Getting started with git

Author:

Waleed Ahmad Mirza

Contributors:

In this tutorial, I will explain how to use Git to manage personal projects. I will talk about the basics of Git here, how to initialize projects, how to manage new and existing files, and how to store your code in the cloud and also relatively complex parts of Git like branching and managing commits between different contributors. Before I start with the tutorial, I will mention some definitions which are commonly used in git context.

Git is a distributed version control system, meaning your local copy of code is a complete version control repository. These fully-functional local repositories make it is easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository. A typical git workflow consist of three stages [1].

1. **Remote repositories** are versions of your project that are hosted on the Internet or network somewhere

2. **Working directory** is the local directory with your source files under git control. Git is tracking the difference between your working directory and remote repository.

3. **Staging area (or index)** Its basically a loading dock which decouples the working directory from the remote repository. This stage contains all the commits get shipped away to the remote repository. The presence of this layer saves the data, you gain a lot more flexibility and control.

An illustration of the workflow is shown in figure 1 [2].

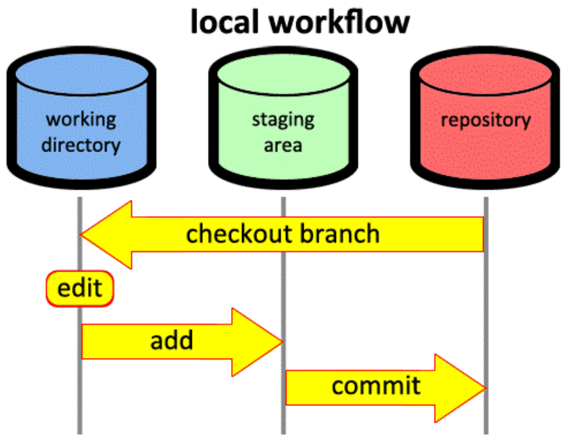

Figure 1: Workflow of **git**

Following sections of the tutorial are meant to acquaint beginners with Git for managing their projects.

1-Intalling Git

Write the following command on bash to install git.

$ ***sudo apt-get install git***

2-Initial Configuration

$ ***mkdir git_practive_project*** // *The first step is to make a directory , that will be used as a working directory to manage and interact the files with the cloud*

$ ***cd git_practive_project***

$ ***git init*** // *initialize Git in a directory*

// *configuring user name and password* //

$ ***git config –global user.name 'xxx'***

$ ***git config –global user.email 'xxx@gmail.com'***

//*Following statement links the working directory to the the online repository* //

// *here it is assumed that the user has already created a repository in Bitbucket account that will serve as version control of the online repository* //

$ ***git remote add*** <***nick name of the repository eg. PracticeRepo*** >

https://WebsiteLinkToTheRepository.git

$ ***git remote*** // to check the repository name

3-Creating file to commit

$ ***echo 'File 1'*** >***file1.txt*** //***creating*** *files in the working directory*

$ ***echo 'File 2'*** >***file2.txt***

$ ***echo 'File 3'*** >***file3.txt***

$ ***git status*** // *to checkout the status of the newly introduced files (red color shows that they are not staged yet, green shows the modified files are staged; as of now the newly created files will be shown in red color* //

4-Moving files to staging area

$ ***git add file1.txt, file2.txt , file3.txt*** *// this adds the files in the staging area*

$ ***git status*** *// now the newly created files will be shows in green color which means that a copy of these files have been moved to the staging area*

5-Committing changes in the repository

$ ***git pull [https://WebsiteLinkToTheRepository.git]****fetch the latest copy of the repository and merge with the existing code*

$ ***git push -u [nick name of repository] master*** *// now check online in the Bitbucket repository to check the newly added file*

6-Downloading repository on local drive

Insert the following command on bash

$ ***git clone https://WebsiteLinkToTheRepository.git***

Using the command a pre-initialized copy of the repository will be fetched from the server.

7-Ignoring files while making a commit in online repository.

Situation Suppose Bill wants to push working directory to the online repository but there are certain files that are personal to him and he wants to keep them locally and exclude them from the list of committed files.

Solution Following sequence of commands should be executed to achieve the aim.

$ echo <***write all the names/extension of files that needed to be exclude*** >>>***.gitignore***

$ ***git add .gitignore***

$ ***git commit -m comment***

$ ***git push [nick name of the repo for eg. PracticeRepo] master***

Now every time Bill will make a commit the *.gitignore* file will not let any of his personal files be committed in the online repository.

8-Getting Old Versions from the Repository

Situation: For instance a programmer has made series of changes in his algorithm from First version to Third version. Each version is being committed in the repository. Later it turns of that the third version is a disaster, therefore now the programmer want to get a fresh copy of the second version in the working directory. Following is supposed to be the work flow of the online repository.

First version - >Second version ->Third version - >Second version

Solution:

1. First checkout reference number of the commit by using the following command.
   $ ***git log***
2. After looking up the reference number of the commit, use the following command.
   $ *git checkout reference number of the commit xxx (* ***just write first few numbers/alphabets)>– <file name >***
   The above statement goes to the commit with reference number xxx and fetch a copy of [file name] from there.
3. Now the following command can be used to check that the new modified file which is different from the one that exist in repository.
   $ ***git status -s***
4. Now push changes in the repository.
   $ ***git commit -m [comment]***

9-Fetching changes from online repository to local drive

Situation:

Suppose programmer A commit changes in the master branch of the remote repository and programmer B wants to fetch these changes in his local repository.

Solution No.1:

1. $ ***git fetch <Repositorys nick name >***

2. $ ***git merge <Repositorys nick name >/ <current-branch>***.

Or just use

1. $ ***git pull <Repositorys nick name >***

Solution No. 2 $ ***git checkout -b <Repositorys nick name >/ <branch >***

git pull contacts the remote repository identified by origin and looks for updates. It fetches any updates and then merges the changes into the target branch. It does not create a new branch.

git checkout -b <branch >origin/ <branch >creates a new branch based on origin/ <branch >, and does not contact the remote repository. It looks at origin/ <branch >as it currently exists in your local repository.

Figure 2 [3] illustrated the flow of commands used so far.

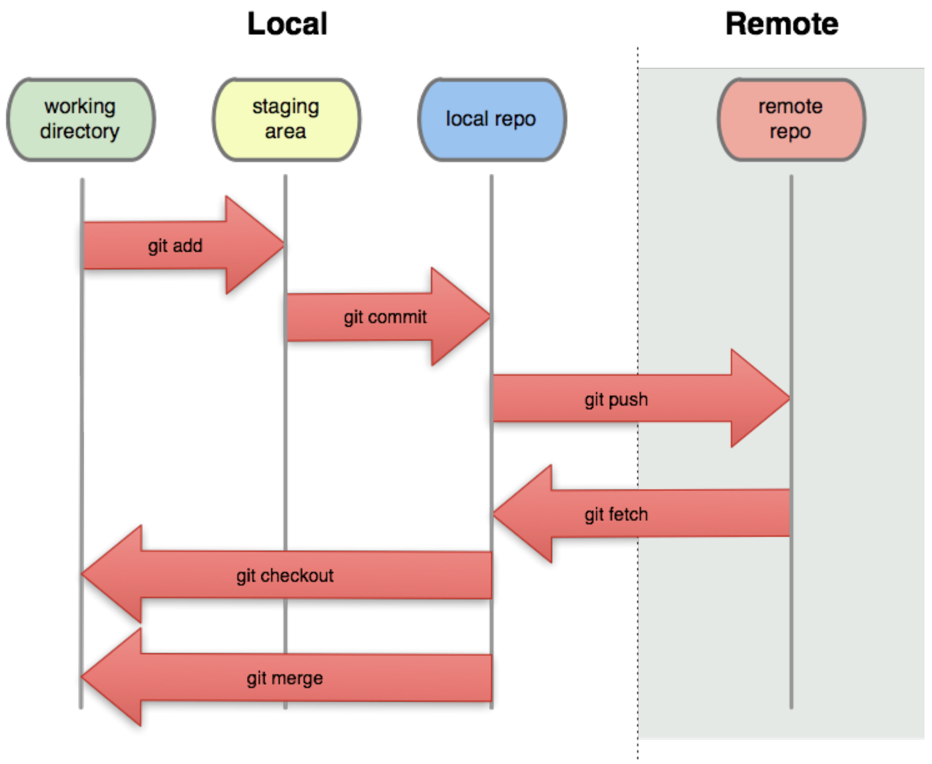

Figure 2: Illustration and work flow of the most important *git commands*.

10-Watching logs of previous commits

Situation Bill now wants to see a brief log of the previous commits on bash terminal.

Solution: $ ***git log –oneline***

11-Reverting Commits

Situation Supposed Bill made some changes in the working directory that he no longer likes and he would like to link his working directory to one of his previous commit.

Solution 1 $ ***git checkout <commit reference No. >***

Solution 2

Solution 2 involves deleting the commit that Bill does not like. It can be done by the following command.

$ ***git revert <commit reference No. >***

12-Identifying changes between the working directory and remote repository

In order to see the difference between the contents of the working directory and the remote repository, use the following command.

$ **git diff** // show differences between working directory and repository

13-Renaming files in the repository

$ ***git mv <old file + extension ><PathToFolder >/ <new file + extension >***

14-Making branches in repository

Situation

Suppose programmer A and B separately want to independently want manage files of the remote repository.

Solution The solution of this issue would be to make two sub branches of the master branch. Following set of commands give an insight into it.

$ ***git branch*** // to identify current branch

* ***master*** // current branch displayed on bash

$ ***git branch <name of the sub branch >*** *// create a sub branch*

$ ***git checkout <name of the sub branch >*** *// switch to the sub branch to work on it*

$ ***git add***. *// the following few commands are to commit changes in the sub branch*

$ ***git commit -m "adding a change from the sub branch"***

15-Merging branches in a repository

Situation Now programmer A think that his sub branch is mature enough to be merged with the master branch. Here is the way to do it.

Solution

$ ***git checkout master*** *// switch to master branch*

$ ***git merge*** *<name of the sub branch >*

$ ***git push <nick name of the repository ><sub branch >***

16-Managing conflicts while merging

<TO BE ADDED LATER >