**GITHUB:**

*The option "--global" means that you set your username and email for Git globally on your computer. No matter how many projects with separate local repositories you create, Git will use the same username and email to mark your commits*

$ git config --global user.name "bannu5545"

$ git config --global user.email [veeramachaneni28@gmail.com](mailto:veeramachaneni28@gmail.com)

*Since you'll see the output from many Git commands in the terminal, it's best to have some pretty colors for the output. To turn on code highlighting, just run the following command:*

$ git config --global color.ui true

*let you view your Git configurations*

$ git config –list

*Once you run "git init", Git will initialize a hidden directory called ".git" in the project's root directory*

$ git init

*You might want to know the status of your box: does it store anything yet? To know the Git status* *(assuming you've added a file to the project's root directory)*

*Note the "Untracked files" message with the file "my\_new\_file.txt". Git conveniently informs us that we've added a new file to the project. But that isn't enough for Git. As Git tells us, we need to track "my\_new\_file.txt". In other words, we need to add "my\_new\_file.txt" to the staging area.*

$ git status

*With Git, this basket is the staging area. When you move files to the staging area in Git, you actually gather and prepare files for Git before committing them to the local repository.*

*To let Git track files for a commit, we need to run the following in the terminal:*

$ git add my\_new\_file.txt / $ git add my-file.ts another-file.js new\_file.rb / git add .

*(There's a problem with the "git add ." command. Since we're currently working in the root directory, "git add ." will only add files located in the root directory. But the root directory may contain many other directorieswith files. How can we add files from those other directories plus the files in the root directory to the staging area? Git offers the command below:)*

$ git add –all

*The option "--all" tells Git: "Find all new and updated files everywhere throughout the project and add them to the staging area." Note that you can also use the option "-A" instead of "--all".*

$ git add –A

*The status has changed! Git knows that there's a newly created file in your basket (the staging area), and is ready to commit the file.*

Again run $ git status

*Remember when we told you that you can take things out of your imaginary basket? Git can also take things out of its basket by removing files from the staging area.* *"--cached" option indicates files in the staging area.*

$git rm --cached filename

*Git is no longer tracking my-file.ts. In this simple way, you can untrack files if necessary. As an alternative to "rm --cached <filename>", you can use the "reset" command:*

$ git reset another-file.js

*Commit tracked files (staging area) with a message*

$ git commit -m ‘message’

*how can we add modified files to the staging area and commit them at the same time?*

$ git commit -a -m "Message"

*As you may recall, the "reset" command is the opposite of the "add" command. This time, "reset" tells Git to undo the commit. What follows "reset" is the "--soft" option. The "--soft" option means that the commit is canceled and moved before HEAD.*

$ git reset --soft HEAD^

*Instead of resetting the HEAD and undoing the last commit, we can rectify a commit by using the "--amend" option when committing to a repository. The "--amend" option lets you*amend the last commit*by adding a new file (or multiple files). Using the "--amend" option, you can also overwrite the message of your last commit.*

$ git add file-i-forgot-to-add.html

$ git commit --amend -m "Add the remaining file"

*Now you need to bind this remote repository to your local repository:*

$ git remote add origin [https://github.com/*project-bannu*/repository.git](https://github.com/project-bannu/repository.git)

*Now that you've added a remote repository, you can view the list of repositories by running the following command:*

$ git remote -v

*There's also that strange "-u" option. What it means is that we're lazy enough not to run a long "git push -u origin master" command each time we push code to the cloud. Thanks to "-u", we can run only "git push" next time!"*

$ git push -u origin master

$ git pull origin

*cloning*

git clone git@github.com:YourUsername/your-app.git

$ git clone git@github.com:YourUsername/your-app.git this-name-is-much-better

git remote origin

$ git commit -a -m "Do something once more"

git commit -a -m ‘committing all changes, equals to git add and git commit

git branch -a \* master remotes/origin/master

git merge newbranchversion

git reset --hard HEAD

git tag -a v1.0 -m 'this is version 1.0 tag'

git log commit

git diff

git archive --format=zip master^ README >file.zip

git gc

git fsck

git prune

### HEAD

This is an alias for the tip of the current branch, which is the most recent commit you have made to that branch.

### Index

The index, also known as the staging area, is the set of files that will become the next commit. It is also the commit that will become HEAD’s parent.

### Working Copy

This is the term for the current set of files you’re working on in your file system.

## **Flow**

When you first checkout a branch, HEAD points to the most recent commit in the branch. The files in the HEAD (they aren’t technically files, they’re blobs but for the purposes of this discussion we can think of them as straight files) match that of the files in the index, and the files checked out in your working copy match HEAD and the index as well. All 3 are in an equal state, and Git is happy.

When you perform a modification to a file, Git notices and says “oh, hey, something has changed. Your working copy no longer matches the index and HEAD.” So it marks the file as changed.

Then, when you do a git add, it stages the file in the index, and Git says “oh, okay, now your working copy and index match, but those are both different than HEAD.”

When you then perform a git commit, Git creates a new commit that HEAD now points to and the status of the index and working copy match it so Git’s happy once more.

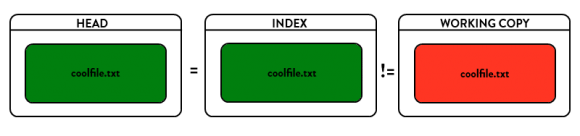
### Soft

The --soft parameter tells Git to reset HEAD to another commit, but that’s it. If you specify --soft Git will stop there and nothing else will change. What this means is that the index and working copy don’t get touched, so all of the files that changed between the original HEAD and the commit you reset to appear to be staged.

[](http://davidzych.com/wp-content/uploads/2014/05/reset-wc-index-changed.png)

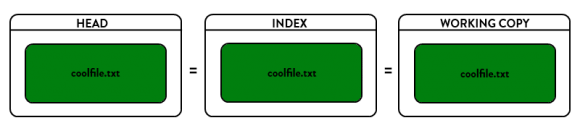
### Mixed (default)

The --mixed parameter (which is the default if you don’t specify anything) will reset HEAD to another commit, **and** will reset the index to match it, but will stop there. The working copy will not be touched. So, all of the changes between the original HEAD and the commit you reset to are still in the working copy and appear as modified, but not staged.

[](http://davidzych.com/wp-content/uploads/2014/05/reset-wc-changed.png)

### Hard

The --hard parameter will blow out everything – it resets HEAD back to another commit, resets the index to match it, **and** resets the working copy to match it as well. This is the more dangerous of the commands and is where you can cause damage. Data might get lost here\*!

[](http://davidzych.com/wp-content/uploads/2014/05/reset-all-happy.png)

BUGS:

ssh-keygen -t rsa -b 4096 -C "veeramachaneni28@gmail.com"

Generating public/private rsa key pair.

When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

Enter a file in which to save the key (/c/Users/*you*/.ssh/id\_rsa):*[Press enter]*

Enter file in which to save the key (/c/Users/Ashish/.ssh/id\_rsa):

At the prompt, type a secure passphrase. For more information, see ["Working with SSH key passphrases"](https://help.github.com/articles/working-with-ssh-key-passphrases).

Enter passphrase (empty for no passphrase): [Type a passphrase]

Enter same passphrase again: [Type passphrase again]

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/Ashish/.ssh/id\_rsa.

Your public key has been saved in /c/Users/Ashish/.ssh/id\_rsa.pub.

**Adding your SSH key to the ssh-agent**

Ensure the ssh-agent is running:

eval $(ssh-agent -s)

Add your SSH private key to the ssh-agent. If you created your key with a different name, or if you are adding an existing key that has a different name, replace id\_rsa in the command with the name of your private key file.

ssh-add ~/.ssh/id\_rsa

[Add the SSH key to your GitHub account](https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account).

To copy SSH key to clip board

clip < ~/.ssh/id\_rsa.pub

paste the SSH key in github account