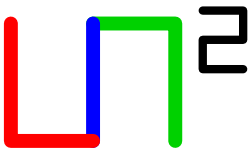

Usage of gitolite on lia server

Repository handling easy

LIA
Universidade de Vigo
Escola Superior de Enxeñaría Informática
E–32004 Ourense

http://lia.ei.uvigo.es
mailto:formella@uvigo.es

Contact: Arno Formella

Reference: M-40100-gitonlia
Version: 1.3
Date: 01/04/2016
Pages: 12

Contents

1 Introduction **3**
1.1 Purpose . . . 3
1.2 Scope . . . 3
1.3 Revision history . . . 3
1.4 Applicable and reference documents . . . 3
1.5 Document overview . . . 4

2 Overview **4**
2.1 Concept . . . 4
2.2 Naming and writing conventions . . . 4
2.3 Machines . . . 4
2.4 Roles . . . 5

3 Prerequisites **5**

4 ssh issues **6**
4.1 Authorization . . . 6
4.2 Key generation . . . 6
4.3 Security aspects . . . 6

5 Repository set-up **7**
5.1 Creating hosting user . . . 7
5.2 Installing gitolite . . . 7
5.3 Administrating the hosting user . . . 7
5.4 Adding repository users . . . 8
5.5 Configuring access rights to repository users . . . 8
5.6 Adding/deleting repositories . . . 9
5.7 Some further actions . . . 10

6 User's point of view **10**
6.1 Becoming a user . . . 10
6.2 Available commands . . . 10
6.3 A note on filenames . . . 11
6.4 A note on directory structure . . . 11
6.5 Trouble shooting . . . 11
6.5.1 Dangling old keys . . . 11
6.5.2 Dangling old hostnames . . . 12
6.5.3 Copying of `hostusr` after system update . . . 12

1 Introduction

This document is part of the software management documents of the research group **LIA**.

1.1 Purpose

This document has been written for a system administrator both to install and to maintain a revision control system for software (and related documents) on a server platform, and for a user of the repositories to access the files. The revision control system used is git and gitolite.

1.2 Scope

This document describes in detail the steps to be performed in order to install a git-server based on gitolite as a revision control system. The repositories on the server can be accessed by different remote or local users with possibly different, individual access rights. The entire process is described as an installation tutorial.

The document does not replace the manuals of the different software tools that are used. It is just a short description of what should be done at least to set-up the revision control system on the server machine and to provide access for different users with individual access rights to its content. Any further information should be taken from the corresponding documentation of the software tools (see applicable documents 1.4).

1.3 Revision history

1.3

- Trouble shooting section re-organized and extended.

1.2

- Document number changed to make it for general use in **LIA**.
- A note on directory structure section added.
- Small extensions and writing revision.

1.1

- Trouble shooting section added.
- This revision history part added.

1.0

This is the initial document.

1.4 Applicable and reference documents

The basic manuals for the different software tools that the interested user and responsible system administrator should read can be found at:

- gitolite documentation: http://gitolite.com/gitolite/gitolite.html
- git documentation: http://git-scm.com/documentation
- ssh documentation: http://www.openssh.org/, for instance.

1.5 Document overview

Section 2 gives an overview of the concept of the repository management, introduces some naming conventions used in this document, and provides some notion of machines and roles users can play.

Section 3 details what previously must be available on the hosting machine, i.e., the server, so you can use gitolite conveniently.

Section 4 tells what parts of ssh play an important role in the set-up.

Section 5 describes how the repository managing system is set-up and maintained.

Section 6 shows what users of the repositories can do and where they get more information.

2 Overview

2.1 Concept

gitolite provides a simple means to provide controlled access to several git repositories for a set of, possibly remote, users.

gitolite needs only one hosting user on the server machine.

gitolite itself uses git to manage the git repositories and the user configuration.

Administration can be shared among different users, playing the role of an administrator, as well.

2.2 Naming and writing conventions

We use **`boldface typewriter`** font in the running text to indicate names for machines or users that must be replaced by the correct names in your installation environment.

We use `typewriter` font for input and output of commands run in a shell. An input command is prefixed either with `xxx@server:` or `xxx@client:` depending whether the command should be introduced on the server or the client machine. The `xxx` stands for the corresponding user to input the command. Output of commands have no prefix.

Sequences of commands are grouped in blocks and there is a preceding line number, so the explaining text can refer to a specific line.

2.3 Machines

We distinguish two logical types of machines:

server machine: Any machine where the centralized revision control system will be installed. We use **`server`** as name for the server machine.

client machine: Any machine from which some user connects to the server machine, either to administrate or to use the revision control system and its repositories. We use **`client`** as name for some client machine.

Note that the server machine and the client machine can be the same physical machine.

2.4 Roles

We distinguish four logical roles of users:

system administrator: A user on the server machine with administrator rights. We use **`admin`** as name (login) for the administration user.

hosting user: The user on the server machine that hosts the repositories. We use **`hostusr`** as name (login) for the hosting user.

repository administrator: A user on any machine with access rights to the hosting user such that he/she can administrate the control revision system. We use **`director`** as name (login) for a repository administrator.

repository user: A user on any machine with access rights to (some) repositories hold by the hosting user. We distinguish here two levels of repository users: leaders that will have read/write access to a set of repositories and workers that will have only read/write to one repository. We use **`leader`** and **`worker`** as names (logins) for some repository users.

Note that actual users may play different roles depending on their access rights granted. The distinction between leaders and workers is just an example, maybe in your project environments more or less levels are convenient, e.g., you might want to add **`observers`** being users with only read access to repositories.

3 Prerequisites

We assume that the software tools ssh, git, and perl are already installed on the server machine. See the current software manuals for gitolite on required cross-compatibility issues for these software tools.

On writing this document, we worked with (output of version information):

```
admin@server: ssh -v
OpenSSH_5.3p1 Debian-3ubuntu7, OpenSSL 0.9.8k 25 Mar 2009

admin@server: git --version
git version 1.7.8

admin@server: perl --version

This is perl, v5.10.1 (*) built for i486-linux-gnu-thread-multi
```

You must have administrator rights on the server machine, at least to add the hosting user.

4 ssh issues

4.1 Authorization

The **server** should allow for password less public key authentication, i.e., in the ssh-daemon configuration file (usually `/etc/ssh/sshd_config` the entries `RSAAuthentication` and `PublickeyAuthentication` should be set both to `yes`.

4.2 Key generation

Use the ssh-utility program `ssh-keygen` to generate a key pair with private and public part:

```
$ ssh-keygen -C rep-user@client
```

You get something similar to the following output and the files are stored on your system:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/rep-user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rep-user/.ssh/id_rsa.
Your public key has been saved in /home/rep-user/.ssh/id_rsa.pub.
The key fingerprint is:
1c:55:e8:81:4b:d9:da:77:17:42:56:86:26:54:f5:27 rep-user@client
The key's randomart image is:
+--[ RSA 2048]----+
|       .++       |
|      .o= .      |
|       * S       |
|    + o . J      |
|     B = S       |
|      * + .      |
|       E o       |
|      . .        |
|                 |
+-----------------+
```

Keep the private key totally secret as user on the client machine. No one ever should have access to that part of the key pair, besides you and your trusted tools.

4.3 Security aspects

gitolite performs just authorization, not authentication, i.e., it is not checked whether the one claiming to be a certain user really is that user, rather access is granted to anyone presenting a valid private key, i.e., a key being a matching key of one of the public keys stored in the system.

Users of the repositories at most get access for a restricted number of actions executable on the repository. So even if the key is compromised, no shell access is granted to the server.

5 Repository set-up

5.1 Creating hosting user

Create the **user** on the server machine and set restrictive access rights on her home directory.

```
1 admin@server: sudo adduser hostusr
2 ...
3 admin@server: sudo chmod 700 /home/hostusr
```

In line 2 you will be asked more data for the user to be added, especially the password will be asked.

Login as user `hostusr` and create a `bin` directory for local user commands and prepend it to the `$PATH` environment variable.

```
1 admin@server: su hostusr
2 ...
3 hostusr@server: cd
4 hostusr@server: mkdir bin
5 hostusr@server: echo "export␣PATH=$HOME/bin:$PATH" >> .bashrc
6 hostusr@server: source .bashrc
```

In line 2 you will be asked the password of the hosting user.

5.2 Installing gitolite

We assume that the repository administrator (here **director**) has already generated an ssh-key pair (see Section 4.2).

Still logged in as hosting user, clone the gitolite software from `github` (https://github.com/sitaramc/gitolite), install the gitolite-executable in the executable–path of the hosting user, copy the public key of a repository administrator, and provide access for at least this repository administrator setting up gitolite:

```
1 hostusr@server: git clone git://github.com/sitaramc/gitolite
2 hostusr@server: gitolite/install -ln /home/hostusr/bin
3 hostusr@server: scp <director.pub> director.pub
4 ...
5 hostusr@server: gitolite setup -pk director.pub
```

where `director.pub` might be any public key file, possibly on a remote machine, of a user who plays the role of an administrator of the hosting user. In line 4 you will be asked the password to access the public key file of the repository administrator on the corresponding machine.

Now you leave the server machine.

5.3 Administrating the hosting user

If you are a user to administrate the hosting user, e.g., your public key file was used in the setup command of gitolite in Section 5.2, then on your client machine (which can be the same

machine as the server machine)—possibly somewhere down a path you like—run:

```
director@client: git clone hostusr@server:gitolite-admin
```

Note that `server` stands for the name of the server machine where the hosting user has been created (see Section 5.1).

From now on you execute all administrative work in the directory `gitolite-admin` as always when working with a git-managed environment. You basically find there two directories `keydir`, which holds the access keys for all users, and `conf`, which holds the configuration set-up.

Once you finished, you commit your changes and push them to the server.

Afterwards, either you delete your working area (and clone again if needed in the future), or you just keep the `gitolite-admin` sandbox for future issues. However, take care to pull before you proceed in the future whenever there exist other administrative users that have might changed the repository inbetween.

5.4 Adding repository users

Add to the key-directory named `keydir` the public keys of the users who should have access to the repositories managed by the hosting user.

The public key files should be organized by hostnames and usernames. As said the repository users are named **`leader`** and **`worker`**, assume they need to have access from three machines where they usually work, say **`leader`** at **`office`** and at **`home`**, and **`worker`** at **`office`** and at **`laptop`**, then their public key files from the corresponding machines should be organized as:

```
keydir/director.pub
keydir/home/leader.pub
keydir/laptop/worker.pub
keydir/office/leader.pub
keydir/office/worker.pub
```

Note that the important information is just a pair (name/key) (in the example, for instance, the user **`worker`** and one of the keys in the files named `worker.pub`), i.e., the user **`worker`** can get access to the repositories whenever the access via ssh uses **`worker`** as name and provides one of the keys stored under `keydir` named `worker.pub`.

5.5 Configuring access rights to repository users

As administrator user, edit the configuration file `gitolite.conf` in the `conf` directory to define the user and repository groups and to grant access rights of users or groups of users to repositories or groups of repositories.

```
# We distinguish three levels of users.
@manager = manager_1 manager_2
@leader = leader_1 leader_2
@worker = worker_1 worker_2 worker_3

# All the managers have complete administrative access.
repo gitolite-admin
     RW+ = @manager
```

```
# All users have complete access to the testing repository.
repo testing
      RW+ = @all

# A possible access rights distribution among the participants.
repo project_1
      RW+ = @leader_1
      R = @leader_2
      RW+ = @worker_1
      R = @worker_2

repo project_2
      RW+ = @leader_2
      R = @leader_1
      RW+ = @worker_2
      RW+ = @worker_3
```

Note that there exists the predefined group `@all` both for users and for repositories (the distinction is clear from the context).

Commit and push the changes to the server.

5.6 Adding/deleting repositories

To add a new repository add the corresponding lines to the configuration file `gitolite.conf` in the `conf` directory, for instance, a third project led by `leader_1`:

```
...
repo project_3
      RW+ = @leader_1
      R = @leader_2
      RW+ = @worker_3
```

Commit and push the changes to the server. The corresponding bare repositories will be generated automatically.

To move an existing repository on your client site to the server, make sure your repository is all correct, create a corresponding bare repository on the server as described above, and push the repository:

```
git push --all hostusr@server:repository-name
git push --tags hostusr@server:repository-name
```

To delete a repository on the server, just remove the corresponding lines for this repository from the configuration file `gitolite.conf` in the `conf` directory. However note: the data is still not deleted, you just have removed the access rights, so no one can read/write the old repository.

Commit and push the changes to the server.

If you really want to remove the data, you must connect as hosting user to the server and remove the correponding directory tree on the server.

5.7 Some further actions

To rename an already existent repository you need direct access to the hosting user. Perform two actions in that order:

1. Rename the repository accordingly on the server as hosting user.
2. Replace in `gitolite.conf` all ocurrences of the old name by the new name as administrative user (don't forget to commit and push).

Note that this does not work for the administration repository `gitolite-admin`.

To import an already existent repository into the set of gitolite repositories clone the repository as bare repository acting as hosting user. Set the file permissions accordingly (at least read/write permissions for the hosting user, which is usually the case when you clone). Run `gitolite setup` as hosting user. Make the appropriate modifications in the `gitolite.conf` file as administrative user (don't forget to commit and push).

6 User's point of view

6.1 Becoming a user

A potential user of a repository on the server machine needs to send his/her public key to a repository administrator who will set-up the access rights for this user on the server machine (see Section 5.4).

Note that the access to the repository on the server machine is given to the person/service knowing the corresponding private key, hence, the system is at most as secure as the protection of the private key on the user's side.

6.2 Available commands

The user does not get a shell access on the server, rather a restricted set of commands is provided.

command	**description**
`ssh hostusr@server info -h`	retrieving of information of accessible repositories
`ssh hostusr@server perms -h`	handling of permissions
`ssh hostusr@server desc -h`	handling of descriptions

Table 1: User commands (help format)

Normally, a user performs normal git actions:

```
1 git clone hostusr@server:some-repo
2 ...
3 < work with data >
4 git add ...
5 git commit ...
6 ...
```

```
git push
...
git pull
...
etc.
```

For more information see the documentation of git and gitolite as referenced in Section 1.4.

6.3 A note on filenames

The repositories on the server are probably accessed by users working on machines with different operating systems (e.g., GNU/linux, Windows, and MacOS) which handle filenames differently.

To avoid any problems when moving files from one machine to another it is strongly recommended to use filenames that are handled identically on all machines. Therefore, you should use only names that comply to the following rules:

1. The filename consists only of characters from the following sets `[A-Za-z0-9]`, i.e., letters and digits, and as special characters `[._-]`, and nothing more (Note the `[]` are meta-symbols and not included!).
2. The hyphen `[-]` is never used as first character.
3. The dot `[.]` is never used as last character.
4. There is at most one dot `[.]` present in the name.
5. Within the same directory there appear never two files that distinguish only in lowercase/uppercase.

If there is a need to use filenames within the working directory that do not comply to the rules, these files should be excluded with the help of the corresponding `.gitignore` file (one example are for instance editor backup files ending in `[~]`).

Symbolic or hard links are not handled consistently on all operating systems either, so it is recommended to exclude symbolic or hard links whenever the users work on different operating systems.

6.4 A note on directory structure

Before starting using centralized repositories within a group of developers you should agree on a consistent philosophy how to organize and name the directories in your working area. Special case should be taken not to stores intermediate files in the repository; such files should be excluded by means of the `.gitignore` files. Note that these files can be placed in any subdirectory and the effect is accumulative along the path.

6.5 Trouble shooting

6.5.1 Dangling old keys

Whenever you try to access the repository on the server (either as administrator or user) and you get a message like

```
Agent admitted failure to sign using the key.
```

your ssh-daemon might still use for some reasons an old key entry. Try to remove all (or only some) keys from the ssh-daemon and add the correct one, i.e.,

```
1 ssh-add -D
2 ssh-add
```

The first line removes all keys in use. After the second line, you will be asked for the pass-phrases of the keys to use.

6.5.2 Dangling old hostnames

If the server hardware is changed, the client will notice the change and is asked to add or modify the keys. If you trust the new server identity, you should remove from your `known_hosts` file under `.ssh` the old server keys and—when connecting–the keys for the new server are added.

The command, with `server` replaced by the server host name,

```
1 ssh-keygen -H -F server
```

lists the line of the `known_hosts` file where the key of the server is located. Remove that line from the `known_hosts` file. Repeat the process for the IP address (each trusted host has two lines in the `known_hosts` file, one for the name, one for the IP address).

6.5.3 Copying of `hostusr` after system update

If you suffer a system crash or simply install new hardware on the server, you might run into the issue of re-installing the **`hostusr`** on the server machine.

Assume that you have created the **`hostusr`** on the new system and have copied all old files from the **`hostusr`** into her home directory. Make sure that ownership and groupness are set correctly for all files. For security reasons and update policy the configuration files (dot-files) of the shell and the `.ssh` directory should not be copied, and therefore some more work is necessary.

As **`hostusr`** the following commands (part of Section 5) should be re-executed:

```
1 hostusr@server: echo "export␣PATH=$HOME/bin:$PATH" >> .bashrc
2 hostusr@server: source .bashrc
3 hostusr@server: gitolite/install -ln /home/`whoami`/bin
4 hostusr@server: gitolite setup -pk director.pub
```

which will re-establish the softlink in the `bin`-directory and create new ssh-keys.