# Git

Master or branches are not file directories, the cloned repository is the directory. The branches are just different indexes or maps of that directory, so give a different view of that directories structure and the contents within it. They don't create new files, they just decides what version of files are shown within the branch

**Repository:** The database that contains all of a project's information and history. Once added to the repository, information is immutable (never gets changed). New objects are created and indexed but the old ones remain.
**Index:** A binary file maintained by Git that describes the repository's directory structure and content at a point in time.
**Bare repository:** A repository without a working directory (*.git*) making it impossible to edit files and commit changes in. You would create a bare repository to *git push* and *git pull* from, but never directly *commit* to it (remote or central repo)
**Remote:** A link to another Git repo. Without a remote 2 repositories maybe the same but will have no link to one another.
**Fork:** A github/gitlab specific way to clone a someone's remote repo into your github/gitlab user account. This can then be cloned to your local machine and changes pushed to either the original remote repo, your remote repo or both.

**Master:** The top level of a repo, you should never make changes in here. Merge changes from branches into here.
**Branch:** A versioned copy of the master with its own index to keep track of its own directory structure and content.
**Working directory:** The area where the user can modify the files in the repo before committing them (normally a branch)
**Checkout:** Is the process of moving between branches and the master.

**clone**: A complete copy of a repository (master). This can be can be done from a local repo or a remote repo.
**commit:** Adds an entry in the repo (Git database) recording metadata for each change made to that branch of the repo.
**merge:** Takes a branches committed changes and merges them into the master. After a merge the branch can be deleted

**fetch:** Retrieve changes from a Git remote. Use to check differences with the remote repo without merging anything
**push**: Push committed branch or merged master to a Git remote. You should only use git push with bare repositories
**pull:** Retrieve and merge changes from a Git remote. It is really doing a git fetch followed by a git merge.

When working on a project that will take a while to complete you would commit changes to local branch but not merge into master until the project is complete. If don't want to push changes back to master can just kill the local branch. Whilst experimenting can create various branches and either kill, commit or merge dependant on the outcome of testing.

## Getting Started

To start with install git on local computer using *command line tools and create a folder for all the repositories.*

| | |
|---|---|
| **mkdir /Users/mucholoco/git-repos** | *Created a folder called git to put all git repos* |

On MAC for colours and awareness (names of checked in branches) need to enable *git-prompt.sh* and *git-completion.bash* which are by default are installed by *command line tools*. To enable them add to ***.bash_profile***:

```
source /Library/Developer/CommandLineTools/usr/share/git-core/git-prompt.sh
source /Library/Developer/CommandLineTools/usr/share/git-core/git-completion.bash
PS1="\h:\[$(tput setaf 6)\]\W\[$(tput setaf 2)\]\$(__git_ps1)\[$(tput setaf 4)\]\[$(tput sgr0)\]\$"
```

| | |
|---|---|
| **source ~/.bash_profile** | *Need to do this for the changes to take effect* |
| **echo $PS1** | *PS1 is the primary bash prompt, this checks its settings (default was \h:\W \u\$)* |

For ubuntu: https://askubuntu.com/questions/730754/how-do-i-show-the-git-branch-with-colours-in-bash-prompt

| | |
|---|---|
| **source ~/.bashrc** | *Need to do this for the changes to take effect* |

A few other Git settings that should be configured within Git:
**git config --global user.name "***name***"**
**git config --global user.email "***email_addr***"**

| | |
|---|---|
| **git config --global color.ui true** | *Enables colored output in the terminal* |
| **git config --global core.editor nano** | *Tells git to use nano for text editor with commits* |

Github and Gitlab are remote reops with web interfaces where you are using https or ssh to clone repos and push/pull changes. I added my macoloco SSH key to them so can-do pushes to these remotes without having to enter the password.

| | |
|---|---|
| **$pbcopy < ~/.ssh/id_rsa.pub** | *Useful way to copy ssh key on mac* |

```
$ssh -T git@github.com          Hi sjhloco! You've successfully authenticated, but GitHub does not provide shell access.
$ssh -T git@gitlab.com          Welcome to GitLab, @sjhloco!
```

## Create local Git Repo

Every Git repository has a hidden .git sub-directory that is the brains of the repository; it's where Git tracks your changes, stores commit objects, refs, etc. It defines the directory structure and content at a given point in time.

a. Create a new repository. Have a new directory that you want to make into a Git repo

```
macoloco:project1$ git init                                  Makes the current dir a git repo
 Initialized empty Git repository in /Users/mucholoco/git-repos/project1/.git/
macoloco:project1 (master)$ touch readme                     Make a readme for the repo
macoloco:project1 (master)$ git add readme                   Add the new object (orphaned) to the repo
macoloco:project1 (master)$ git commit                 Index the object, it updates the directory structure

macoloco:project1 (master)$ git status                       Check git repo status
macoloco:project1 (master)$ git log --oneline                Lists history of  commits
```

b. Create a new repo from an existing project. Have an existing project that you want to start tracking with git.

```
macoloco:project2$ git init                                  Makes the current dir a git repo
macoloco:project2 (master)$ git add <name>                   Add all the files in dir you want to be in Git
macoloco:project2 (master)$ git add .                  Can just add all files rather than each individually
macoloco:project2 (master)$ cat > .gitignore                 Create and include files don't want in git
test1.txt
macoloco:project2 (master)$ git add .gitignore               Need to add gitignore file to git
macoloco:project1 (master)$ git commit
```

You don't have to have a *.gitignore* file, but if you don't, those files will show up every time you type git status.

A bare repository is used to git push and git pull from, but never directly commit to. Central repositories should always be created as bare repositories because pushing branches to a non-bare repository has the potential to overwrite changes.

```
macoloco:project3$ git init --bare        Create a bare repo. Used as a storage facility, as opposed to a development
```

## Clone a Git Repository

*git init* and *git clone* can both be used to *initialize* a new git repository. However, git clone is dependent on git init as it first calls git init to create a new repo before coping the data from the existing repo and checking out a new set of working files.

When you clone a repo it creates an exact copy of all the files from the .git repository at that point in time.
The new clone is a local master from which you can create branches off. If you clone a remote repository to a local system it creates a relationship between master branch on local machine and master branch on the remote repository.

```
macoloco:git-repos$ git clone project2 project3              Clone project2 to a new repo called project3
Cloning into 'project3'...
done.

macoloco:git-repos$ git clone https://github.com/sjhloco/test2.git          If private asks for user/pass
Cloning into 'test2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

macoloco:test2 (master)$ git remote -v
origin        git@github.com:sjhloco/ test2.git (fetch)
origin        git@github.com:sjhloco/ test2.git (push)

macoloco:git-repos$ git clone https://github.com/sjhloco/test2.git test4        Give cloned repo new name
```

History of the commits/ changes to the repository each identified by a hash and description (commit comment)

```
macoloco:test2 (master)$ git log --oneline
macoloco:test2 (master)$ git log
```