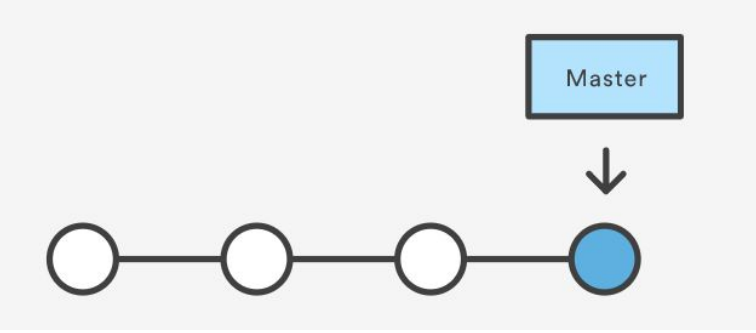
```
$ git add <filename>           # add a file to the staging area
$ git add file1 file2 file3
$ git add .                    # add all files in current directory to staging area

$ git commit <filename>            # save a file to the local repo; file must be staged
$ git commit file1 file2 file3
$ git commit                       # save all staged files to the local repo

$ git push <remote>         # push changes in local repo to <remote> repo
$ git push <remote> <branch>   # push changes in specific local <branch> to <remote> repo
```

Here are the commands to save new changes to the remote repo.

# git Workflow



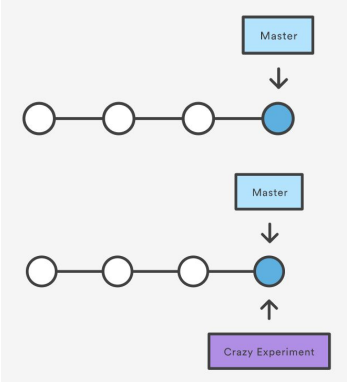

When you first create a git repository. You will start out with just a Master branch.

Let’s say I want to create a new branch for a specific feature I am developing.

# git Commands

```
$ git branch                    # list all branches in local repo
$ git branch <new branch>       # creates a new branch; does NOT check out new branch

$ git push <remote> <branch>
```




First I create the branch.

git automatically inserts the branch into our tree.

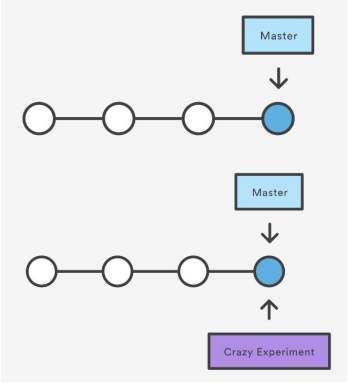
To upload the branch to the remote repository we simply push it.

# git Commands

```
$ git branch                    # list all branches in local repo
$ git branch <new branch>       # creates a new branch; does NOT check out new branch

$ git checkout <new branch>
$ git add .
$ git commit

$ git push <remote> <branch>
```

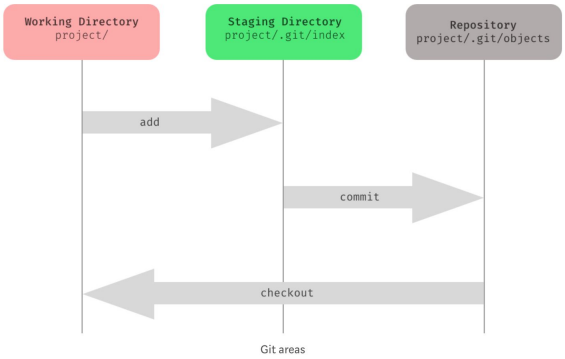


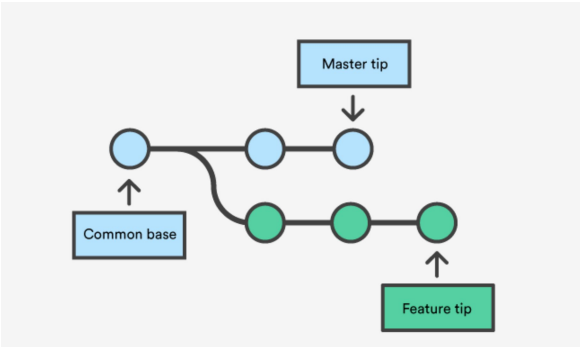

What if I want to work on the new branch I just created.

I first need to checkout my new branch.

Then I can add files, commit them, and push the local changes to the remote repository.

# git Checkout



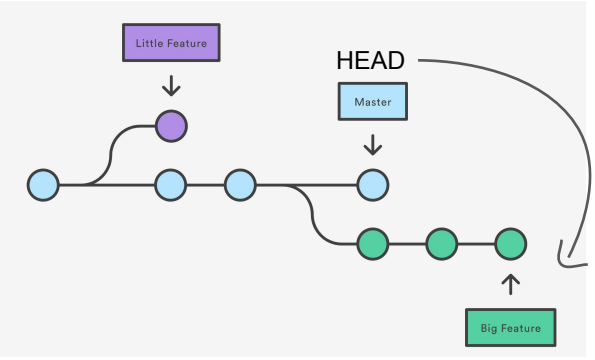

git checkout is a command that lets you switch between branches.

How does this work?

# git HEAD

- HEAD is git's way of referring to the most current version
- git checkout simply updates the HEAD

```
$ git checkout remote/big_feature
$ git checkout -b big_feature
```




git uses pointers to keep track of your commits and your commit history so that you can go back through your history

HEAD is a pointer or a reference to the most up-to-date version of whatever you are currently working on.

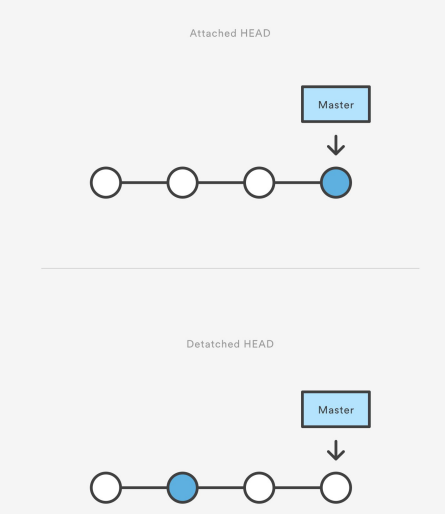
It should always point to a leaf node in your tree. git uses HEAD to determine where it should append new commits in your commit log.

git checkout simply updates the HEAD to point to the checked out branch

So to make sure that my commits are going to the correct branch, I have to check out the correct branch.

# git Checkout

- Check out one branch at a time
- You can check out old commits or previous versions




You can only check out a single branch at a time. You can only work on a single branch at a time.

If you check out a another branch, the files in your working tree will be removed and replaced with those from the second branch.

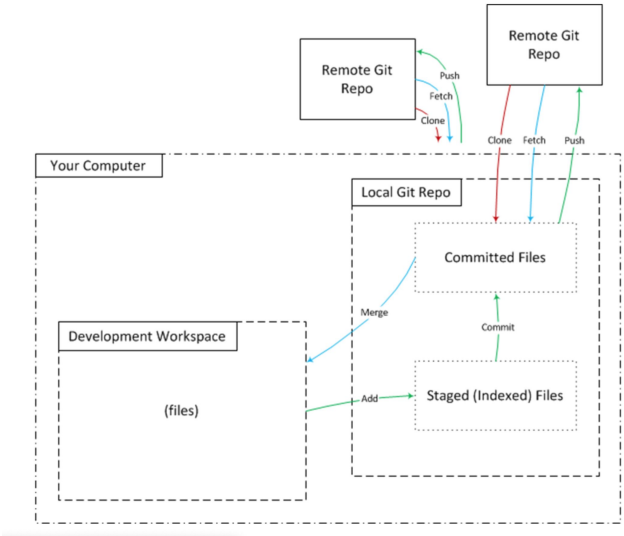
You can check out old commits a.k.a. old version, but this will result in a detached HEAD.

Detached HEAD means that whatever you are currently working on is detached from the rest of your project.

git doesn’t know where to save the new changes.

If you were to start developing code while in a detached HEAD state, there would be no way to get to those changes later.

# *git* Workflow



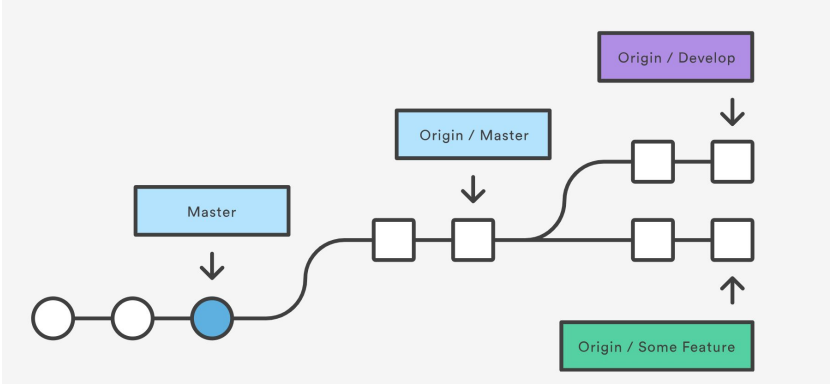

What if we want to download new content from a remote repository into our local repository.

We can do that in two ways, we can use fetch or pull.

Let’s talk about fetch first.

# git Commands

```
$ git fetch origin              # downloads all of the new branches: origin/master, origin/develop, origin/some-feature
$ git checkout master           # checkout local master branch
$ git merge origin/master       # merge the remote master branch with the local master branch
```




So let's say I haven't worked on my project in a couple of days, but my teammates did. They pushed a bunch of changes to the remote repository and I want to download them.

Let's say our remote repository is called "origin"

In this example, we are fetching the remote repository called "origin"

The fetch command downloads all of the branches of origin. It gives you the entire branch structure.

But the fetch command doesn't know where to insert the newly downloaded content into our local repository.

We need to do that manually.

So we first checkout our local branch and then merge the remote branch into our local branch.

The merge command merges the content and commits it to the local repository.

# git Commands

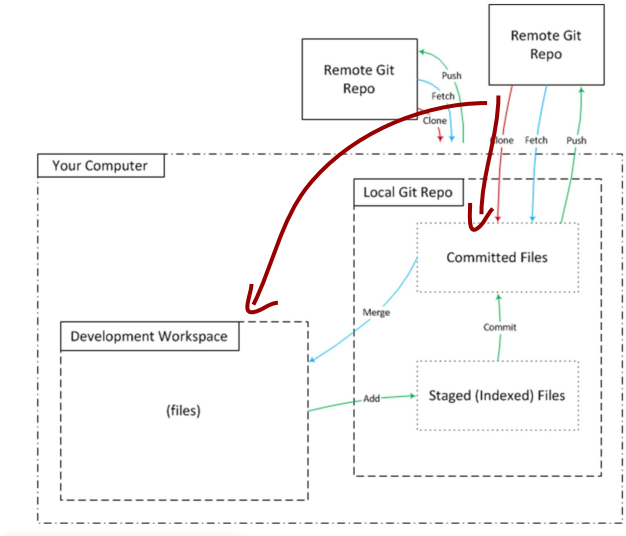
```
$ git fetch <remote>                # fetch all the branches from remote repo
$ git fetch <remote> <branch>       # fetch specific <branch> from remote repo

$ git checkout <branch>             # checkout <branch> to working tree
$ git checkout -b <new branch># create <new local branch> with checked out <branch>

$ git merge <remote>/<branch>       # generates a commit; saves new content to local repo
```

Here are the commands.

# git Workflow



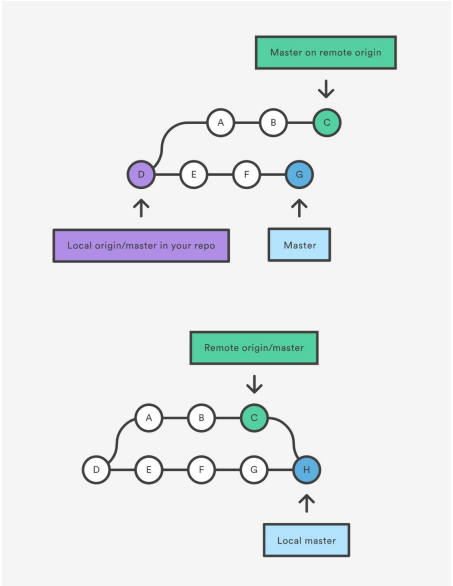

Now let's talk about pull.

git pull combines git fetch and git merge.

It automatically creates a new commit and merges the contents of the remote repo with the local repo.

# git Workflow

```
$ git pull <remote>
```




https://www.atlassian.com/git/tutorials/syncing/git-pull

**Links to online video training**

**https://try.github.io/levels/1**
**(start here)**

**https://www.git-tower.com/learn/git/videos**
**(11 free videos)**

# git for your project

**For your group project, you must create a git repositories, and share them with the instructor and grader**

**We will do this in lab tomorrow.**