GitLab Quickstart

Jens Lechtenbörger

Summer Term 2023

Contents

1 Notes

This quickstart just provides pointers

- to install Git and
- to setup SSH keys for use with GitLab.

I recommend that you first watch this video (or read its script) for general comments on Git and examples for collaboration with Git.

Besides, you may want to work through this presentation for an introduction to Git.

2 Install and setup Git, use command line

First, install Git. Or, maybe, install GNU/Linux first, then Git.

Perform the First-time Git setup.

Note that some Git commands will invoke a (text) editor in which you then need to type a message. (I suppose that you know how to use editors.) If you do not change the editor as suggested among the First-time Git setup, you may find yourself inside the editor vi (vi reference card). Your instructor prefers GNU Emacs, about which Neal Stephenson wrote "emacs outshines all other editing software in approximately the same way that the noonday sun does the stars. It is not just bigger and brighter; it simply makes everything else vanish."

Our exercises suppose that you use Git on the command line, typically with a command line called Bash. If you are unfamiliar with this environment, maybe see there for basic commands with Bash (e.g., `ls`, `pwd`, `cd`).

Clone any public project of interest to you and inspect its contents. (For GitLab projects, there is a "Clone" button showing the necessary address; other platforms offer similar functionality.) Maybe do this:

```
git clone https://gitlab.com/oer/cs/distributed-systems.git
cd distributed-systems
pwd
ls
```

Note that you just downloaded files into a new directory on your computer. Locate them in your usual work environment (e.g., explorer, finder, command line, Emacs, IDE, ...; the output of `pwd` above may help) and make sure that you can edit, add, and delete files and directories.

3 Create SSH Key Pair

Secure Shell (SSH) is based on asymmetric cryptography and is used by Git in the background to setup secure communication channels with strong authentication.

Briefly, with asymmetric cryptography, keys come in pairs, a private and a public key. While you share your public key, you need to protect your private key so that nobody else can access it. For authentication of Git commands, you register your public key on the server (as instructed below). Whenever you perform a Git operation that requires authentication, the server initiates a challenge-response protocol, in which your client needs to prove that it can access the private key that belongs to the public key registered on the server.

Thus, you need an SSH key pair (consisting of private and public key).

Along the lines of instructions on GitLab, you may want to create a key pair using the algorithm Ed25519 with the following command (in your command line, e.g., Bash, potentially coming with Git for Windows):

```
ssh-keygen -t ed25519 -C "Test key pair for gitlab"
```

Ed25519 may not be available in older SSH implementations. You can leave out option `-t ed25519` to go with the program's default settings, or use `-t rsa -b 4096` for RSA keys with a length of 4096 bits.

This produces a response and asks you where to save your private key:

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
```

Press Enter/Return to **accept the default** (if you change this, Git may not find your key later on).

Then, you are asked for a passphrase.

```
Enter passphrase (empty for no passphrase):
```

Your private key is protected with that passphrase. Thus, if you leave this empty, everyone with access to your machine (or the file with the private key, e.g., a backup) can use that private key and authenticate as you. If you specify a non-empty passphrase, you need to enter it when Git attempts to use your private key.

The command asks you to confirm your passphrase:

```
Enter same passphrase again:
```

Afterwards, more output tells you where the public key is stored (and more, which is not important for our purposes).

```
Your identification has been saved in /home/user/.ssh/id_ed25519.
Your public key has been saved in /home/user/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:0mT81aPfvIYiab64hrwSvIZ0yYlfvh3E90LcmqXFYcE Test key pair for gitlab
The key's randomart image is:
+--[ED25519 256]--+
|         ..      |
|       .  E. .   |
|        + o . o  |
|     . = = o . . |
|  .   + S * .    |
|o oo . + B   . o |
|.=. = o = o   o o|
|.o * + o.= . . ..|
|..o +o+o+oo . .. |
+----[SHA256]-----+
```

(Beyond the scope of this quickstart: You need to enter the passphrase for every Git command that accesses the GitLab server. To avoid this, the passphrase can be cached with `ssh-agent` or `gpg-agent`. E.g., add the commands `eval ‘ssh-agent‘` and `ssh-add` to `.bashrc`.)

4 Register Public Key on GitLab Server

You need to store your **public** key in your profile on wiwi-gitlab.uni-muenster.de. Access of our server is only possible, **after** your university account was activated, about which you will be notified.

Briefly, sign in, go to your “Preferences” via avatar icon on top right, select “SSH Keys”, paste public key (`/home/user/.ssh/id_ed25519.pub` in my case above) into “Key” box, enter a title.

This process (e.g., how to copy&paste the public key) is documented in more detail at `gitlab.com` in the section “Add an SSH key to your GitLab account”. Note that you must register your public key with your GitLab account on **our** server wiwi-gitlab.uni-muenster.de, not on `gitlab.com`.

(Note that similar actions are necessary on other servers such as `gitlab.com` if you want to contribute to projects outside our university.)

Once you registered your public key, continue here to test your SSH setup.

5 Test

If you are not sure whether SSH setup was successful, execute this:

```
ssh -T git@wiwi-gitlab.uni-muenster.de
```

When you connect for the first time, your SSH client does not know the server’s public key yet. To prevent man-in-the-middle attacks (someone else between you and the server might pretend to be the server), it displays the

fingerprint of that key and asks you whether to continue. Type `yes`, if you see this fingerprint among the output:

```
SHA256:yyvXkPVi0q1Gg/HgL0jkUG4hjva0sFPxKZ5PgsHAveE
```

If you see a success message from the previous command, you are good to go. Otherwise, add option `-v` for verbose output:

```
ssh -v -T git@wiwi-gitlab.uni-muenster.de
```

Pay attention to lines about `identity file`. Those ending in `-1` imply that `ssh` tried to access a key that does not exist. At least one must exist. Ask.

License Information