
This book is available in English.

Full translation available in

български език,
Español,
Français,
Ελληνικά,
日本語,
한국어,
Nederlands,
Русский,
Slovenščina,
Tagalog,
Українська
简体中文,

Partial translations available in

Čeština,
Deutsch,
Македонски,
Polski,
Српски,
Ўзбекча,
繁體中文,

Translations started for

azərbaycan dili,
Беларуская,
فارسی,
Indonesian,
Italiano,
Bahasa Melayu,
Português (Brasil),
Português (Portugal),
Türkçe.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

Chapters ▾

2nd Edition

4.4 Git on the Server - Setting Up the Server

Setting Up the Server

Let's walk through setting up SSH access on the server side. In this example, you'll use the `authorized_keys` method for authenticating your users. We also assume you're running a standard Linux distribution like Ubuntu.

Note A good deal of what is described here can be automated by using the `ssh-copy-id` command, rather than manually copying and installing public keys.

First, you create a `git` user account and a `.ssh` directory for that user.

```
$ sudo adduser git
$ su git
$ cd
$ mkdir .ssh && chmod 700 .ssh
$ touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

Next, you need to add some developer SSH public keys to the `authorized_keys` file for the `git` user. Let's assume you have some trusted public keys and have saved them to temporary files. Again, the public keys look something like this:

```
$ cat /tmp/id_rsa.john.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCB007n/ww+ouN4gSLKssMxXnBOvf9LGt4L
ojG6rs6hPB09j9R/T17/x4lhJA0F3FR1rP6kYBRsWj2aThGw6HXLm9/5zytK6Ztg3RPKK+4k
Yjh6541NYsnEAZuXz0jTTyAUfrtU3Z5E003C4oxOj6H0rfIF1kKI9MAQLMdpGW1GYEIgS9Ez
Sdfd8AcCIicTDWbqLAcU4UpkaX8KyGlLwsNuuGztobF8m72ALC/nLF6JLtPofwFBlgc+myiv
O7TCUSBdLQlgMVOFq1I2uPWQOkOWQAHukEOmfjy2jctxSDBQ220ymjaNsHT4kgtZg2AYYgPq
dAv8JggJICUvax2T9va5 gsg-keypair
```

You just append them to the `git` user's `authorized_keys` file in its `.ssh` directory:

```
$ cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.josie.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.jessica.pub >> ~/.ssh/authorized_keys
```

Now, you can set up an empty repository for them by running `git init` with the `--bare` option, which initializes the repository without a working directory:

```
$ cd /srv/git
$ mkdir project.git
$ cd project.git
$ git init --bare
Initialized empty Git repository in /srv/git/project.git/
```

Then, John, Josie, or Jessica can push the first version of their project into that repository by adding it as a remote and pushing up a branch. Note that someone must shell onto the machine and create a bare repository every time you want to add a project. Let's use `gitserver` as the hostname of the server on which you've set up your `git` user and repository. If you're running it internally, and you set up DNS for `gitserver` to point to that server, then you can use the commands pretty

much as is (assuming that `myproject` is an existing project with files in it):

```
# on John's computer
$ cd myproject
$ git init
$ git add .
$ git commit -m 'initial commit'
$ git remote add origin git@gitserver:/srv/git/project.git
$ git push origin master
```

At this point, the others can clone it down and push changes back up just as easily:

```
$ git clone git@gitserver:/srv/git/project.git
$ cd project
$ vim README
$ git commit -am 'fix for the README file'
$ git push origin master
```

With this method, you can quickly get a read/write Git server up and running for a handful of developers.

You should note that currently all these users can also log into the server and get a shell as the `git` user. If you want to restrict that, you will have to change the shell to something else in the `/etc/passwd` file.

You can easily restrict the `git` user account to only Git-related activities with a limited shell tool called `git-shell` that comes with Git. If you set this as the `git` user account's login shell, then that account can't have normal shell access to your server. To use this, specify `git-shell` instead of `bash` or `csh` for that account's login shell. To do so, you must first add the full pathname of the `git-shell` command to `/etc/shells` if it's not already there:

```
$ cat /etc/shells   # see if git-shell is already in there. If not...
$ which git-shell   # make sure git-shell is installed on your system.
$ sudo -e /etc/shells  # and add the path to git-shell from last command
```

Now you can edit the shell for a user using `chsh <username> -s <shell>`:

```
$ sudo chsh git -s $(which git-shell)
```

Now, the `git` user can still use the SSH connection to push and pull Git repositories but can't shell onto the machine. If you try, you'll see a login rejection like this:

```
$ ssh git@gitserver
fatal: Interactive git shell is not enabled.
hint: ~/git-shell-commands should exist and have read and execute access.
Connection to gitserver closed.
```

At this point, users are still able to use SSH port forwarding to access any host the git server is able to reach. If you want to prevent that, you can edit the `authorized_keys` file and prepend the following options to each key you'd like to

restrict:

```
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty
```

The result should look like this:

```
$ cat ~/.ssh/authorized_keys
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCB007n/ww+ouN4gSLKssMxXnBOvf9LGt4LojG6rs6h
PB09j9R/T17/x4lhJA0F3FR1rP6kYBRsWj2aThGw6HXLm9/5zytK6Ztg3RPKK+4kYjh6541N
YsnEAZuXz0jTTyAUfrtU3Z5E003C4oxOj6H0rfIF1kKI9MAQLMdpGW1GYEIgS9EzSdfd8AcC
IicTDWbqLAcU4UpkaX8KyGlLwsNuuGztobF8m72ALC/nLF6JLtPofwFBlgc+myivO7TCUSBd
LQlgMVOFq1I2uPWQOkOWQAHukEOmfjy2jctxSDBQ220ymjaNsHT4kgtZg2AYYgPqdAv8JggJ
ICUvax2T9va5 gsg-keypair

no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDEwENNMomTboYI+LJieaAY16qiXiH3wuvENhBG...
```

Now Git network commands will still work just fine but the users won't be able to get a shell. As the output states, you can also set up a directory in the `git` user's home directory that customizes the `git-shell` command a bit. For instance, you can restrict the Git commands that the server will accept or you can customize the message that users see if they try to SSH in like that. Run `git help shell` for more information on customizing the shell.

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy