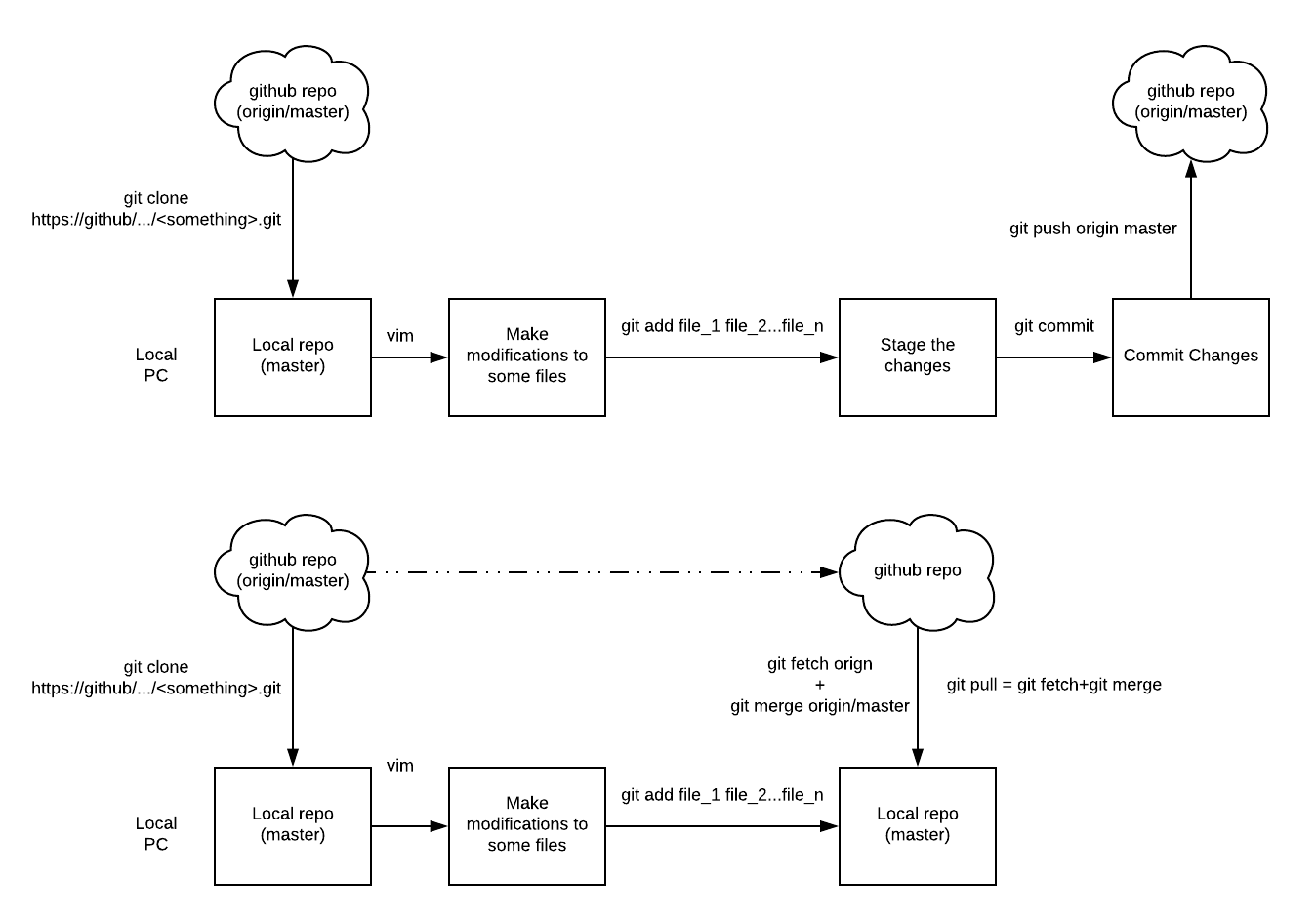
# Git Workflow



**Start a repository for your project or create a clone of a repository**

|  |
| --- |
| *For your already existing project that you want to version control:*  $ cd /home/..../my\_project  $ git init  *Suppose you are actually creating a clone of an existing repo, then simply..*  ~: git clone https://github.com/saravanadinesh/ESPLite.git  Cloning into 'ESPLite'...  remote: Counting objects: 34, done.  remote: Compressing objects: 100% (32/32), done.  remote: Total 34 (delta 6), reused 0 (delta 0), pack-reused 0  Unpacking objects: 100% (34/34), done.  Checking connectivity... done.  ~: cd ESPLite/  ~/ESPLite: ls -a  . Controller.ino Hardware.ino \_P001\_Switch.ino \_P005\_DHT.ino \_P034\_DHT12.ino README.md WebServer.ino  .. \_\_CPlugin.ino Misc.ino \_P002\_ADC.ino \_P006\_BMP085.ino \_P050\_ESPinfoSensor.ino \_\_ReleaseNotes.ino Wifi.ino  \_C002.ino ESPEasy.ino Networking.ino \_P003\_Pulse.ino \_P013\_HCSR04.ino \_P51\_WaterValve.ino \_Reporting.ino  Command.ino .git \_\_NPlugin.ino \_P004\_Dallas.ino \_P029\_Output.ino \_\_Plugin.ino Serial.ino |

Notice that, in the case of cloning of a repo, you would get the latest version of all the files in the repo as well as the entire repo itself in the .git/ directory.

In git, data exists in three distinct spaces: 1. Modification Area, 2. Staging Area, 3. Committed Area. So the latest version of the project is in the Modified Area. Once someone wants to submit those modifications to the repo, one has to first stage them - this happens with a ‘git add’ command that creates a list of changes you would like to submit. And then that someone has to actually commit the changes listed in the staging area using the ‘git commit’ command.

**(Modifying), Adding (Staging) and Committing Changes**

|  |
| --- |
| *In case the repo is empty and you are populating it the first time...*  $ git add \*.c  $ git add \*.ino $ git add LICENSE $ git commit -m 'First time populating the repo with existing files'  *In case the repo is a clone of an already old repo with many revisions and files in it*  $ git add modified\_file1.c modified\_file2.ino  $ git commit -m ‘Added feature A and fixed bug no. 233423’ |

Note that, instead of typing in the commit message with a ‘-m’ in ‘git commit’, if one simply typed in ‘git commit’, automatically an editor will open up - one can write a long message in it. This is preferred as it gives really a good summary of what that commit was about to other collaborators of the project.

Initially when a repo is created all the files will be on the main branch, or the tree trunk or whatever you want to call it. It is called ‘master’ in git terminology. If the repo is a clone of a remote master (say in github), then the remote master is called ‘origin/master’ and the local master is called ‘master’. In git, all the commits first go into the local repo copy of the remote (github) repo. It is the responsibility of the user to push these changes to the remote master.

If you are pushing a local repo for the first time to gitHub, create the github repo first, go to the repo’s Quick Setup page, copy the URL and then:

$ git remote add origin <remote repository URL>

# Sets the new remote

$ git remote -v

# Verifies the new remote URL

Diff - Comparing local file with remote

$ git diff HEAD <path to local file relative to pwd>

Overwrite changes made to a file

git checkout path/to/file

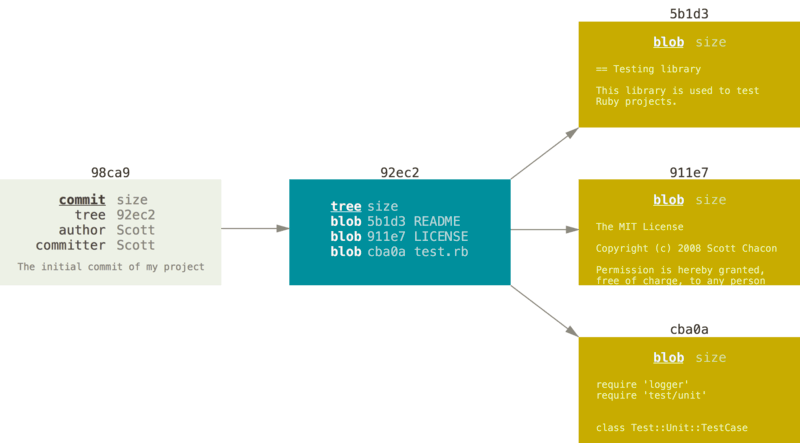
**Branching Out and Merging in**

When a project is started in git, the default branch is called a “master” branch.

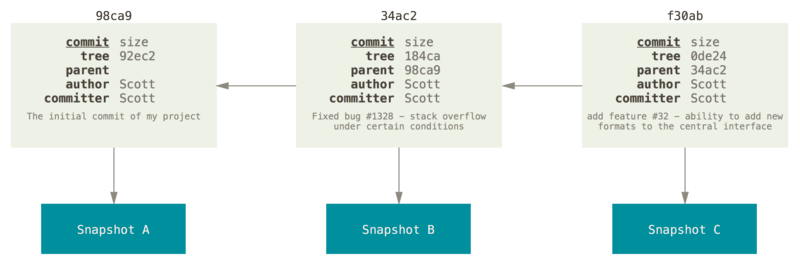
|  |
| --- |
| *Create a branch and checkout files*  $ git branch new\_branch # Create a branch called ‘new\_branch’ off of the master branch  $ git checkout new\_branch # Checkout files (otherwise working directory files will still be tracked with the master branch)  *Alternatively,*  $ git branch -b new\_branch # Combines ‘git branch’ and ‘git checkout’ |

# Git Internals

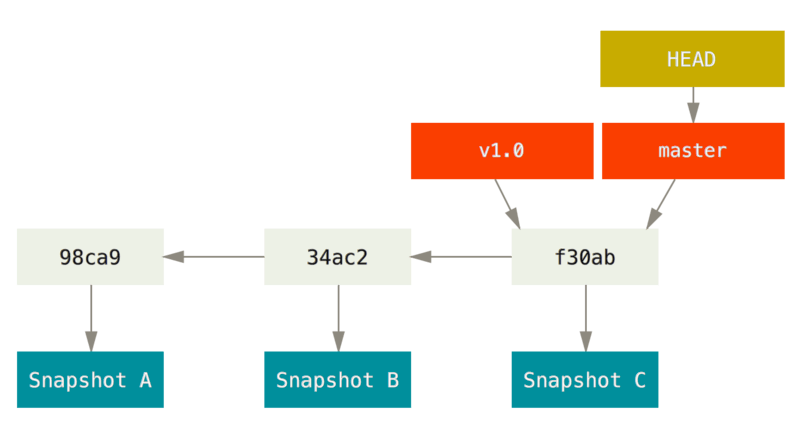
Git stores everything in a linked list. The very first time a repo is populated by using a commit operation, inside the .git directory, git creates one blob - or Binary Large OBject - for each file. This blob is just the data in the file without any of the metadata. A 40-character SHA-1 hash is generated for the blob and the blob is referred by everyone by using this hash. This blob will be a node in the git linked list. The folder in which the file is contained is another node. Git creates this node with data that contains the name of files with SHA-1 pointer references. Later, the commit will be another node that contains metadata of these folders and SHA-1 pointer references to them as shown below (Note: The five digit hex values in the pictures below are short-hands for the 40 character SHA-1 values).



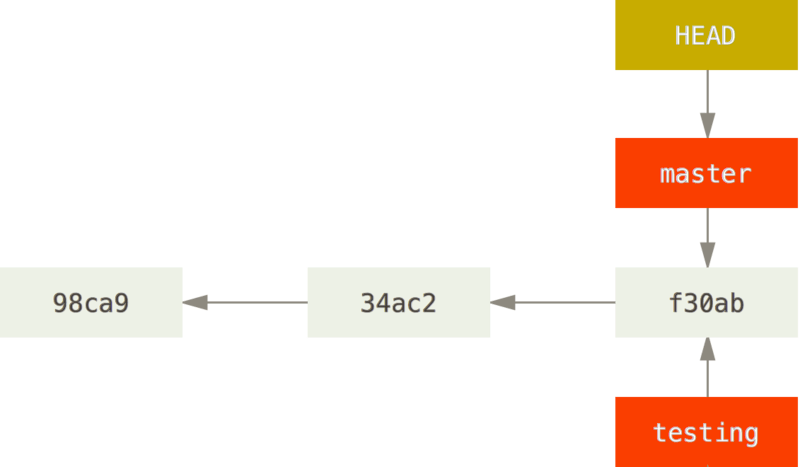
As more and more commits happen, links happen between different commit nodes in the same manner.



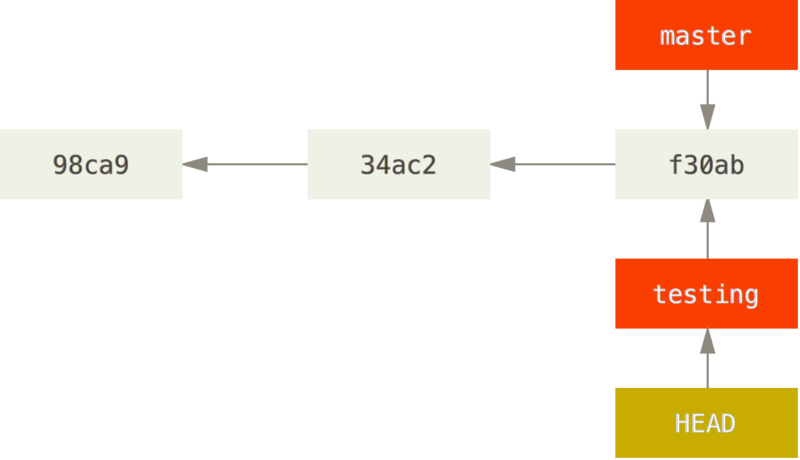
Again, branches and other references, such a ‘HEAD’ used to refer to the files you are working on, ‘origin/master’ used to refer to the commit that you synced with your local repo the last time etc., are all just nodes as shown below.



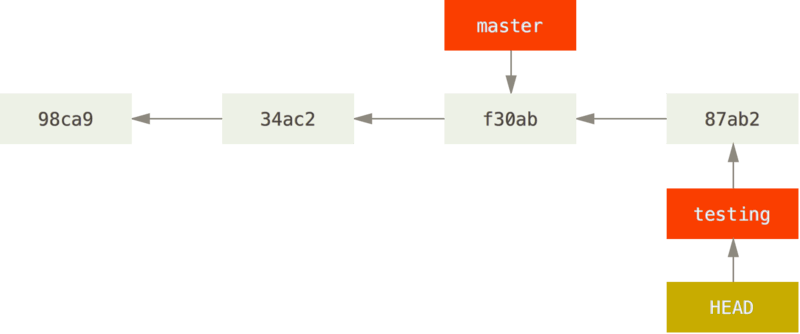
And when you create a branch using ‘git branch testing’ operation...



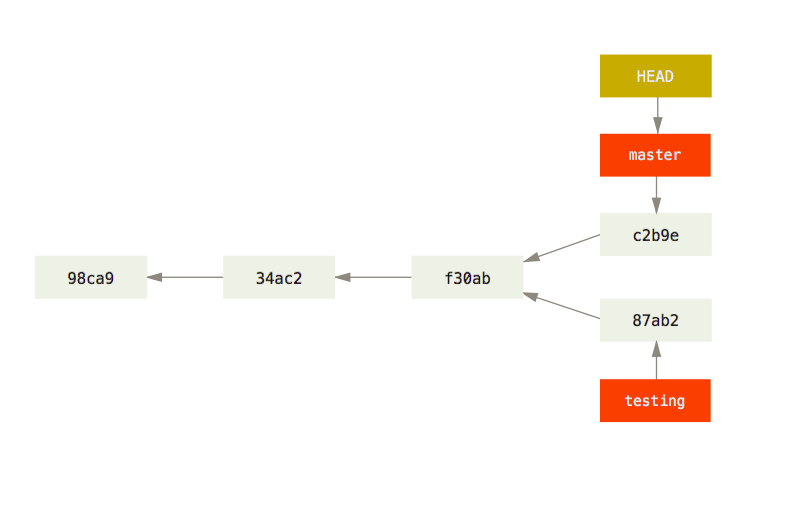
And then, when you execute, ‘git checkout testing’



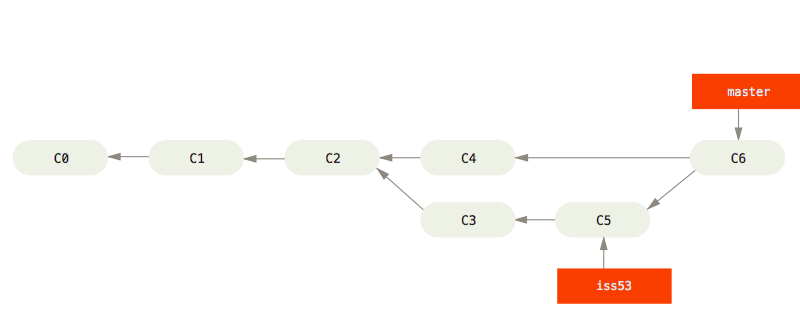
And when you make changes on the testing branch and do a commit,



If, after a while, you checkout the master again and made some changes there and checked in, then...



Merging a branch into the master creates links between the commits as shown below.



# References

[1] <https://git-scm.com/book/en/v2/>

[2] <https://www.tutorialspoint.com/git/git_basic_concepts.htm>