  



**Git**

**Mina Bahgat**

**12 April 2020**

**Contents**

Getting Started with Git 3

GitHub: Getting Started 4

# Tesr

# Getting Started with Git

## Version Control System (VCS)

- Software desgined to recored changes made to files over time.  
- Ability to revert back to a previous file version or project version.  
- Compare changes made to files from one version to another.  
- Version control any plain test file, not just source code.

## The Three Stages of a Tracked File

- Committed: This is the file version committed in the DB.  
- Modified: This is the state of the file while modifying it.  
- Staged: This is the state of the file where the changes are marked to be committed to be added to a commit snapshot once changes are saved the state of the file is changed to committed.

These stages for tracked files only, any files added to the project that are untraked, you can put then in stage state and these files will be added in next commit.

## The Three States of a Git Project

- .git directory (Repository)  
Files in this state are committed and recorded to the project as a version snapshots.  
- Staging Area (Index)  
Files in this state have been modified and added to be staged in the next commit snapshot.  
- Working Directory  
Changes to files since the last checkout have not yet been added to the staging area for commit.  
  
Checkout: it is the step of getting a specific version of the project from the repository.

## Github

minabahgat/brnaba

If u initialize ur project using (git init), this project is still locally saved.  
In order to use cloud solution to solve this issue, there are many Git Cloud Providers, like:  
- Github  
- Gitlab  
- Bitbucket

### Github branches

As Github saves many versions of ur project, we need to use branches  
branch in git can be considered as point of time for ur project.  
so each branch on Github has a specific version (point in time) of ur project.  
When u initialize ur repo there is a branch called “master”, created by default.

## GitLab

You can install gitlab on a container using docker image.  
In order to host local Gitlab repo (not the Cloud one).

## Install Git

Debian: sudo apt-get install git  
Fedora: sudo yum install git  
Mac: brew install git  
Windows: <https://git-scm.com/download/win>

$ git --version

## Configure Git

$ git config --global user.name “Mina Bahgat” 🡪 Configure Git  
NB: --global will set the configuration globally if u need to config for specific project then no need for this flag.  
$ git config --global user.email [mina.bahgat.zarif@gmail.com](mailto:mina.bahgat.zarif@gmail.com)  
$ git config --list 🡪 check the configuration settings values.  
$ git config user.name 🡪 Check specific configuration set value.  
Mina Bahgat

## Create Repo

### Local Repo

1. Create a new directory where u need to add ur project files (mkdir)  
2. (cd) inside this directory then initialize ur repo (git init)   
3. Add files to this directory, but take care all added files are untracked ones.  
4. Use “git add <file> …..” to add any file to the repo, make it tracked file.  
5. Any changes to the tracked files need to be saved in the stage area waiting for a commit, using “git add <file> …..”  
6. Then you can commit all changes in the stage area

### Github Repo

1. Sign up on github.com  
2. Create a project (repository), <Account\_Name>/Repo-name  
3. U need to start using ur new repo.  
NB: U have many options to commit ur local project to Github repo:  
a. Use Github Desktop Client   
b. Use Https or ssh  
c. Push ur local repo using CLI

echo "# brnabaschool" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/minabahgat/brnabaschool.git

git push -u origin master

d. Import the code through Web browser.

## $ git help

Short help list.  
$ git help config 🡪 short list for the config options.

## $ man git

The man page gives all the details regarding the git command.  
You will notes in the man page 2 sections of the commands:  
1. HIGH-LEVEL COMMANDS 🡪 These are the most commanly used commands.  
2. LOW-LEVEL COMMANDS 🡪 These are the more advanced commands, these commands most properly are used by tools & scripts not manually on command line.

## $ git init

This command initializes an empty repo for ur project.  
The repo schema (config folders & files) are created under a hidden directory named (.git).

## $ git rm <file\_name>

Remove the file from the repo.  
Next commit this file will be removed from the project.

$ git rm --cached <file\_name>  
This will untrack this file but leave it in the working directory.

## $ git mv <file\_name> <new\_destination>

This command is used to move the file to nes destination, it can be used of course to rename the file as well.  
  
The change will take place with the next commit command.

## $ git add <file\_name>

To add file in the stage state.  
   
NB: This is used to track any newly added file to ur repo.

## $ git commit

To add changes in the stage state to the repo creating a new commit snapshot with all changes in stage state.

$ git commit –a –m “Add changes in Working Directory”🡪 -a option will commit all changes even the ones in the working directory (not staged yet), -m option will just write message on the screen while committing

The seven rules of a great Git commit message:  
  
Separate subject from body with a blank line  
Limit the subject line to 50 characters  
Capitalize the subject line  
Do not end the subject line with a period  
Use the imperative mood in the subject line  
Wrap the body at 72 characters  
Use the body to explain what and why vs. how

<https://chris.beams.io/posts/git-commit/>

## $ git reset <commit\_ID>

This command is used to reset the commits from the repo to staging area or working directory, this command uses the commits history.

This command is disructve so take care before running it.  
it is not recommended to reset changes once we pushed them to remote repos and pthers have access to ur changes.

$ git reset --soft <commit\_ID> 🡪 it will remove the specified commits to the staging area, usefull if u need to regroup the changes with the commits or u need to add domething before commiting.  
$ git reset --mixed <commit\_ID> 🡪 it will remove the specified commits to the working directory (default option).  
$ git reset --hard <commit\_ID> 🡪 it will remove the specified commits to the trash, but if u run this command any chnges in the working directory or the staging area will be removed as well.

NB: you can get the commit ID as well as the commit message using “git log --oneline” command

## $ git remote add origin https://github.com/....

To link ur local repo with Cloud one on any Git Cloud Provider (Ex: Github).  
This allow push & pull between local & remote repos.

## $ git push –u origin master

This will push ur local repo to ur remote repo that already linked with (git remote add) command.  
NB: this step will ask u the username/password to login ur account.

## $ git pull –u origin master

This will pull ur remote repo to ur local repo that already linked with (git remote add) command.

## .gitignore File

https://www.pluralsight.com/guides/how-to-use-gitignore-file

## $ git status

This will show u the status of ur local repo and if there is something waiting to be commited to ur remote repo on Github.  
it will show u the branch on Github, u are working on, it will even show it’s commit state compared to the master branch.  
It will also report the untracked files.  
  
Changes to be committed  
Changes not staged for commit

### $ git status –s/--short

This is same as git status but in much smaller format.

$ git status –s  
M M hello.txt

This means the file “hello.txt” is modified in Staged Area (first M), also modified in the working directory (second M).

M = Modified  
A = New file added to staging area  
?? = New file untracked by Git

## $ open –a vim <file\_name>

This command is used to open a file in vim editor, the changes done on the file need to be added to the stage area using ‘git add <file>’, so we can add these changes to the snapshot by next commit.  
  
NB: You can use any editor instead of vim

## $ git diff

Shows:  
1. What changes have I staged that are ready to be committed ?  
Check the difference between the committed version and the staged version of the file.  
git diff --staged  
  
diff –git a/file1.txt b/file1.txt 🡪 Compared Files.  
Index <first\_index> <second\_index> 🡪 File Metadata  
--- a/file1.txt   
+++ b/file1.txt 🡪Change Marhkers for file A/B  
@@ -12, 2 +12, 3 @@ 🡪 Chunk Header  
This header means starting from line 12 in file a (-12), we have 2 different lines and starting from line 12 in file b (+12) we have 3 different lines.  
Example lines… 🡪 Chuck Changes  
- Old Content  
+ New content

git diff --staged --no-renames  
The default case git don’t care of the file name, it just care of it’s content, so if we have 2 files of the same content but with different names (index similarity 100%), git will consider the 2 files are same file and u are just renaming it, if u need to change this behavoir and let git consider bot files as different ones then use “--no-renames” option.  
  
2. What changes I have done to working directory but not staged ?  
git diff

## $ git log

This will show u log with ur commits (basic info.), with the newest commit at the top of the log file.  
  
$ git log -3 🡪 This will show the lastest 3 commits in the log file.  
$ git log --oneline 🡪 simplified view of the commits (each commit in a line showing the commit message)  
$ git log --stat 🡪this will show the changes happened to the files in each commit.   
$ git log --patch 🡪 this file will get the detailed changes happened for each file in each commit.  
NB: if you need to fins the user that committed this change, you can search the log for the word “Author”

## $ git branch <branch\_name>

This command creates a new branch of the one u are working on now (by default it is the master branch), you need to checkout to this new branch if u need to work on it.

Branching let u work on ur project without affecting ur master branch (Production branch), then after finishing ur changes, u can merge the changes with ur master branch.

## $ git checkout <branch\_name>

This lets u checkout and commint on a specific branch.  
by default git is using master branch to checkout and commit, this command is used to change this default to another branch

$ git checkout master 🡪 this command is used to return (checkout & commit) to master branch.

$ git checkout –b <branch\_name> 🡪Create a new branch of the one u are on now (by default it is the master branch), this command create new branch and checkout at the same moment.

**Visualize Branches:**  
$ open <http://git-school.github.io/visualizing-git>

Use git status command to check the branch u currently checkout.

The git branch u currently checkout and commit is called the HEAD branch.

### sparse-checkout

How can I download a specific folder or directory from a remote Git repo hosted on GitHub? I only want just that a specific folder where the files for the front-end part of the app are kept, without having the other files as I don't really need them.

If you are willing to use git to do this, you can do a sparse checkout.

 We are going to start by creating an empty git repo locally, and then add the repository we want as a remote. This allows us to easily just checkout the folder we want

mkdir directoryName  
cd directoryName  
git init  
git remote add origin -f repoUrl  
git config core.sparseCheckout true # enable this

After that, we want to add the directory to   
cat >> .git/info/sparsecheckout  
<folder>  
(this is essentially the opposite of the gitignore file).

Then we can   
git pull origin master  
it should download only the folder you want.

NB:   
Git doesn't store empty folders. Just make sure there's a file in the folder like doc/foo.txt

## $ git stash

This will save the changes in the working directory as well as the staging area, it will even save the untracked files in the working directory.

All these changes will be saved for this branch without commiting these changes.

This is useful while checkout between different branches any uncommitted data will be lost, so this option is used to save all the uncommitted data specifically for this branch (saving them at their places working directory or staging area without commiting to the repo).

$ git stash list 🡪 will show list of stashes with the branch name of the stash  
$ git stash show 🡪 show the detailed data of the stash for the current working branch.  
$ git stash pop 🡪 to get the data back to their places in working directory or staging area.

## $ git clone <repo-URL>

This comman will clone ur repo to the directory, u are at.

## $ git merge new\_branch

This commad will merge new\_branch into ur currently working branch (by default master).  
All commited changes in new\_branch will be added to the master branch.

This command will open vim editor in order to write the merge comment “similar to the commit message”.

## Moving and Renaming Files on GitHub

<https://github.blog/2013-03-15-moving-and-renaming-files-on-github/>

## Untrack a file previously pushed with Git

Last week I had to untrack a file previously pushed on a git repository and I wasn’t sure on how to do it. Took me a while to wrap my head around the process so today I would like to share that in order to have it documented here.

This post will be composed by two parts:

1. Scenario
2. git rm --cached
3. git update-index --assume-unchanged

### Scenario

I have a file test already pushed in my repository.

> git ls-tree -r master

100644 blob 63123fbe81571b48b7d65602f9828524f9d84b5f .gitignore

100644 blob a6712f67380bebb75d15c817820e8d2f5c97fb4c test

Now I wanted to untrack the file from the repository.

### git rm --cached

If I want to **remove the file from the git repository**, I can do the following:

> git rm --cached test

> git commit -m "remove test"

> git push

rm is used to remove a file from the index (The index is where the staged changes are held). It would be the same as manually deleting the file and then staging git add . the deletion.

--cached is used to specify that **I want to keep my local copy**.

Therefore rm --cached test means **“remove test from the index but keep my local copy”**. And when we execute it, we get the following result:

> git status

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

deleted: test

Untracked files:

(use "git add <file>..." to include in what will be committed)

test

As expected, test is deleted and staged - Changes to be committed: 'deleted: test' - and it is removed from the index Untracked files: 'test'. I now need to add test to .gitignore and commit .gitignore it will be removed from the untracked files.

Once you push that, **test will be removed from the repository but your local copy will still remain and subsequent changes on the file will not be tracked.**

### git update-index --assume-unchanged

Now let’s say we have another scenario where we actually **do not want to remove the file from the repository**.

To do that we can do the following:

> git update-index --assume-unchanged test

This will tell git that we won’t change the file so no need to track it. The problem with that is that if we change test, our changes won’t be tracked and if the file changes on the remote repository, when trying to pull the latest, it will fail.

In that even, we need to undo the assume-unchanged then undo our changes to be able to pull again.

> git update-index --no-assume-unchanged test

> git checkout test

> git pull

**checkout revert the changes on the workspace to the index. If you staged the changes, you will need to use git reset test before to revert the changes staged on index then you can perform git checkout test.**

If you need to see what file are assume-unchanged, you can run git ls-files -v.

> git ls-files -v

H .gitignore

h hello

The files assume-unchanged are marked with a small h. If you have a lot of files, you can pipe grep and specify start with h to filter your files.

> git ls-files -v|grep '^h'

h hello

## Other

Aal github

Github api

Integration Jira with Gitlab

Pipeline in Gitlab create jira with the name of the pipeline.

Wokflow, pipeline in git