# Using git, the distributed version control system - Part 2

October 4, 2011

## 1 Introduction

One of the main uses of a version control system is to facilitate collaboration. In a project like the Linux operating system kernel, hundreds of developers will be working in parallel on different parts of the code base. Getting all this work synchronised is a tricky problem; a tool like *git* helps us manage most of the complexity.

## 2 A git workflow for small teams

Suppose you are part of a two-person web development team in a small company. You can create a git repository on a machine in your network and designate it as your *central* repository. Yourself as well as your co-developer then creates copies of this central repository on their own personal desktops. Both of you commit code to your local repository and occasionally synchronise with each other through the central repository. This is one kind of *workflow* which you can implement using *git*. Let's try this out!

## 3 Setting up the central repository

[Note: We will set up all the three repositories on the same machine to simplify things]

Let us create a folder called *proj-server* and intialize a git repo in it[1]:

```
mkdir proj-server
cd proj-server
git init –bare
```

Note that we are calling *git init* with the *–bare* option; this will create what is called a *bare repository.* A bare repository is one which you will use purely for

[1]Please note that there are two hyphens in the *–bare* option to *git init* - it may look appear as a single "-" sign in the printed document

versioning your files; you will not do any kind of development work inside this folder. All the development work will be done in the two folders which both programmers in the team create for their own personal use.

## 4 Setting up the "personal" repositories

The first programmer in the team will do[2]:

```
git clone proj-server proj-dev1
```

The git clone command is used for creating a copy (cloning) an already existing repository.

Similarly, the second programmer will do:

```
git clone proj-server proj-dev2
```

Now, both developers have their own personal work areas; *proj-dev1* for the first developer and *proj-dev2* for the second developer.

Assume that initially, developer 1 is assigned with the task of coding some mathematical functions and developer 2 is assigned with the task of writing some graphics functions. Let's now peek over the shoulders of both these developers and see what they are doing!

## 5 Developer 1: first math function

Developer 1 has written her first math function in a file called *sqr.c* in her personal folder *proj-dev1*:

```
int sqr(int x)
{
    return x*x;
}
```

She will now add and commit this file to her local repository:

```
git add sqr.c
git commit -m 'Implement function for squaring a number'
```

She will now push this change to the central repository from which proj-dev1 was cloned:

```
git push origin master
```

[2]Make sure that you create all three folders, proj-server, proj-dev1 and proj-dev2 at the same level under a common folder. For example, if your login name is rahul, the paths to these three folders should be: /home/rahul/proj-server, /home/rahul/proj-dev1 and /home/rahul/proj-dev2.

The name *origin* is used by git to refer to the repository from which *proj-dev1* was cloned. The change which the first developer has committed to the local repository has now been copied to the central repository. In a real-world scenario, the central repository may be located on a machine on the Net and the two developers may be at two different locations; say one in India and the other one in the US.

## 6 Developer 2: first string function

Developer 2 has written her first string function in a file called *firstchar.c* in her personal folder *proj-dev2*:

```
int firstchar(char s[])
{
    return s[0];
}
```

She adds and commits this to her local repository:

```
git add firstchar.c
git commit -m 'Implement function for returning first char of
a string'
```

She now tries to push this over to the central repository:

```
git push origin master
```

But this doesn't work; git reponds with an error:

```
To /home/recursive/t/proj-server/
! [rejected] master -> master (non-fast-forward)
error: failed to push some refs to '/home/recursive/t/proj-server/'
To prevent you from losing history, non-fast-forward updates were
rejected
Merge the remote changes before pushing again. See the 'Note
about
fast-forwards' section of 'git push –help' for details.
```

The problem is that Developer 1 has pushed some changes to the central repository; if Developer 2 wants to push her changes also to the central repository, she has to first *fetch* and *merge* the contents of the central repository to her local repository:

```
git fetch origin
git merge origin/master
```

After this fetch-and-merge, Developer 2 will have the file *sqr.c* in her working directory. She can now *push* to the central repository:

```
git push origin master
```

At this point, the central repository holds the additions made by both Devloper 1 and Developer 2. Developer 2 has the addition made by Developer 1 in her repository. But Developer 1 is still to get the file *firstchar.c* created by Developer 2; so she will do (in her local repository *proj-dev1*):

```
git fetch origin
git merge origin/master
```

Now, both developers have the changes made by each other in their local repositories!

## 7 Conclusion

What you have seen just now is perhaps the simplest workflow possible using git. Each developer gets changes from the central repsotiory, merges in the changes to the local repository and then executes a push to the central repository. Even in this case, we have not seen things like:

1. What happens if there is a merge conflict? (that is, both developers have made changes to the same file).

2. What happens if the developers have different branches in their local repositories?

Readers are encouraged to consult the excellent book ProGit (available online: http://progit.org/book/ ) to learn more about these things.

Many other more complex collaboration models are possible using git; check out ProGit!