Git Quick Guide (1.0)

Rz Li
Sep. 7, 2013

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Programmers use it to keep track of every versions during the development of their code.

This is a guide for very basic git operations, and based on a simple example. For detailed information and inside mechanisms, please read the online Git Book at http://git-scm.com/documentation, especially Chapter One and Two. There's another Git cheat sheet available for your reference: http://overapi.com/git/.

I'm going to show you how to create or clone a Git repository, change the content of files, save the changes, fetch and push with the server, and making branches. Commands are in blue, results will be in red, and notes in orange. It may take about 90 minutes to try out the commands.

---

* I. Local Operations

1. Install Git

To use Git, you need to get Git. I assume everyone has a Linux environment and installation can be as easy as:

```
$ sudo apt-get install git
```

After that, when you type

```
$ git
```

some help information should appear, and you know the package is installed. Though there's a detailed help documentation in the system, Googling your problem can be more helpful in practice.

2. Basic configurations to personalize your Git account

```
$ git config --global user.name "Rongzhong Li"
$ git config --global user.email lir0@wfu.edu
$ git config --global core.editor nano
$ git config --list
```

When you use Git later, Git will remember every changes under your ID.

3. Create a Git repository and add something (You can always hit "TAB" when typing to show possible following commands)

A Git repository is where Git lives in. We can create it or copy it from somewhere else. First of all, let's try something locally.

Make a project folder in a neat directory (which is easy to remember and cd into)

```
$ mkdir gitproj
$ cd gitproj
$ ls
```

Nothing there because it's an empty folder. If you try a command I'll mention later, and probably will be as frequently as $ ls.

```
$ git status
```

It'll tell you this is "Not a git repository (or any of the parent directories)"

So to make it gitted, type
$ git init
$ git status
$ nothing to commit (create/copy files and use "git add" to track)
So Git already remember the directory as a git repository, but it's empty. We can create a text file and add it to Git's memory.
$ echo "Hello Git!" > readme. (You can edit the readme using other editors)
$ ls
$ cat readme
$ git status
$ git add readme
$ git status
I'm not showing the results returned. But notice I type $ git status almost every time after I do something. So observe the different results and it will help understanding how things are changed.
Now save the changes to Git's memory
$ git commit -m "The first line saved in Git"
$ git status
If you keep track of the files status, only the "staged" changes will be committed. And committing is like submitting changes to Git. If you omit -m "The first line saved in Git", Git will open your favorite text editor and ask for some similar tips to mark this commit.

4. Modifications in the directory
Now let's change the readme file.
$ echo "This is my first line on your second line." >> readme
$ cat readme
$ git status
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
modified: readme
It's not staged so use git add to stage the file
$ git add readme
$ git status
# Changes to be committed:
#   modified: readme
Now it's ready for commit
$ git commit -m "add the 2nd line."
$ git status
The change has been accepted by Git.

5. Making branch of your current work and merge it.
Suppose we are satisfied with current version of work, however want to try something fancy. It's ok to create another folder called gitproj_ver1.1. But it's not neat. We can use brach function of Git.
$ git checkout -b borntoleave
Switched to a new branch "borntoleave"
This is shorthand for:
$ git branch borntoleave
$ git checkout borntoleave
Before you are on the branch "master". Now you are working on branch "borntoleave". We can do these things very quickly:

$ echo “Here’s something to be deleted.” > junk.dat
$ ls
$ cat junk.dat
$ git status
$ echo “I added something fancy on branch borntoleave.” >> readme
$ cat readme
$ git status
$ git add junk.dat
$ git status
$ git add readme
$ git status
$ git commit -m “add junk.dat on branch borntoleave.”
$ git status
What I did just now was add another file “junk.dat”, and changed “readme” on branch borntoleave. To check branch information, use
$ git branch -v
How will it affect the work on branch “master”? We can “checkout” the master and check out.
$ git checkout (try pressing “TAB” here and it will show existing branches) master
$ ls
$ cat readme
You can see the “readme” is not modified, and the “junk.dat” is not even there! That’s exactly what they were before you make the branch.
But the changes of branch “borntoleave” is there. We know this because we can merge it into branch “master”.
$ git merge borntoleave
$ ls
$ cat readme
$ git branch --merged
Now the changes has been merged. There’ll be some conflicting cases when projects grow larger, but it’s enough for now.

6. Deleting
In Git, every tracked files are tracked. So deleting and removing needs to be specified, or the changes won’t be saved when you commit.
$ rm junk.dat
$ ls
$ git status
$ git checkout -- junk
$ ls
See, you can still get it back
$ git rm junk.dat
$ ls
$ git status
$ git commit -m “remove junk.dat”
$ git checkout -- junk
Now the “junk.dat” is gone.
You can also delete a branch. Since branch “borntoleave” has been merged, we no longer need it. So simply type
$ git branch -d borntoleave

The branch will be deleted. And don't worry, if the branch has never been merged, Git will inform you and tell you what to do.
Since we've committed several times, we can check the logs.
$ git log
It shows who, when, did what, in order.
---

*II. Remote server

This part might be poorly written and needs Cody's help. You need to be added to "git" group to use the resource in yourusername@greenflash01.cs.wfu.edu:/opt/git/
Before we were working locally, but now we need some group work.

1. Create repository on server using git clone

The server is @greenflash01.cs.wfu.edu. First let me log in and cd to /opt/git/ and clone my gitproj there.

```
$ ssh -X lir0@greenflash01.cs.wfu.edu:
$ password:
$ cd /opt/git/
$ ls
```

You can see something already there. In case you want to submit your repository, I rename it using

```
$ git clone lir0@greenflash31.cs.wfu.edu:~/gitproj rz_gitproj
$ ll (short for ls -l)
d  rwx     r-x      r-x  3 lir0     lir0 4096 Sep  8 21:42 rz_gitproj/
0  123     456      789
 Owner Group Other
```

0 shows rz_gitproj/ is a directory. 123 shows owner can Read, Write, and Xcute. 456 shows group member can Read and Xcute. 789 shows Others can Read and Xcute.
As a group, we want to have write permission to other's work. So you can re-initialize the folder using:(if you are creating a repository directly on the server, you still need this step)

```
$ cd rz_gitproj
$ git init --shared=group
$ ll
drwxrwsr-x 7 lir0 lir0 4096 Sep  8 22:03 .git/
-rw-rw-r-- 1 lir0 lir0    6 Sep  8 21:42 junk
-rw-rw-r-- 1 lir0 lir0   49 Sep  8 21:42 readme
```

So the files can be written by the group now. But if you check the permission of the directory it belongs to,

```
$ ll
drwxr-xr-x  3 lir0     lir0 4096 Sep  8 22:01 rz_gitproj/
```

It doesn't allow group to write. So always remember the "--shared" tag because if you don't include it, others can't change it for you.

2. Pull repository to local

Now the repository is set on server. You can pull its contents to your machine. On your local terminal, type:

```
$ git clone lir0@greenflash01.cs.wfu.edu yourfoldername(optional)
```

I already have the repository on my computer. To simulate your situation I re-cloned it using

```
$ rm -r gitproj
$ git clone lir0@greenflash01.cs.wfu.edu:/opt/git/gitproj
```

If you type

$ git remote -v
It will show you who the origin source is.

3. Making local changes and push them back (Could you check this chapter, Cody?)

Now I want to add a line in readme. You can't make direct changes on the master branch directly. If you do push it back, you'll get errors.
Instead, let's make a new branch and make the changes:
$ git checkout -b new
$ echo "Rz add the 4th line." >> readme
$ git add readme
$ git commit -m "add Rz's line and try to push back"
$ git push origin new
If I'm on the server, I should be able to see your new branch. I can see the new line in the readme. If I think it's a great improvement, I can merge it to the master branch.
$ git checkout (TAB) new
$ cat readme
$ git checkout master
$ git merge new
This part is the same as what you did on your local machine.

4. Shared repository with others
If you're sudoer, you can push your changes to others repository. But if you're a normal user, case is slightly different. We need to use a bare Git repository for sharing work.
In a normal repository, it contains two things. One is the working directory, or working tree. The other is the **.git** documents. **.git** is isolated from our work and only record changes that made to our work. However, if we have a working directory, with the change logs, we are able to get the newest version of work.
A bare repository can be created in many ways. I highly recommend the method below:
In an existing project folder, you should already have it Git-initialized. i.e. you have the **.git** in it.
After checking that, cd out. and
$ git clone --bare --shared gitproj gitproj.git
What it does is basically copying the **.git** directory outside gitproj, and name it **gitproj.git**. And it doesn't contain any work, but only the changes made to the changes.
After that, DON'T FORGET TO MAKE IT SHARED by using
$ chgrp -hR git gitproj.git
On the server, the working directory is NOT supposed to change too often. What we should do is git cloning the original gitproj to our machine(and do this only for the first time). Make changes, locally commit, and then push it to the bare repository, **gitproj.git**. Others should pull the gitproj.git, instead of the whole working directory. Then check out the branch in the new repository. See others changes, and make proper merging.
To test the sharing feature of Git, I recommend you to use the Git repository **gitproj** I created on the greenflash01. After you modified the readme file and commit correctly, If you check the log file,
$ git log
you should see different people's contribution to it. And that will make a sense of teamwork.

These are the basics for Git. I just started to use it, find it very powerful and also confusing sometimes. So this is the first version of our Git Guide and needs your contribution.
To be continued….