4.2 Git on the Server - Getting Git on a Server (Linux)

**Getting Git on a Server**

Now we’ll cover setting up a Git service running these protocols on your own server.

In order to initially set up any Git server, you have to export an existing repository into a new bare repository — a repository that doesn’t contain a working directory.

This is generally straightforward to do.

In order to clone your repository to create a new bare repository, you run the clone command with the --bare option.

By convention, bare repository directory names end with the suffix .git, like so:

$ git clone --bare my\_project gb.git

Cloning into bare repository 'gb.git'...

done.

You should now have a copy of the Git directory data in your gb.git directory.

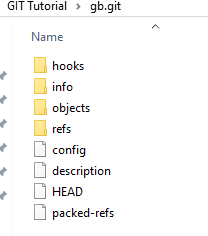


Figura 1 No WA and no .git folder

~~This is roughly equivalent to something like~~

~~$ cp -Rf my\_project/.git gb.git~~

~~There are a couple of minor differences in the configuration file but, for your purpose, this is close to the same thing. It takes the Git repository by itself, without a working directory, and creates a directory specifically for it alone.~~

**Putting the Bare Repository on a Server**

Now that you have a bare copy of your repository, all you need to do is put it on a server and set up your protocols.

* Let’s say you’ve set up a server called git.example.com
* to which you have SSH access, (SSH (Secure Shell) allows you to connect to a different computer and gain terminal access to it despite not physically sitting right in front of it. <https://www.makeuseof.com/tag/beginners-guide-setting-ssh-linux-testing-setup/>)
* and you want to store all your Git repositories under the /srv/git directory.

Assuming that /srv/git exists on that server, you can set up your new repository by copying your bare repository over:

$ scp -r gb.git [user@git.example.com:/srv/git](mailto:user@git.example.com:/srv/git) (Secure Copy from local to remote)

At this point, other users who have SSH-based read access to the /srv/git directory on that server can clone your repository by running

$ git clone user@git.example.com:/srv/git/gb.git

If a user SSHs into a server and has write access to the /srv/git/gb.git directory, they will also automatically have push access.

Git will automatically add group write permissions to a repository properly if you run the git init command with the --shared option. Note that by running this command, you will not destroy any commits, refs, etc. in the process.

$ ssh user@git.example.com

$ cd /srv/git/gb.git

$ git init --bare --shared

You see how easy it is to take a Git repository, create a bare version, and place it on a server to which you and your collaborators have SSH access. Now you’re ready to collaborate on the same project.

It’s important to note that this is literally all you need to do to run a useful Git server to which several people have access — just add SSH-able accounts on a server, and stick a bare repository somewhere that all those users have read and write access to. You’re ready to g, nothing else needed.

In the next few sections, you’ll see how to expand to more sophisticated setups.

This discussion will include not having to create user accounts for each user, adding public read access to repositories, setting up web UIs and more.

However, keep in mind that to collaborate with a couple of people on a private project, all you **need** is an SSH server and a bare repository.

**Small Setups**

If you’re a small outfit or are just trying out Git in your organization and have only a few developers, things can be simple for you.

One of the most complicated aspects of setting up a Git server is **user management**. If you want some repositories to be read-only for certain users and read/write for others, access and permissions can be a bit more difficult to arrange.

**SSH Access**

If you have a server to which all your developers already have SSH access, it’s generally easiest to set up your first repository there, because you have to do almost no work (as we covered in the last section).

If you want more complex access control type permissions on your repositories, you can handle them with the normal filesystem permissions of your server’s operating system.

If you want to place your repositories on a server that doesn’t have accounts for everyone on your team for whom you want to grant write access, then you **must** set up SSH access for them.

We assume that if you have a server with which to do this, you already have an SSH server installed, and that’s how you’re accessing the server.

There are a few ways you can give access to everyone on your team.

~~The first is to set up accounts for everybody, which is straightforward but can be cumbersome.~~

~~You may not want to run adduser (or the possible alternative useradd) and have to set temporary passwords for every new user.~~

A second method is to create a single **git** user account on the machine, ask every user who is to have write access to send you an SSH public key, and add that key to the ~/.ssh/authorized\_keys file of that new **git** account.

At that point, everyone will be able to access that machine via the **git** account.

This doesn’t affect the commit data in any way — the SSH user you connect as doesn’t affect the commits you’ve recorded.

Another way to do it is to have your SSH server authenticate from an LDAP server or some other centralized authentication source that you may already have set up.

As long as each user can get shell access on the machine, any SSH authentication mechanism you can think of should work.

<https://help.ubuntu.com/community/SSH/OpenSSH/Keys>