

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/340528902>

A Beginner's Guide to Git and GitHub

Research · January 2017

DOI: 10.13140/RG.2.2.20126.89927

CITATIONS

0

READS

4,908

2 authors, including:



Pankaj Mudholkar

Thakur Institute of Management Studies, Career Development & Research, Mumbai, India

25 PUBLICATIONS 23 CITATIONS

SEE PROFILE

A Beginner's Guide to Git and GitHub

Megha Mudholkar¹ Pankaj Mudholkar²

^{1,2}Assistant Professor

^{1,2}Department of Master of Computer Application

^{1,2}Thakur Institute of Management Studies, Career Development & Research, Mumbai, Maharashtra, India

Abstract— Git and GitHub are the development platforms which help developers to host and review code, manage projects, and build software alongside millions of other developers. Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. Git, GitHub, and GitHub Pages are all very closely related. Git is the workflow to get things done and GitHub and GitHub Pages are places to store the work done. Projects that use Git are stored publicly in GitHub and GitHub Pages, so in a very generalized way, Git is what you do locally on your own computer and GitHub is the place where all this gets stored publicly on a server. This article explains basics of Git and Github along with installation guide and some basic commands.

Key words: Git, GitHub, GitHub Pages, Distributed version control system

I. INTRODUCTION

As with many great things in life, Git began with a bit of creative destruction and fiery controversy. The Linux kernel is an open source software project of fairly large scope. For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size). [1]

So, Linus Torvalds designed and developed Git for Linux kernel development in 2005. Git is free software distributed under the terms of the GNU General Public License version 2. The following section explain some background about version control system followed by introduction to git and its installation for windows O.S.

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.[1] The functions of a Version Control System(VCS) :

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.

- Maintains a history of every version.
- Following are the types of VCS:
- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

Git is a Distributed version control system that tracks changes to files in a project over time. Following section explains Git basics, its installation on windows O.S. and Some basic commands like init, clone, add, commit, diff, and log.

II. BASICS OF GIT AND GITHUB

Git is a fast, scalable distributed version control system that tracks changes to computer files and coordinates work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any files. Git Repository maintains information about the changes made in a file, who made the changes, at what time changes were made, notes and comments about the changes made. It typically records what the changes were (what was added? what was removed from the file?), who made the changes, notes and comments about the changes by the changer, and at what time the changes were made. Git is for people who want to maintain multiple versions of their files in an efficient manner and travel back in time to visit different versions without juggling numerous files along with their confusing names stored at different locations. [2]

GitHub is a web hosting service for the source code of software and web development projects (or other text based projects) that use Git. In many cases, most of the code is publicly available, enabling developers to easily investigate, collaborate, download, use, improve, and remix that code. The container for the code of a specific project is called a repository. [2]

A. The Three States of Git:

Git has three main states that your files can reside in: committed, modified, and staged. Committed means that the data is safely stored in your local database. Modified means that you have changed the file but have not committed it to your database yet. Staged means that you have marked a modified file in its current version to go into your next commit snapshot.

This leads us to the three main sections of a Git project: the Git directory, the working directory, and the staging area.

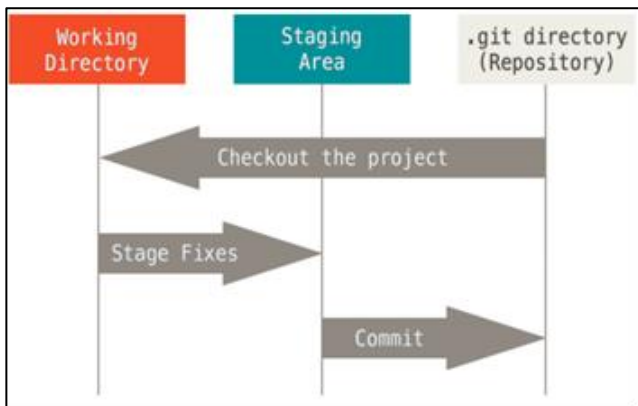


Fig. 1: Working Directory, Staging area and Git Directory

The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

The working tree is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

The staging area is a file, generally contained in your Git directory that stores information about what will go into your next commit. It's sometimes referred to as the "index", but it's also common to refer to it as the staging area.

The basic Git workflow goes something like this:

- Modify files in working tree.
- Stage the files, adding snapshots of them to staging area.
- Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to Git directory.

If a particular version of a file is in the Git directory, it's considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified. [1]

III. INSTALLING GIT ON WINDOWS

There are a lot of different ways to use Git. There are the original command line tools, and there are many graphical user interfaces of varying capabilities. Preferred tool to use Git is Command line because all Git commands run on it. To get start with Git, it should be installed on computer. It can be installed as a package or via another installer, or download the source code and compile it yourself. Following are few ways to install Git on Windows:

- 1) Download Git from Git website : <http://git-scm.com/download/win> OR
- 2) Install GitHub for Windows: <http://windows.github.com>. The installer includes a command line version of Git as well as the GUI. OR
- 3) Install Git from source: <https://www.kernel.org/pub/software/scm/git>. If you want to install Git from source, you need to have the following libraries that Git depends on: curl, zlib, openssl, expat, and libiconv.
- 4) Once the Git is installed on computer, there will be a need to customize Git environment.
- 5) After installation of Git, we have to do a few things to customize our Git environment. These things(set

identity, set Editor, check personal setting, get help) should be only once on any given computer; they'll stick around between upgrades. You can also change them at any time by running through the commands again.

- 6) Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates. Things include set identity, set editor, check personal setting, get help.

1) Setting Identity:

The first thing that should be done after Git installation is to set user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

```
git config --global user.name "Megha Mudholkar"
git config --global user.email
```

meghakunte2000@gmail.com

2) Setting editor:

The next thing that should be done is to configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system's default editor. To use different text editor on Windows such as Notepad or Notepad++ use the following command:

```
git config --global core.editor "C:/Program Files/Notepad++/Notepad.exe"
```

3) Check Setting:

To check settings, use the git config --list command to list all the settings Git can find at that point:

```
git config --list
user.name=Megha Mudholkar
user.email=meghakunte2000@gmail.com
color.status=auto color.branch=auto
color.interactive=auto color.
diff=auto ...
```

4) Get help:

There are three ways to get the manual page (manpage) help for any of the Git commands:

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

For example, you can get the manpage help for the config command by running

```
git help config
```

Following section covers Basic Git commands like init, clone, add, commit, diff, and log.

IV. INITIALIZING A REPOSITORY

First step to work with Git is to initialize a project repository, setting it up so that Git will manage it. To initialize project repository, open up terminal (Git CMD), and in your project directory run the command git init as shown in the screenshot below.

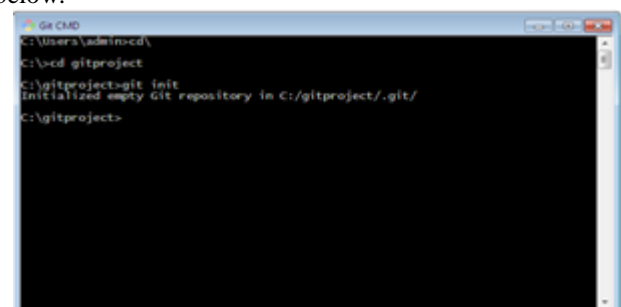


Fig. 2:

A new hidden directory called .git will be created in project directory. This is where Git stores its database and configuration information, so that it can track the project.

V. CLONING A REPOSITORY

Cloning is an alternative way to access a repository. `git clone <repository URL>` command will pull in a complete copy of the remote repository to the local system. Local system contains copy of remote repository. By using local machine, we can work away on it, make changes, staged them, commit them, and push back the changes.

```

Git CMD
C:\Users\admin>cd\
C:\>cd gitproject
C:\gitproject>git clone c:/gitproject/git1
Cloning into 'git1'...
warning: You appear to have cloned an empty repository.
done.
C:\gitproject>
  
```

Fig. 3:

VI. ADDING A NEW FILE

Any file (Python, Ruby, Go, or another language) can be added into the repository. Create a new file, called `Hello.py`, in project directory, and in it, add the following code:

```

Hello.py - Notepad
File Edit Format View Help
print "Hello World"
  
```

Fig. 4:

Save the file, run the command `git status` on the terminal. It will show the current status of working repository. It should look similar to the screenshot below, with `Hello.py` listed as a new, untracked file.

```

Git CMD
C:\gitproject>
C:\gitproject>git status
On branch master

Initial commit

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

    Hello.py

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

Fig. 5:

A. *Work on multiple files, without having to commit all of them -*

Now let's see how you can work on multiple files, without having to commit all of them. Create a second file, called `README.md` (every good project has to have one, right?). In that, add a few details, such as the project name, your

name, and your email address. Run `git status` again. There are two files listed as untracked, as shown below.

```

Git CMD
C:\gitproject>git status
On branch master

Initial commit

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

    Hello.py
    README.md

nothing added to commit but untracked files present (use "git add" to track)
C:\gitproject>
  
```

Fig. 6:

Stage `Hello.py` only, because we're not interested in `README.md` just for the moment. To do that, run `git add Hello.py`. Now run `git status` again, and you'll see `Hello.py` listed as a new file under "Changes to be committed," and `README.md` left in the "Untracked files" area.

```

Git CMD
C:\gitproject>git add Hello.py
C:\gitproject>git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   Hello.py

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

    README.md

C:\gitproject>
  
```

Fig. 7:

VII. MAKING THE FIRST COMMIT

Before making a commit configure the editor, which Git will use when writing commit messages. To do that, run the following command from terminal:

`git config --global core.editor <your app's name>`

```

Git CMD - git commit
C:\gitproject>git config --global core.editor Notepad
C:\gitproject>git commit
unix2dos: converting file C:/gitproject/.git/COMMIT_EDITMSG to DOS format...
  
```

Fig. 8:

Committing in Git is a lot like committing in other version control systems, such as Subversion. You start the process, add a meaningful commit message to explain why the change was made, and that's it, the file's changed. So run `git commit`. This will automatically open up your editor and display the commit template below.

```

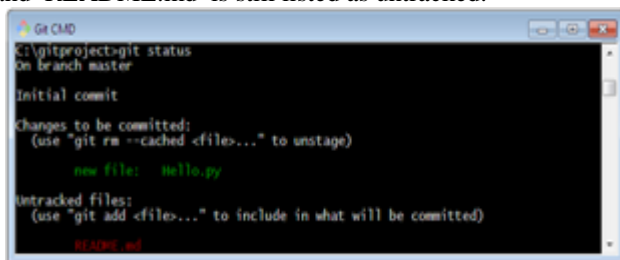
COMMIT_EDITMSG - Notepad
File Edit Format View Help
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   Hello.py
#
# Untracked files:
#   README.md
#
  
```

Fig. 9:

The output in the editor window is same as that of output of git status command in the terminal which helps to view the state of current working repository, which makes it easy to remember what are committing and what are not. A good commit message is composed of two parts: a short message, less than 72 characters long, which briefly states (in active voice) the change being made; and a much longer, optional description, which is separated from the brief description by a newline.

In this case, there's no need to write anything too involved, as it just add the file to the repository. But if the changes made involved a complex algorithm, perhaps in response to a bug filed against the code, it requires giving the fellow developers a good understanding of why the changes made. So add the following simple message "Adding the core script file to the repository," save it, and exit the editor.

Now that the file is committed, run git status again, and README.md is still listed as untracked.



```
Git CMD
C:\gitproject>git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   hello.py

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

    README.md
```

Fig. 10:

VIII. SEEING DIFFERENCES

To review changes in a file, use the command git diff command. Git diff, similar to Linux diff and other diff programs, compares two files and shows the changes the more recent file contains, if any.

IX. VIEWING CHANGE HISTORY

To see your repository or file history over time use git log command in your project repository will show you a list of changes in reverse chronological order. With no further arguments, you'll see the commit hash, the author name and email, a timestamp for the commit, and the commit message.

X. BRANCHING

Branching means you diverge from the main line of development and continue to do work without messing with that main line [1]. Following are some of the branching commands:

- 1) git branch
Lists all local branches in the current repository
- 2) git branch [branch-name]
Creates a new branch
- 3) git checkout [branch-name]
Switches to the specified branch and updates the working directory
- 4) git merge [branch]
Combines the specified branch's history into the current branch
- 5) git branch -d [branch-name]
Deletes the specified branch [3]

XI. CONCLUSION

This paper gives basic understanding of Git as a distributed version control system. This paper also explains three different states of Git project along with WINDOWS installation guide followed by some basic Git command. The basic Git commands includes creating or cloning a repository, making changes, staging and committing those changes, and viewing the history of all the changes the repository has been through. This paper also discusses some basic Branching commands.

REFERENCES

- [1] Scott Chacon, Ben Straub, "Pro Git: Everything you need to know about Git", Experts voice, Second Edition.
- [2] Matthew Setter, "A Beginner's Git and GitHub Tutorial, <http://blog.udacity.com>
- [3] Github-Git Cheat sheet <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>
- [4] <https://try.github.io/levels/1/challenges/1>