Andhra Pradesh State Skill Development Corporation

Source Code Management Using Git & GitHub

Introduction to Git

History of Git

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 a 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the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked.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)

What is Git?

Git is a free and open-source distributed version control system developed by Linus Torvalds, designed to handle everything from small to very large projects with speed, efficiency and that facilitates GitHub activities on your laptop or desktop.

Installing Git

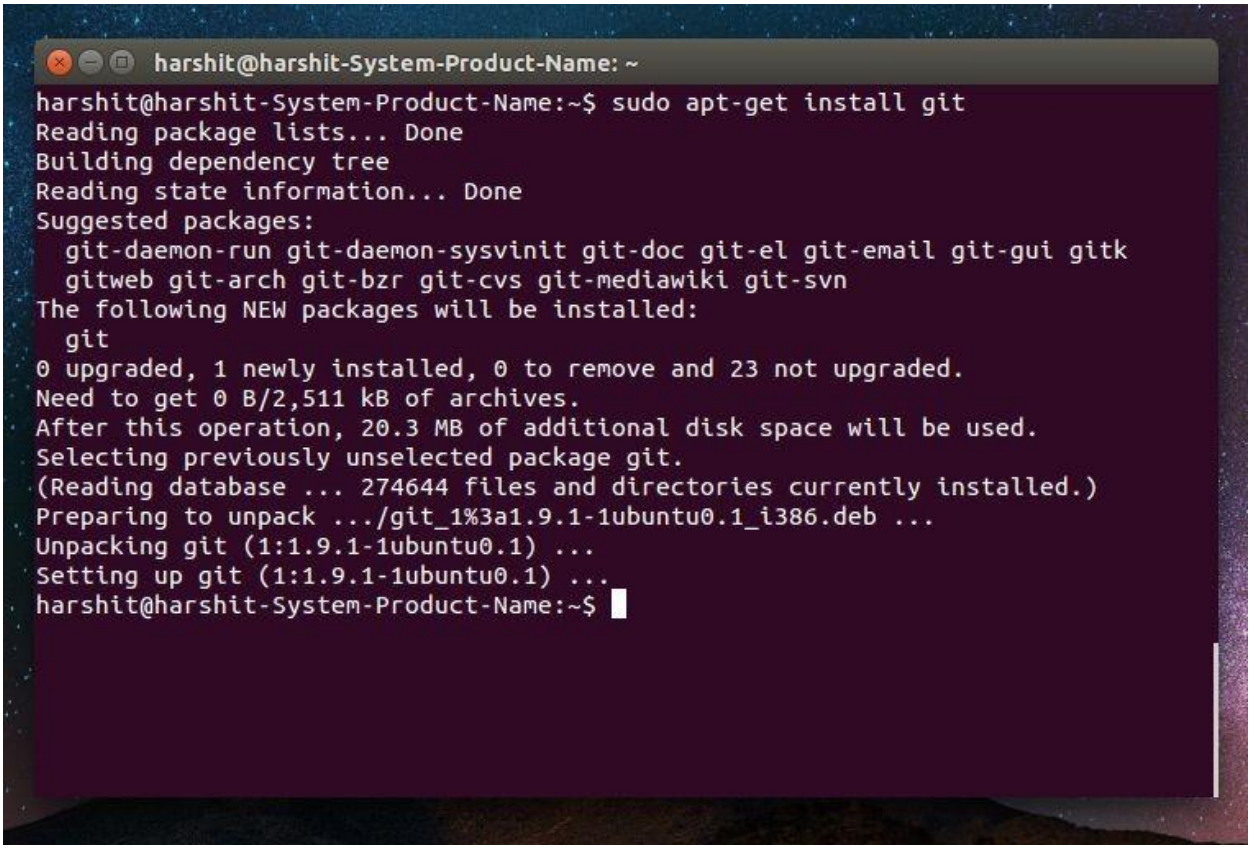
Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you're on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:

sudo dnf installgit

If you're on a Debian-based distribution, such as Ubuntu, try apt:

sudo apt-get installgit



Installing on macOS

There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time.

git --version

If you don't have it installed already, it will prompt you to install it.

Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to https://git-scm.com/download/win and the download will start automatically.

Getting a GitRepository

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository

2. You can clone an existing Git repository from elsewhere.

Initializing a Repository in an Existing Directory

If you have a project directory that is currently not under version control and you want to start controlling it with Git, so let us create a repository i.e, my_folder. you first need to go to that project's directory. Here we should have to initialize this repository with git.

git init

This creates a new subdirectory named .git that contains all of your necessary repository files —a Git repository skeleton. At this point, nothing in your project is tracked yet. Let's create some files like sample1.txt, sample2.txt after creating the files we should move these filesto the staging area and then commit.

To add the particular files to the staging area

git add filename

To add all files to the staging area

git add .

After completion of the staging area next, we should have to commit the files

git commit -m "Commit Message"

Cloning an ExistingRepository

If you want to get a copy of an existing Git repository — for example, a project you'd like to contribute to—the command you need is git clone. This is an important distinction — instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every versionof every file for the history of the project is pulled down by default when you run git clone. In fact, if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned (you may lose some

server-side hooks and such, but all the versioned data would be there.

You clone a repository with the command:

git clone <url>

For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

git clone https://github.com/libgit2/libgit2

That creates a directory named libgit2, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory that was just created, you'll see the project files in there, ready to be worked on or used.

If you want to clone the repository into a directory named something other than libgit2, you can specify the new directory name as an additional argument:

git clone https://github.com/libgit2/libgit2 mylibgit

That command does the same thing as the previous one, but the target directory is called mylibgit.

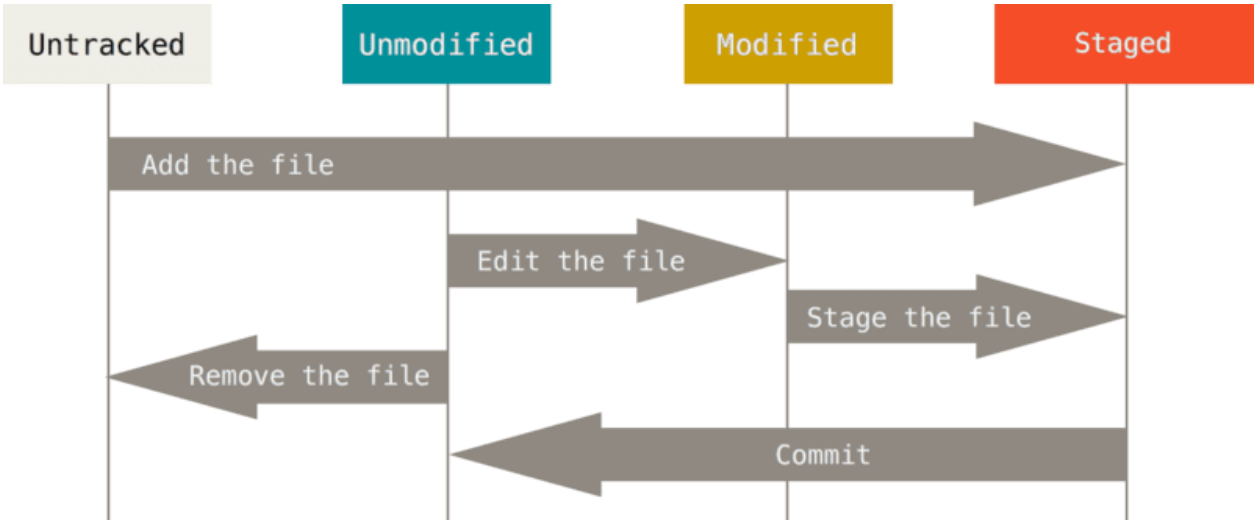
Recording Changes to the Repository

At this point, you should have a bona fide Git repository on your local machine, and a checkout or working copy of all of its files in front of you. Typically, you'll want to start making changes and committing snapshots of those changes into your repository each time the project reaches a state you want to record.

Remember that each file in your working directory can be in one of two states: tracked or untracked. Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.

Untracked files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area. When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

As you edit files, Git sees them as modified, because you've changed them since your last commit. As you work, you selectively stage these modified files and then commit all those staged changes, and the cycle repeats.



Checking the Status of Your Files

The main tool you use to determine which files are in which state is the git status command. If you run this command directly after a clone, you should see something like this:

```
git status
On branchmaster
nothing to commit, working directory clean
```

This means you have a clean working directory; in other words, none of your tracked files are modified. Git also doesn't see any untracked files, or they would be listed here. Finally, the command tells you which branch you're on and informs you that it has not diverged from the same branch on the server. For now, that branch is always master, which is the default; you won't worry about it here. Git Branching will go over branches and references in detail.

Let's say you add a new file to your project, a simple README file. If the file didn't exist before, and you run git status, you see your untracked file like so:

```
touch
README git
status
On branchmaster
Untracked files:
  (use"gitadd<file>..."toincludeinwhatwillbecommitted)

    README
```

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

You can see that your new README file is untracked because it's under the "Untracked files" heading in your status output. Untracked means that Git sees a file you didn't have in the previous snapshot (commit); Git won't start including it in your commit snapshots until you explicitly tell it to do so. It does this so you don't accidentally begin including generated binary

files or other files that you did not mean to include. You do want to start including README, so let's start tracking the file.

The files which are in the untracked stage we should have to move those files to the staging area which the commands is discussed earlier

Viewing the CommitHistory

After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened.

git log

When you run git log in this project, you should get output that looks something like this:

```
git log
commit ca82a6dff817ec66f44342007202690a93763949 Author:
Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Author:
Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6 Author: Scott
Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

By default, with no arguments, git log lists the commits made in that repository in reverse chronological order; that is, the most recent commits show up first. As you can see, this command lists each commit with its SHA-1 checksum, the author's name and email, the date written, and the commit message.

To check the log in reverse order

git log --reverse

To check the log in a short description

git log --oneline

Changing a commitmessage

If a commit message contains unclear, incorrect, or sensitive information, you can amend it locally and push a new commit with a new message to GitHub. You can also change a commit message to add the missing information.

git commit --amend -m "Commit Message"

Delete the Commit

When working with Git you will find that sometimes commits need to be removed as they have introduced a bug or need to be reworked.

If it is the last commit this is very straight forward. Simply run:

git reset HEAD^

This will pop off the latest commit but leave all of your changes to the files intact. If you need to delete more than just the last commit it is possible using rebase this will allow you to remove one or more consecutive commits

gitrebase--onto<branchname>~<firstcommitnumberto remove>
<branchname>~<firstcommittobekept>
<branch name>

Example git log

Number	Hash	Commit Message	Author
1	2c6a45b	(HEAD) Adding public method to access protected	Tom
2	ae45fab	Updates to database interface	Contractor 1
3	77b9b82	Improving database interface	Contractor 2
4	3c9093c	Merged develop branch into master	Tom
5	b3d92c5	Adding new Event CMS Module	Paul
6	7feddbb	Adding CMS class and files	Tom
7	a809379	Adding project to Git	Tom

Using the git log above we want to remove the following commits; 2 & 3 (ae45fab & 77b9b82).

git rebase --onto master~3 master~1 repairCheckout from one version to another version

Basically, if you want to change from one version to another version it is possible with SHA(Secure Hash Algorithm) key, this key is generated when you make a commit.TO change the version the command we will use is:

git checkout <shakey>

Based on the above example

git checkout b3d92c5