Short Git tutorial

Leemoo

2015 年 5 月 8 日

Contents

1 Introduction

Git is a source code version system. Such a system is most useful when you work in a team, but even when you're working alone, it's a very useful tool to keep track of the changes you have made to your code.

2 Configure your git environment

Tell git your name and email:

git config –global user.name ”Your Full Name”

git config –global user.email you@somewhere.com Git stores this information in the ⁀.gitconfig file.

You may also wnat to set the EDITOR environment variable to vim or eamcs, this controls which editor you use to enter log messages.

3 Creating a project

Let's create a new direcotory, ⁀tmp/test1, for our first git project. **cdmkdir** tmp**cd** tmp**mkdir** test1 **cd** t

Put the directory under git revision control:

git init

If you type ll(I'll assume that ll is an alias for **ls -alF**), you will see that there is a .git directory.

Let's start our programming project. Write hello.c with your editor:

include <stdio.h>

int main()

printf(”return 0;

include <stdio.h>int main()printf(”return 0;

Compile and run it:

gcc hello.c./a.out

Let's see what git thinks about what we're doing:

git status

The git status command reports that hello.c and a.out are ”Untracked.” We can have git track hello.c by adding it to the ”staging” area(more on this later):

git add hello.c

Run git status again. It now reports that hello.c is ” a new file to be committed.” Let's commit it:

git commit

Git opens up your editor for you to type a commit message. A commit message should succinctly describe what you're committing in the first line. If you have more to say, follow the first line with a blank line,and then with a more through

multi-line description. For now, type in the following one-line commit message, save, and exit the editor.

```
Added hello-world program.
```

Run git status again. It now reports that only a.out is untracked. it has no mention of hello.c. When git says nothing about a file it means that it is being tracked, and that it has not changed since it has been last committed.
We have successfully put our first coding project under git revision control.

4 Modifying files

Modifying hello.c to print "bye world" instand, and run git status. It reports that the file is "Changed but not updated." This means that the file has been modified since the last commit, but it is still not ready to be committed because it has not been moved into the staging area. In git, a file must first go to the staging ara before it can be committed.
Before we move it to the staging area, let's see what we changed in tht file:

```
git diff
```

Or, if your terminal supports color,

```
git diff –color
```

The output should tell you that you took out the "hello world" line, and added a "bye world" line, like this:

```
-printf("-printf("
```

We move the file to the staging area with git add command:

```
git add hello.c
```

In git, "add" means this: move the change you made to the staging area. The change could be a modification to a tracked file, or it could be a creation of a brand new file. This is a point of confusion for those of you who are familiar with other version control systems such as Subversion.
At this point, git diff will report no change. Our change–from helo to bye–has been moved into satging already. So this means that git diff reports the difference between the staging area and the working copy of the file.
To see the difference between the last commit and the staging area, add –cached option:

```
git diff –cached
```

Let's commit our change. If your commit message is a one-liner, you can skip the editor by giving the message directly as part of the git commit command:

git commit -m "changed hello to bye"

To see your commit history:

git log

You can add a brief summary of what was done at each commit:

git log –stat –summary

Or you can see the full diff at each commit:

git log -p

And in color:

git log -p –color

5 The tracked, the modified, and the staged

A file in a directory under git revision control is either tracked or untracked. A tracked file can be unmodified, modified but mustaged, or modified and staged. confused? Let's try again. There are four possibilities for a file in a git-controlled directory:

beginitemize

untracked

Object files and executable files that can be rebuilt are usualy not tracked.

Tracked, unmodified

The file is in the git repository, and it has not been modified since the last commit. git status says nothing about the file.

Tracked, modified, but unstaged

You modified the file, but didn't git add the file. The change has not been staged, so it's not ready for commit yet.

{textbfTracked, modified, and staged

You modified the ifle, and did git add the file. The changed has been moved to the staging area. It is ready for commit.

The staging area is also called the ”index.”

6 Other useful git commands

Here are some more git commands that you will find useful.

To rename a tracked file:

git mv old-filename new-filename

To remove a tracked file from the repository:

git rm filename

The mv or rm actions are automatically staged for you, but you still need to git commit your actions.

Sometimes you make some changes to a file, but regret it, and want to go back to the version last committed. If the file has not been staged yet, you can do:

git checkout – filename

If the file has been staged, you must first unstage it:

git reset HEAD filename

There are two ways to display a manual page for a git command. For example, for the git status command, you can type one of the following two commands:

git help status**man git**-status

Lastly, git grep searche for specified patterns in all files in the repository. To see all places you called *printf()*:

git grep printf

7 Cloning a project

You created a brand new project in the test1 directory, added a file, and modified the file. But more often than not, a programmer starts with an existing code base. When the code base is under git version control, you can *done* the whole repository.

Let's move up one direcotry, done test1 into test2, and cd into the test2 directory:

cd ..git clone test1 test2**cd** test2

Type ll to see that your hello.c file is cloned here. Morevoer, if you run git

log, you will see that the whole commit history is replicated here. git clone not only copies the latest version of the files, but also copies the entire repository, including the entire commit history. After cloning, the two repositories are indistinguishable.

Let's make some changes –and let's be bad. Edit hello.c to replace printf with printf verbatim| **git** hello.c **git** add hello.c **git** commit -m '" hello world modification - work in progress

Now run git log to see your recent commit carrying on the commit history that was cloned. If you want to see only the commits after cloning:

git log origin.. Of course you can add -p and –color to see the full diff in color:

git log -p –color origin.. Let's make one more modification. Fix the *printf*, and perhaps change the '" bye world" to '"rock my world" while we're there.

vim hello.c **git** add hello.c **git** commit -m "fixed typo now printf rock my world"

Run git log -p –color origin.. again to see the two commits you have made after cloning.

8 Adding a directory into your repository

Enter the original test1 direcotry, create the solution subdirecotry, and add two files to it:

cd ../text1
mkdir solution
cd solution
cp ../hello.c .
echo 'hello:' > Makefile