Tutorial on Git

A Tracking System

Yonas M.

Ethiopian Institute of Water Resources

December 6, 2018

Introduction to GIT i

Definition

Git: is a distributed open source version control system(VCS) which gives us an efficient control over our projects.

In short, it's a file tracking system.

What is a "distributed system"?

It is a system which provide a platform for developers or partners to work on the same project with out being in the same server.

What is "Version Control System"?

It manages and records changes to a files, documents, codes and projects over time so that we can recall a specific version at a later time (it's a time travel with in files).

Introduction to GIT ii

Git provides three working areas and it's critical to understand this three terms:

Working directory or local copy

This are the files you are editing and that will be in the next commit.

Staging area or index area

It is an intermediate space to store files scheduled for the next commit.

Repository

It is a subdirectory named .git and contains the whole history of your project.

Introduction to GIT iii

A typical workflow of any Git project takes the following pattern:

1. Modify files in your working directory.
2. Add a file to the staging area.
3. Commit a file to the git directory.

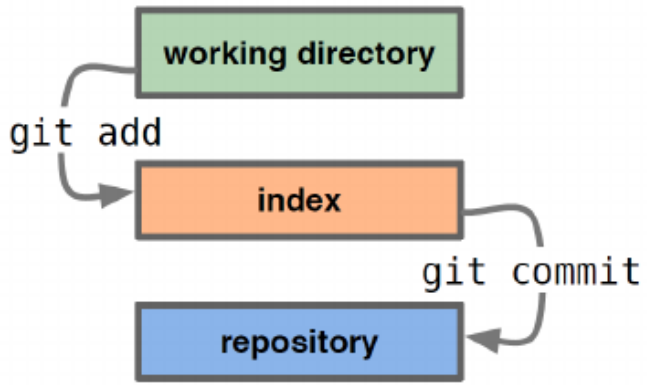

Figure: A typical workflow of Git project.

Advantages of using Git

Why we need to use Git?

- the life history of a file is recorded from the beginning so that, you can inspect any vision of the file by the author, time and content.
- at any moment you can revert to a previous vision of your file.
- it provides simpler approach for coordinated work between multiple developers.
- it displays the differences between each versions.
- it allows to work locally at your machine, with a remote repository and synchronizing both.
- it allows to work on multiple variants of one project as branches and merge them easily when needed.

Installing Git

Basically, you can use Git in two ways. The command line and the graphical user interfaces (GUIs). This tutorial is prepared based on the command line. If you know how to use Git on command line, then using it in GUIs will be come easier for you.

In Ubuntu the installation is simple, type the following command on the terminal and press enter:

Code for Git installation

```
sudo apt-get install git-core
```

For further instructions on how to install Git on Linux systems can be found at `http://git-scm.com/download/linux`

To check the version of Git

```
git --version
```

Configuring Git i

It's mandatory to setup your user name and email address, for credential purpose.
Replace the username and email address with your with your own.

Code for Configuration

```
git config --global user.name [user name]
git config --global user.email [e-mail address]
```

To colorize the output with different colors use the following code:

Code for setting the color output

```
git config --global color.ui auto
```

To check for the configuration of Git:

Code for checking configuration

```
git config --list
```

Configuring Git ii

Specifically, to check for the user name and email address:

Code for Git configuration

```
git config user.name
git config user.email
```

Remark

The configuration of Git must be done at once. If you want to use a different name/email address for a particular project. You can change it by navigating to the project directory and use the configure command but leaving out the "`--global`" part.

Configuration code for a particular project

```
cd [project directory]
git config user.name [user name]
git config user.email [e-mail address]
```

Getting Help

For getting help about Git commands, use the following methods and don't forget to replace the term [verb] by the command name.

Code for help

```
git help <verb>
git <verb> --help
```

A precise and quick reference on Git command's can be obtained using the following command:

Code for specific help

```
git <verb> --h
```

You can learn more about the individual Git commands using the following code:

Code for getting detail information on Git commands

```
man git
man gitglossary
```

Initializing Git

To initialize a local repository, use the [git init] command.

First create a folder or move to an existing directory where you want to track your files:

Code for initializing Git

```
cd [folder name]
git init [Repository name]
```

This creates a new repository with [.git] extension and will contain a full history your project. It's also a hidden so that you can use the [list all] command to view it.

Code to viewing hidden files

```
ls -la
```

Git status checking

At any stage, it's important to review the current state of your file, for that use the [git status] command.

Code for checking the file status

```
git status
```

You may find your file either as tracked or untracked file:

Untracked files: are files that are not known by Git.

Tracked files: are files that were in the last snapshot; they can be unmodified, modified, or staged (files known by Git).

If you need a summarized output of the Git status, use the following command:

Code for summarized output

```
git status -s
```

?? new files that are not tracked

A new files that have been added to the staging area

M modified files

Git Ignore

Sometimes there are files you don't want to track or show to others. In this scenario, first you have to create [.gitignore] file as follow:

Code for creating .gitignore file

```
touch .gitignore
```

The next step is to open the [.gitignore] file using a text editor and enter file names & extensions as described bellow:

Code for editing .gitignore file

[file.log] it tells Git to ignore a file named [file.log]

[*.log] it tells Git to ignore all files with [.log] extension.

[/dir] it tells Git to ignore a directory named [dir]

Then run [git status] command to see those files are under untracked category.

Git Staging

To begin tracking a file, you can use the [git add] command followed by the file name with it's extension.

Code for staging a single file

```
git add [file name]
```

It's possible to add files using wildcard. In this case, all files with the specified extension will be added to the staging area.

Code for staging using a wildcard

```
git add *.[extension name]
```

You can add all files at once using the following command:

Code to stage all files

```
git add -A
```

Removing & Renaming in Git i

Sometimes you need to remove a file from Git. You can either remove the file from staging area or completely from your working directory.

To remove a file from staging area to unstaged area, use the following command and it will show up under the "Changes not staged for commit":

Code for removing files from staging area

rm [file name]

To completely delete a file from the working directory and stage it to deletion you can use the following command as a result file will be gone and no longer tracked.

Code for completely removing files from Git

git rm [file name]

Removing & Renaming in Git ii

To remove the file from version control system but to preserves it in working directory, you can use the following command:

Code for removing files but keeping it in the working directory

```
git rm --cached [file name]
```

To completely remove the Git directory, use the following command:

Code for removing Git directory

```
rm -rf .git
```

If you want to rename a file in Git, use the following command:

Code for renaming a file

```
git mv [file-original] [file-renamed]
```

Resetting changes in Git

The rest command is used when you want to cancel current changes from the staging area back to the previous version or commit.

If you want to reset a file to the last commit ignoring the changes, use the fallowing command:

Code for resting a single file

```
git reset [file name ]
```

To reset all files at once from staging area, use the following command:

Code for resting all files

```
git reset
```

To overwrite all changes in the staging area and in the working copy, use the following command:

Code for resting all changes

```
git reset --hard
```

Git Commits

The Git commit command makes a "snapshot" of your staged files and saves it permanently in the git database.

To commit changes to the repository, use the following command:

Code for committing files

```
git commit -m"reason for the commit"
```

Important [-m]

The comments should describe the rationale behind your changes.

Git Inspecting & Comparing tools i

By running a variety of [git log] commands, you can get information about the commit history of a file such as who make the changes (author), when(time) and why(reason).

To show the commit history for the currently active branch, use the following command:

Code for displaying version history

```
git log
```

To display the full difference between each of consecutive versions, use the following command:

Code for viewing the difference b/n each versions

```
git log -p
```

Git Inspecting & Comparing tools ii

It's possible to decide the total number of versions to be displayed by setting the desired number.

Code for viewing fixed number of versions

```
git log -[number of version required]
```

If you want a summarized output as a single lines, use the following command:

Code for viewing summarized history

```
git log --oneline
```

Sometimes you may want to know the number of lines inserted or deleted from each version, for that use the following command:

Code to view the number of deletion/insertion of each version

```
git log --stat
```

Git Inspecting & Comparing tools iii

Another tool which is used to show the differences or review history between two revisions is the [git diff] tool.

To know the difference between the index (staging area) and the working copy, use the following command:

Code to review difference between index and working copy

```
git diff
```

To know the difference between the index (staging area) and the last commit/version, use the following command:

Code to review difference between index and last commit

```
git diff --staged
```

Git Inspecting & Comparing tools iv

To show the difference between working copy and last commit/head, use the following command:

Code to review difference between working copy and head

```
git diff HEAD
```

To see the content difference between two branches, use the following command:

Code to see differences between two branches

```
git diff [first-branch] [second-branch]
```

Git Branching i

A branch is a parallel version of your default repository mainly the master branch. It can be used for experimental purpose.

To know in which branch you are working, you can use the [git branch] command:

Code for knowing your current branch

```
git branch
```

Your current branch will have [*] in-front of the branch name.

To create a new branch use the git branch command followed by the new branch name:

Code for creating a new branch

```
git branch [New branch name]
```

Git Branching ii

For navigating to the newly created branch, use the [git checkout] command followed by the name of the branch.

Code for navigating into other branche

```
git checkout [Branch name]
```

To list the complete set of available local and remote branches, use the following command:

Code to list branches

```
git branch -a
```

To delete the branch, use the "git branch -d" followed by the branch name:

Code to delete branch

```
git branch -d [branch name]
```

Git Branching iii

If you want to merge a single branch to a master branch, first go to the master branch then use the [git merge] command followed by branch name.

Code for merging branches

```
git checkout master
git branch --merged
git merge [branch name]
```

Sometimes you want to switch to other branches with out loosing your current work. In this case, run the first command before going to other branches and after returning to the original branch run the second command:

Code for not loosing intermediate work

```
git stash
git stash apply
```

Cloning a remote repository i

Working with remote repositories is one of the primary features of Git. As a result, you may need to download a remote repository into your local machine.

To get a copy of an existing repository, use [git clone] command:

Code for cloning a remote repository

```
git clone [url]
```

It's possible to clone the repository with the different name:

Code for cloning with d/f name

```
git clone [url] new name
```

Cloning a remote repository ii

To see which remote repository you have configured, run the git remote command as follow:

Code for listing configured repositories

```
git remote
```

It's possible to add a new remote repository by introducing a name and url for it:

Code for adding remote repository

```
git remote add [short name] [url]
```

The full address of the cloned repository can be viewed using the following command:

Code to obtain information about the cloned repository

```
git remote -v
```

Cloning a remote repository iii

You can use fetch command if you want to download the latest remote repository to the local machine with out automatically merging it with your work.

Code for cloning with out merging

```
git fetch [remote name]
```

If you want to automatically fetch data from the remote repository and merge it to the local working directory, use the git pull command:

Code to download the latest remote repository

```
git pull origin master
```

Cloning a remote repository iv

To upload the local repository to the remote serve, use [git push] command as follow:

Code for uploading local repository to remote server

```
git push origin master
```

To delete a branch from the local repository use the following command:

Code for deleting a local branch

```
git branch -d [branch name]
```

To delete a branch from the remote repository, use the following command:

Code for deleting remote branch

```
git push origin --delete [branch name]
```

Cloning a remote repository v

To see more information about a particular remote, use the following command:

Code for viewing detail information about remote repository

```
git remote show [name of remote]
```

To change the name of remote repository,use the following command:

Code for renaming remote repository

```
git remote rename [old name new name ]
```

To remove a remote repository, you can use either of the two command:

Code for removing remote repository

```
git remote remove [remote name]
git remote rm [remote name]
```

Cloning a remote repository vi

Sometimes you may want to know why some changes are made such as by whom, when and why, to answer such questions, use the following command:

Code for viewing blame

```
git blame [file name]
```

If you want to create tag to a particular release point, use the following command:

Code for tagging release points

```
git tag -a [version number] -m ["comment about version number"]
```

To list available tags, use the following command:

Code for listing tags

```
git tag
```

Example i

This example is prepared for illustration purpose to introduce the basic Git commands which will help you to work with local and remote repositories. Use your terminal to execute the codes.

Create a new repository named Git_tutorial

```
$ git init Git_tutorial

Initialized empty Git repository in /Git_tutorial/.git/
```

Check files inside the .git directory

```
$ cd Git_tutorial
$ ls .git

branches config description HEAD hooks info objects refs
```

Create a text file nammed [example1.txt] in side Git_tutorial

```
$ touch example1.txt
```

Example ii

Check the status of the file using [git status]

```
$ git status

On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in
what will be committed)

example1.txt

nothing added to commit but untracked files
present (use "git add" to track)
```

Example iii

Make your first edit to the text file and add it to the staging area

```
$ echo This is the first line. > example1.txt
$ git add example1.txt
$ git status

On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file:   example1.txt
```

Example iv

Make your first commit

```
$ git commit -m"First commit"
$ git status

[master (root-commit) 53bc8e4] First commit
1 file changed, 1 insertion(+)
create mode 100644 example1.txt
On branch master nothing to commit, working tree clean
```

Modify the file and make your second commit

```
$ echo This is the second line. >>example1.txt
$ git add example1.txt
$ git commit -m"Second commit"

[master 667a6b2] Second commit
1 file changed, 1 insertion(+), 1 deletion(-)
```

Example v

To view a commit history of a file

```
$ git log

commit 2d3cb8cdfa6646934c60bf9d95d7c4fea9c1ca6b
(HEAD -> master)
Author: Yonas <yonas.mersha14@gmail.com>
Date:   Sun Dec 2 13:52:59 2018 +0300

Second commit

commit 483ffbf239b6dec69a0d21c0d05b0d54d2e7dc92
Author: Yonas <yonas.mersha14@gmail.com>
Date:   Sun Dec 2 13:50:31 2018 +0300

First commit
```

Example vi

To display the changes btween the stage and working copy, first add a third line to the text file with out commiting it. Then use the git diff tool to see the diffrence between files.

```
$ echo This is the third line. >> example1.txt
$ git diff

diff --git a/example1.txt b/example1.txt
index c1c0fb4..9aaeaf9 100644
--- a/example1.txt
+++ b/example1.txt
@@ -1,2 +1,3 @@
This is the first line.
This is the second line.
+This is the third line.
```

Example vii

Add a second file [example2.txt] to the working directory and check the status

```
$ touch example2.txt
$ git add example2.txt
$ git status

On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
modified:   example1.txt
new file:   example2.txt
```

Example viii

Rename the newly added file and check the status

```
$ git mv example2.txt renamed.txt
$ git status

On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

modified:   example1.txt
new file:   renamed.txt
```

Example ix

Delete the newly added file

```
$ rm renamed.txt
$ git rm renamed.txt
```

rm 'renamed.txt'

```
Changes not staged for commit:
(use "git add/rm <file>..." to update what
will be committed)
(use "git checkout -- <file>..." to discard
changes in working directory)

deleted:    renamed.txt
```

Example x

Login into your GitHub account using your username and password. If you don't have account go to the following site https://github.com/login and create a GitHub account. You need to have a valid email address.
Once you have a GitHub account, create a new repository as follow:

- Give a repository name, in this example, `[Git_tutorial]`, you can use your own name.
- Description [optional]
- Choose the public option
- In you want initialize it with readme file [Optional]
- Click on create repository

Example xi

Copy the remote address provided by GitHub. In this example, the repository address provided by GitHub is https://github.com/YonSci/Git_tutorial.git

Adding a remote repository and print information about it

```
$ git remote add origin https://github.com/YonSci/Git_tutorial.git

$ git remote -v

origin https://github.com/YonSci/Git_tutorial.git(fetch)
origin https://github.com/YonSci/Git_tutorial.git (push)
```

Example xii

Send the local repository to the remote repository: here you will be asked [username] and [passwod] for the GitHub account

```
$ git push -u origin master

Username for 'https://github.com': YonSci
Password for 'https://YonSci@github.com':

Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (12/12),1.80 KiB |1.80 MiB/s,done.
Total 12 (delta 0), reused 0 (delta 0)remote:
remote:https://github.com/YonSci/myrepo/pull/new/master
remote:To https://github.com/YonSci/myrepo.git
* [new branch] master -> master
```

Example xiii

Create a new branch named [branch1] in the local directory

```
$ git branch branch1
```

Navigate to the new branch

```
$ git checkout branch1
Switched to branch 'branch1'
$ git branch

* branch1
master
```

Example xiv

Edit the file in the branch1, go to the master branch and merge branch1 to the master branch locally

```
$ echo This is the forth line. >>  example1.txt
$ git add example1.txt
$ git commit -m"comment on branch1"
$ git checkout master
$ git merge branch1

Updating ae6a07f..1c2b224
Fast-forward
example1.txt                    | 2 ++
1 files changed, 2 insertions(+)
```

Example xv

Get the latest version of the the remote repository

```
$ git pull origin master

From https://github.com/YonSci/Git_tutorial
* branch            master     -> FETCH_HEAD
Already up to date.
```

Example xvi

Send changes from the local repository to the remote master repository

```
$ git push origin master

Username for 'https://github.com': YonSci
Password for 'https://YonSci@github.com':

Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3),296 bytes|296.00 KiB/s,done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/YonSci/myrepo.git
9e05f26..6666e9d  master -> master
```

Example xvii

Send changes from the local repository to the remote branch repository

```
$ git push origin branch1

Username for 'https://github.com': YonSci
Password for 'https://YonSci@github.com':

Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'branch1' on GitHub
by visiting:
remote:https://github.com/YonSci/Git_tutorial/pull/new
/branch1
remote:
To https://github.com/YonSci/Git_tutorial.git
* [new branch]      branch1 -> branch1
```

Example xviii

Delete a branch from the remote repository

```
$ git push origin --delete branch1

Username for 'https://github.com': YonSci
Password for 'https://YonSci@github.com':
To https://github.com/YonSci/Git_tutorial.git
- [deleted]         branch1
```

Clone a remote Repository named [Git_tutorial]

```
$ git clone https://github.com/YonSci/Git_tutorial.git

Cloning into 'Git_tutorial'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9(delta 0),reused 9(delta 0),pack-reused 0
Unpacking objects: 100% (9/9), done.
```

Example xix

Clone a remote Repository named [Git_tutorial] with new name

```
$ git clone https://github.com/YonSci/Git_tutorial.git new_name

Cloning into 'new_name'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9(delta 0),reused 9(delta 0),pack-reused 0
Unpacking objects: 100% (9/9), done.
```

Example xx

Get more information about the remote repository

```
$ git remote show origin

* remote origin
Fetch URL: https://github.com/YonSci/Git_tutorial.git
Push  URL: https://github.com/YonSci/Git_tutorial.git
HEAD branch: master
Remote branch:
master tracked
Local branch configured for 'git pull':
master merges with remote master
Local ref configured for 'git push':
master pushes to master (up to date)
```

Further Reading i

For additional materials, tutorials and books visit the following resources:

Official Git site:
http://git-scm.com

Commonly used Git references:

GitHub Training
GIT CHEAT SHEET
https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

Atlassian Git Cheat Sheet
https://www.atlassian.com/dam/jcr:8132028b-024f-4b6b-953e-e68fcce0c5fa/atlassian-git-cheatsheet.pdf

Further Reading ii


M. Mccullouhh
Getting Started With Git, 2016.
http://enos.itcollege.ee/~jpoial/allalaadimised/git/getting-started-with-git.pdf


A. Baire
GIT for Beginners, 2017.
https://people.irisa.fr/Anthony.Baire/git/git-for-beginners-handout.pdf

Book:


S. Chacon & B. Straub
Pro Git
Everything you need to know about Git, 2018.
https://git-scm.com/book/en/v2