GIT - CREATE OPERATION

In this chapter, we will see how to create a remote Git repository; from now on, we will refer to it as Git Server. We need a Git server to allow team collaboration.

Create New User

```
# add new group
[root@CentOS ~]# groupadd dev

# add new user
[root@CentOS ~]# useradd -G devs -d /home/gituser -m -s /bin/bash gituser

# change password
[root@CentOS ~]# passwd gituser
```

The above command will produce the following result.

```
Changing password for user gituser.
New password:
Retype new password:
passwd: all authentication token updated successfully.
```

Create a Bare Repository

Let us initialize a new repository by using **init** command followed by **--bare** option. It initializes the repository without a working directory. By convention, the bare repository must be named as **.git**.

```
[gituser@CentOS ~]$ pwd
/home/gituser

[gituser@CentOS ~]$ mkdir project.git

[gituser@CentOS ~]$ cd project.git/

[gituser@CentOS project.git]$ ls

[gituser@CentOS project.git]$ git --bare init
Initialized empty Git repository in /home/gituser-m/project.git/

[gituser@CentOS project.git]$ ls
branches config description HEAD hooks info objects refs
```

Generate Public/Private RSA Key Pair

Let us walk through the process of configuring a Git server, **ssh-keygen** utility generates public/private RSA key pair, that we will use for user authentication.

Open a terminal and enter the following command and just press enter for each input. After successful completion, it will create a **.ssh** directory inside the home directory.

```
tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-keygen
```

The above command will produce the following result.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa): Press Enter Only
Created directory '/home/tom/.ssh'.
```

```
Enter passphrase (empty for no passphrase): --------------> Press Enter Only
Enter same passphrase again: ----------------------------> Press Enter Only
Your identification has been saved in /home/tom/.ssh/id_rsa.
Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:
df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 tom@CentOS
The key's randomart image is:
+--[ RSA 2048]----+
| |
| |
| |
|
.
|
| Soo |
| o*B. |
| E = *.= |
| oo==. . |
| ..+Oo
|
+-----------------+
```

ssh-keygen has generated two keys, first one is private $i.\,e.\,,\ id_rsa$ and the second one is public $i.\,e.\,,\ id_rsa.\,pub$.

Note: Never share your PRIVATE KEY with others.

Adding Keys to authorized_keys

Suppose there are two developers working on a project, namely Tom and Jerry. Both users have generated public keys. Let us see how to use these keys for authentication.

Tom added his public key to the server by using **ssh-copy-id** command as given below:

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub gituser@git.server.com
```

The above command will produce the following result.

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Similarly, Jerry added his public key to the server by using ssh-copy-id command.

```
[jerry@CentOS ~]$ pwd
/home/jerry

[jerry@CentOS ~]$ ssh-copy-id -i ~/.ssh/id_rsa gituser@git.server.com
```

The above command will produce the following result.

```
gituser@git.server.com's password:
Now try logging into the machine, with "ssh 'gituser@git.server.com'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
```

Push Changes to the Repository

We have created a bare repository on the server and allowed access for two users. From now on, Tom and Jerry can push their changes to the repository by adding it as a remote.

Git init command creates **.git** directory to store metadata about the repository every time it reads the configuration from the **.git/config** file.

Tom creates a new directory, adds README file, and commits his change as initial commit. After commit, he verifies the commit message by running the **git log** command.

```
[tom@CentOS ~]$ pwd
/home/tom

[tom@CentOS ~]$ mkdir tom_repo

[tom@CentOS ~]$ cd tom_repo/

[tom@CentOS tom_repo]$ git init
Initialized empty Git repository in /home/tom/tom_repo/.git/

[tom@CentOS tom_repo]$ echo 'TODO: Add contents for README' > README

[tom@CentOS tom_repo]$ git status -s
?? README

[tom@CentOS tom_repo]$ git add .

[tom@CentOS tom_repo]$ git status -s
A README

[tom@CentOS tom_repo]$ git commit -m 'Initial commit'
```

The above command will produce the following result.

```
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README
```

Tom checks the log message by executing the git log command.

```
[tom@CentOS tom_repo]$ git log
```

The above command will produce the following result.

```
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530

Initial commit
```

Tom committed his changes to the local repository. Now, it's time to push the changes to the remote repository. But before that, we have to add the repository as a remote, this is a one-time operation. After this, he can safely push the changes to the remote repository.

Note: By default, Git pushes only to matching branches: For every branch that exists on the local side, the remote side is updated if a branch with the same name already exists there. In our tutorials, every time we push changes to the **origin master** branch, use appropriate branch name according to your requirement.

```
[tom@CentOS tom_repo]$ git remote add origin gituser@git.server.com:project.git

[tom@CentOS tom_repo]$ git push origin master
```

The above command will produce the following result.

```
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
```

```
 * [new branch]
master -> master
```

Now, the changes are successfully committed to the remote repository.

```
 * [new branch]
master -> master
```

Now, the changes are successfully committed to the remote repository.