Basic Git Home "Remote" Server Using SSH With Gitolite

Mark G.

January 13, 2024

An installation and configuration of a source code control system (SCCS). The SCCS uses the Secure Shell (SSH) protocol, the Git version control system (VCS), and a software package named Gitolite. A user of the system provides a public key that is tied to configured identities for the purposes of access control to git repositories. The repositories configured on this server are intended to be used as remotes, in terms of how git defines a remote.

This demonstration is hosted on a FreeBSD 13.x operating system running on a raspberry pi 4.

Files for this presentation can be found at: Presentations for VicPiMakers.

Contents

1 Introduction

For quite a few years, I used a number of inefficient methods for keeping my documents and code files synchronized between my various laptops and desktop workstations.

Some of the methods included:

- Copying to USB drives and plugging them into the computers that needed updated copies of the files.
- Using fancy programs like `rsync` and `syncthing`.
- Using SSH's secure copy program `scp`.

These ended up not working very well for one reason or another.

Why not just stick to one development system only? My main personal reason is to vary the ergonomics of my computing enough to eliminate repetitive stress injuries.

My latest solution is to use a source code control system (SCCS), also known as a version control system (VCS). There were a few to choose from, the main two being `subversion` and `git`. I had experience with `subversion`, but settled on `git` because it is more modern and has all the features I need.

1.1 Why Gitolite?

The choices for a shared backend for a `git` server are numerous. Some of these choices are:

- Github - famously at https://github.com. I disagree with their terms of service.
- Gitlab (Community Edition) - a contender found at https://gitlab.com/rluna-gitlab/gitlab-ce. This solution is totally too large for my modest needs.

So I chose `gitolite` since it has these simple features:

- Can easily be installed on hardware that I control.
- Multiuser.
- Allows granular repository access control.
- Uses SSH keys. SSH keys are only usable for git access, not shell or login.

Home page: https://gitolite.com/gitolite/index.html

2 Remote Host System Configuration

Note: I will try and identify commands run on the workstations by using a `root@client` and `user@client` command prompt. The server commands will have prompts of `root@server` and `user@server`.

In some cases, there will be prompts with `mv@git` and `root@git`. These prompts are the administrative user (`mv`), and the `root` user on the remote server. They are equivalent to `user@server` and `root@server`, respectively.

Assuming you've just installed your favourite operating system, login on the console as the `root` user to configure it.

Delete the default `freebsd` user:

```
root@think:~ # rmuser
Please enter one or more usernames: freebsd
Matching password entry:

freebsd:$6$...:5002:5002::0:0:FreeBSD:/home/freebsd:/bin/tcsh

Is this the entry you wish to remove? y
Remove user's home directory (/home/freebsd)? y
Removing user (freebsd): mailspool home passwd.
```

You can run the `bsdconfig` command to setup various parts of the system, including changing the `root` user's password and setting network configuration.

2.1 Remote Host Admin User

We need a normal, unprivileged user for administering the remote server. Use your operating system's `adduser` command and create this user:

```
root@server:~ # adduser
Username: mv
Full name: Remote Admin
Uid (Leave empty for default):
Login group [mv]:
Login group is mv. Invite mv into other groups? []: wheel
Login class [default]:
Shell (sh csh tcsh) [sh]: tcsh
Home directory [/home/mv]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
```

```
Username   : mv
Password   : *****
Full Name  : Remote Admin
Uid        : 5002
Class      :
Groups     : mvg wheel
Home       : /home/mv
Home Mode  :
Shell      : /bin/tcsh
Locked     : no
OK? (yes/no): yes
adduser: INFO: Successfully added (mvg) to the user database.
Add another user? (yes/no): no
Goodbye!
```

Note: in FreeBSD, it is important to add the user to the `wheel` group, as shown above. This will allow the administrative user, `mv` in this case, to use the `su -` command to become the `root` user. Other linux type systems use the `sudo` command when necessary.

Also, you can choose a shell other than `tcsh` if you want. On FreeBSD, you'll have to install `bash` before you create this user if you want to use that shell (`pkg install bash`).

2.2 SSH Server and System Startup

This system will listen for SSH connections on tcp port 22. You can change the port in the `/etc/ssh/sshd_config` file if you like, just be sure to update any `~/.ssh/config` entries with the `Port` keyword.

At system setup, we will briefly allow password authentication to copy SSH public keys for the administrative user (YOU) and the public key for the `gitolite-admin` repository.

2.2.1 System Startup

We only need to start the Secure Shell (SSH) service. We also set static internet protocol addresses as needed.

```
root@server:/ # cat /etc/rc.conf

# Add to DNS and /etc/hosts
hostname="git.vinnythegeek.ca"

sshd_enable="YES"
```

```
# IPv4
ifconfig_genet0="inet 10.24.7.22 netmask 255.255.255.0"
defaultrouter="10.24.7.1"

# Add IPv6 config here as needed.
#ifconfig_jail60_ipv6="inet6 fdea:557c:9747:1060::2443 prefixlen 64"
#ipv6_defaultrouter="fdea:557c:9747:1060::6"

# Keep tmp tidy by emptying on startup
clear_tmp_enable="YES"
# do not open any network sockets for syslogd
syslogd_flags="-ss"
# leave sendmail off
sendmail_enable="NONE"
# do not start the DNS caching server
local_unbound_enable="NO"
```

Check that the `sshd` service starts correctly.

```
root@server:/ # service sshd start
Performing sanity check on sshd configuration.
Starting sshd.
```

Success!

2.3 SSH Key Pairs for the Git Client and Admin Users

Back on your workstation (i.e. the git client system) we'll create a number of SSH key pairs using `ssh-keygen`. Make sure to place a passphrase on these keys (you can probably use the same passphrase if you like).

The key pairs we create will be:

1. A key pair for administering the system.

   ```
   user@client:~ ssh-keygen -t ed25519 -f ~/.ssh/git_remote_admin_user \
                              -C "Administrative user for git remote server"
   Generating public/private ed25519 key pair.
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /home/mv/.ssh/git_remote_admin_user
   Your public key has been saved in /home/mv/.ssh/git_remote_admin_user.pub
   ```

   Copy the key to the remote server using `ssh-copy-id`:

```
user@client:~ ssh-copy-id -i ~/.ssh/git_remote_admin_user mv@git.vinnythegeek.ca
```

Let's check the remote server:

```
$ ssh mv@10.24.7.22
(mv@10.24.7.22) Password for mv@git.vinnythegeek.ca:

mv@git:~ % ls -la .ssh
total 12
drwx------  2 mv  mv  512 Jan 12 21:46 .
drwxr-xr-x  3 mv  mv  512 Jan 12 21:46 ..
-rw-------  1 mv  mv  123 Jan 12 21:46 authorized_keys
```

The `ssh-copy-id` command created the `.ssh` folder for us, with the correct permissions and placed the public key into the `auhtorized_keys` file.

2. A key pair for accessing the `gitolite-admin` repository as the `gitolite` administrator.

```
user@client:~ ssh-keygen -t ed25519 -f ~/.ssh/gitolite_admin_user \
                         -C "Gitolite admin user"
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mv/.ssh/gitolite_admin_user
Your public key has been saved in /home/mv/.ssh/gitolite_admin_user.pub
```

We do not use `ssh-copy-id` for this key, since it will be used in the `gitolite setup` command that follows later in this document. For now we just copy it to the admin user's, `mv` in this example, home directory on the remote server.

```
user@client:~ scp ~/.ssh/gitolite_admin_user.pub mv@git.vinnythegeek.ca:
(mv@10.24.7.22) Password for mv@git.vinnythegeek.ca:
gitolite_admin_user.pub
```

Note: we append `.pub` to the key name so as to copy the actual public key portion. `ssh-copy-id`, which we used for the first key takes care to do the right thing. Here we must make sure we copy only the public key.

Check the copy results on the server:

```
mv@git:~ % ls -l
total 4
-rw-r--r--  1 mv  mv  101 Jan 12 21:54 gitolite_admin_user.pub
```

We'll use that key in `gitolite setup`.

3. A key pair as a normal git user to access our git repositories for the usual `git push/pull` activities. This key pair can be shared among the various workstations

that you use to do work on. These are the same workstations on which you want your work synchronized.

```
user@client:~ ssh-keygen -t ed25519 -f ~/.ssh/git_remote_user_mv \
                        -C "User for git remote repositories"
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mv/.ssh/git_remote_user_mv
Your public key has been saved in /home/mv/.ssh/git_remote_user_mv.pub
```

You need a separate key for at least the `gitolite-admin` repository, but the remote server admin user and the 'normal' repository client user could probably use the same keys.

I prefer to use separate keys for separate functions.

2.3.1 Client SSH Config Settings

To match our generated keys, we setup the following `.ssh/config` entries:

```
Host gitremote
 User mv
 AddKeysToAgent yes
 HostName git.vinnythegeek.ca
 AddressFamily inet
 Port 22
 IdentityFile ~/.ssh/git_remote_admin_user

Host git
 User git
 AddKeysToAgent yes
 HostName git.vinnythegeek.ca
 AddressFamily inet
 Port 22
 IdentityFile ~/.ssh/git_remote_user_mv

Host gitolite-vp
 User git
 HostName git.vinnythegeek.ca
 Port 22
 IdentityFile ~/.ssh/gitolite_admin_user
```

2.3.2 Test SSH Keys and Disable Passwords

Login to the remote server using your admin key to ensure it works:

```
user@client:~ $ ssh gitremote
Enter passphrase for key '/home/mv/.ssh/git_remote_admin_user':
Last login: Fri Jan 12 21:46:44 2024 from 192.168.60.59
mv@git:~ %
```

Using the `.ssh/config` file entry `gitremote` will ensure that the private key listed in the `IdentityFile` statement is tried.

Once you can login with this key pair, become the `root` user:

```
mv@git:~ % su -
Password:
root@git:~ #
```

Now turn off password authentication in the SSH server by setting /checking these lines in the `/etc/ssh/sshd_config` file. I had to set `KbdInteractiveAuthentication yes` to `no`. On FreeBSD, `PasswordAuthentication` is already set to `no`.

```
# Change to yes to enable built-in password authentication.
# Note that passwords may also be accepted via KbdInteractiveAuthentication.
#PasswordAuthentication no
#PermitEmptyPasswords no

# Change to no to disable PAM authentication
KbdInteractiveAuthentication no
```

While still logged in, restart the SSH server:

```
root@git:~ # service sshd restart
Performing sanity check on sshd configuration.
Stopping sshd.
Waiting for PIDS: 1134.
Performing sanity check on sshd configuration.
Starting sshd.
```

Now we'll try and login without using our `gitremote` entry.

```
user@client:~ $ ssh mv@10.24.7.22
mv@10.24.7.22: Permission denied (publickey).
```

As an aside, the key pair for `gitremote` was stored in my `ssh-agent` instance, since I have `AddKeysToAgent yes` in the `.ssh/config` entries. In order to properly test password authentication is disabled, we had to remove that key from the agent.

First list the keys, then delete the key by name:

```
user@client:~ $ ssh-add -l
```

```
2048 SHA256:rXkLvSxbedydWyHwWq6GENirka5aRQznd1p6LpcbqFU pollbox_admin_gitolite (RSA)
256 SHA256:QaJxdrjBX9h3EfKOwxWVunz3wNRE5NG66eZxldBITI4 Administrative user for git remote server

user@client:~ $ ssh-add -d .ssh/git_remote_admin_user
Identity removed: .ssh/git_remote_admin_user ED25519 (Administrative user for git remote server)
```

The next time we login with the `gitremote` config entry, the key will be added back to the agent.

3 Gitolite / Git Installation

Login to the remote server using the admin account and key you just created, and become the `root` user.

```
user@client:~ $ ssh gitremote
Enter passphrase for key '/home/mv/.ssh/git_remote_admin_user':
...
mv@git:~ %
mv@git:~ % su -
Password:
root@git:~ #
```

We install the `git` and `gitolite` packages using the freebsd `pkg` command.

Other operating systems have their own package management commands such as `apt`, which are left as an exercise to the student.

3.1 Client Git

On all the systems that you wish to work on (i.e. your workstations), you'll need to install `git`. You don't need to install `gitolite` on the workstations.

```
root@client:~ # pkg update

root@client:~ # pkg install git

New packages to be INSTALLED:
...
git: 2.39.1
...

Number of packages to be installed: 37

The process will require 238 MiB more space.
44 MiB to be downloaded.
```

```
Proceed with this action? [y/N]: y
...
===> Creating groups.
Creating group 'git_daemon' with gid '964'.
===> Creating users
Creating user 'git_daemon' with uid '964'.
[37/37] Extracting git-2.39.1: 100%
=====
...
Message from git-2.39.1:

--
If you installed the GITWEB option please follow these instructions:

In the directory /usr/local/share/examples/git/gitweb you can find all files to
make gitweb work as a public repository on the web.

All you have to do to make gitweb work is:
1) Please be sure you're able to execute CGI scripts in
/usr/local/share/examples/git/gitweb.
2) Set the GITWEB_CONFIG variable in your webserver's config to
/usr/local/etc/git/gitweb.conf. This variable is passed to gitweb.cgi.
3) Restart server.


If you installed the CONTRIB option please note that the scripts are
installed in /usr/local/share/git-core/contrib. Some of them require
other ports to be installed (perl, python, etc), which you may need to
install manually.
```

We switch over to the remote server to install `gitolite` and `git`.

3.2 Remote Host Server Gitolite and Git

We install `gitolite` on the system that we chose to be our remote (a.k.a. "cloud") server. In my case this server is in my home, but acts as a central point of storage for my git repositories. You can use a third party hosting service for this server if you like.

Installing `gitolite` pulls in `git` so all we need to do is install `gitolite` as follows:

```
root@git:~ # pkg install gitolite
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
All repositories are up to date.
```

```
The following 37 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        ...
        git: 2.42.0
        gitolite: 3.6.12,1
        ...

Number of packages to be installed: 37

The process will require 239 MiB more space.
43 MiB to be downloaded.

Proceed with this action? [y/N]:y
...
[36/37] Installing git-2.42.0...
===> Creating groups.
Creating group 'git_daemon' with gid '964'.
===> Creating users
Creating user 'git_daemon' with uid '964'.
[36/37] Extracting git-2.42.0: 100%
[37/37] Installing gitolite-3.6.12,1...
[37/37] Extracting gitolite-3.6.12,1: 100%
...
=====
Message from git-2.42.0:

--
If you installed the GITWEB option please follow these instructions:

In the directory /usr/local/share/examples/git/gitweb you can find all files to
make gitweb work as a public repository on the web.

All you have to do to make gitweb work is:
1) Please be sure you're able to execute CGI scripts in
   /usr/local/share/examples/git/gitweb.
2) Set the GITWEB_CONFIG variable in your webserver's config to
   /usr/local/etc/git/gitweb.conf. This variable is passed to gitweb.cgi.
3) Restart server.


If you installed the CONTRIB option please note that the scripts are
installed in /usr/local/share/git-core/contrib. Some of them require
other ports to be installed (perl, python, etc), which you may need to
```

```
install manually.
=====
Message from gitolite-3.6.12,1:

--
Final gitolite setup instructions:

Any ssh user can be a gitolite provider. Simply run the following command as
the user:

/usr/local/bin/gitolite setup -pk /path/to/admin.ssh.key.pub

This will setup up the configuration files and repositories for gitolite.

The admin ssh key allows full access to the gitolite-admin repository where
additional users and repositories can be configured.

By default, the git user is created for use by gitolite.

A quick-install guide can be found in:

/usr/local/share/doc/gitolite/README.markdown
```

The post installation message reminds us of the next steps for setting up `gitolite`. We'll implement them in the sections that follow.

Optionally, capture the package list after installation:

```
root@server:~ # pkg info > pkg.list.for.git.20240113
```

To check the versions you have installed, you can run this command:

These are the packages installed by or with git and gitolite:

```
root@server:~ # pkg info | grep -i git
git-2.42.0                    Distributed source code management tool
gitolite-3.6.12,1             Access control layer on top of git
```

4 Gitolite Server Configuration

The `gitolite` installation did not create the `git` user account, so we'll do that.

4.1 Git User

```
root@git:~ # adduser
Username: git
```

```
Full name: git for gitolite
Uid (Leave empty for default):
Login group [git]:
Login group is git. Invite git into other groups? []:
Login class [default]:
Shell (sh csh tcsh git-shell nologin) [sh]:
Home directory [/home/git]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]: no
Lock out the account after creation? [no]:
Username   : git
Password   : <disabled>
Full Name  : git for gitolite
Uid        : 1002
Class      :
Groups     : git
Home       : /home/git
Home Mode  :
Shell      : /bin/sh
Locked     : no
OK? (yes/no): yes
adduser: INFO: Successfully added (git) to the user database.
```

Note: we used all the defaults, except for saying `no` to password-based authentication.

The details from the `passwd` file are:

```
root@git:~ # cat /etc/passwd | grep git

git_daemon:*:964:964:git daemon:/nonexistent:/usr/sbin/nologin
git:*:1002:1002:git for gitolite:/home/git:/bin/sh
```

The `git_daemon` user is not used by `gitolite`. However, the `git` user will be and its home directory will be the location of all repositories.

4.2 Gitolite Setup

According to the post-installation instructions we must run the `gitolite setup` command and specify a public key for the admin user. The public key we use is the one we copied to the remote server administrator's (`mv`) home directory:

```
mv@git:~ % whoami
mv
mv@git:~ % pwd
/home/mv
```

```
mv@git:~ % ls -la *.pub
-rw-r--r--  1 mv  mv  101 Jan  12  2024 gitolite_admin_user.pub
```

We become the `git` user (after becoming the `root` user via the `mv` user):

```
root@git:~ # su - git
git@git:~ $
```

```
git@git:~ $ pwd
/home/git
```

```
git@git:~ $ gitolite setup -pk /home/mv/gitolite_admin_user.pub

hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/git/repositories/gitolite-admin.git/
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/git/repositories/testing.git/
WARNING: /home/git/.ssh missing; creating a new one
    (this is normal on a brand new install)
WARNING: /home/git/.ssh/authorized_keys missing; creating a new one
    (this is normal on a brand new install)
```

The files and directories created look like this:

```
git@git:~ $ ls -l
total 8
-rw-------  1 git  git   12 Jan 12 21:07 projects.list
drwx------  4 git  git  512 Jan 12 21:07 repositories
git@git:~ $ ls -l repositories/
```

```
total 8
drwx------  8 git  git  512 Jan 12 21:07 gitolite-admin.git
drwx------  7 git  git  512 Jan 12 21:07 testing.git
git@git:~ $ ls -l .ssh/
total 4
-rw-------  1 git  git  270 Jan 12 21:07 authorized_keys
```

The `.ssh/` directory contains the `authorized_keys` file, which is managed by the gitolite software.

A gitolite `perl` script is run at every SSH login to the `git` server, when a configured key is used. This script (`/usr/local/libexec/gitolite/gitolite-shell`) is listed in the `authorized_keys` file.

```
git@git:~ $ cat .ssh/authorized_keys
# gitolite start
command="/usr/local/libexec/gitolite/gitolite-shell gitolite_admin_user",
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty
 ssh-ed25519 AAA...luP8 Gitolite admin user
# gitolite end
```

The script is presented with a username as the first parameter, which is the name associated with the public key file. Access is granted or denied to the repositories according to the this name and the contents of the `gitolite.conf` file, as we'll see in the next sections.

This is all the work that is required on the remote server for now. Most of the normal administration of the repositories happens on the remote server administrator's workstation, once the `gitolite-admin.git` repository is cloned onto it.

4.3 Gitolite Admin Repository

Gitolite uses an authoritative remote git repository for storing other repository specifications and their users. On the `git` host, this repository is here:

```
git@git:~ $ cd repositories/
git@git:~/repositories $ ls -lad gitolite-admin.git/
drwx------  8 git  git  512 Jan 12 21:07 gitolite-admin.git/
```

We return to our workstation and work using the key pair created for the `gitolite_admin_user`.

Cloning a working copy is done as follows. We create a directory to store the `gitolite-admin` repository, here it is called `vp`:

```
user@client:~ $ mkdir vp
user@client:~ $ cd vp
user@client:~/vp $
```

Now issue a clone command using our `gitolite` entry from our `.ssh/config` file:

```
$ git clone gitolite-vp:gitolite-admin.git
```

```
Cloning into 'gitolite-admin'...
Enter passphrase for key '/home/mv/.ssh/gitolite_admin_user':
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
```

```
[mv@think /usr/home/mv/vp]$ ls -la gitolite-admin/
total 11
drwxr-xr-x  5 mv  mv   5 Jan 12 20:56 .
drwxr-xr-x  3 mv  mv   3 Jan 12 20:55 ..
drwxr-xr-x  8 mv  mv  13 Jan 12 20:56 .git
drwxr-xr-x  2 mv  mv   3 Jan 12 20:56 conf
drwxr-xr-x  2 mv  mv   3 Jan 12 20:56 keydir
```

4.4 File `conf/gitolite.conf`

The initial contents of this file are:

```
$ cat gitolite-admin/conf/gitolite.conf
repo gitolite-admin
    RW+     =   gitolite_admin_user

repo testing
    RW+     =   @all
```

We edit this file to add `repo` sections, for creating new repositories, with access control symbols and usernames beneath them.

The usernames are specified by the public key used to interact with a repository.

4.5 Directory `keydir/`

The contents of the `keydir/` directory are initially as follows:

```
$ ls -la gitolite-admin/keydir/
total 6
drwxr-xr-x  2 mv  mv    3 Jan 12 20:56 .
drwxr-xr-x  5 mv  mv    5 Jan 12 20:56 ..
-rw-r--r--  1 mv  mv  101 Jan 12 20:56 gitolite_admin_user.pub
```

We add public keys to this directory as we add users to the repositories.

4.6 Add a User and Their Repository

To add a user, copy their public key to the `gitolite-admin/keydir/`. We'll use key number 2 that we created in section 2.3. That key was `git_remote_user_mv.pub`, and we'll copy it and rename it to just `mv.pub`. This means that the username used for access control will be `mv`.

```
$ cp ~/.ssh/git_remote_user_mv.pub gitolite-admin/keydir/mv.pub
```

Now we edit the `gitolite.conf` file and add a `repo` entry with the results as so:

```
repo gitolite-admin
    RW+     =   gitolite_admin_user

repo gitstuff
    RW+     =   mv
    R       =   @all

repo testing
    RW+     =   @all
```

We added the repository named `gitstuff` with two permission lines. One full control for user `mv` and one read-only for all others.

4.7 Commit the New User and Repository

The `gitolite` software reinforces using `git` by requiring you to use a git workflow to manage its configuration. This means we must use `git status`, `git add`, `git commit` and `git push` to finish the new configuration.

```
$ git status
fatal: not a git repository (or any parent up to mount point /usr)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
```

Whoops, we have to be in the `gitolite-admin` repository directory:

```
[mv@think /usr/home/mv/vp]$ cd gitolite-admin/
[mv@think /usr/home/mv/vp/gitolite-admin]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   conf/gitolite.conf

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        keydir/mv.pub

no changes added to commit (use "git add" and/or "git commit -a")
```

Let's add the new key and the modified configuration file to staging for commit:

```
mv@think /usr/home/mv/vp/gitolite-admin]$ git add keydir/mv.pub conf/gitolite.conf
```

Now commit the staged files to the local copy of the repository:

```
[mv@think /usr/home/mv/vp/gitolite-admin]$ git commit -m "Add user mv \
    and create the gitstuff repository."
[master 29cf12a] Add user mv and create the gitstuff repository.
 2 files changed, 5 insertions(+)
 create mode 100644 keydir/mv.pub
```

Now push the changes to the remote server, but first let's look at the results of the `git remote -v` command:

```
[mv@think /usr/home/mv/vp/gitolite-admin]$ git remote -v
origin  gitolite-vp:gitolite-admin.git (fetch)
origin  gitolite-vp:gitolite-admin.git (push)
```

The push:

```
[mv@think /usr/home/mv/vp/gitolite-admin]$ git push
Enter passphrase for key '/home/mv/.ssh/gitolite_admin_user':
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 631 bytes | 631.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote: hint: Using 'master' as the name for the initial branch. This default branch name
remote: hint: is subject to change. To configure the initial branch name to use in all
remote: hint: of your new repositories, which will suppress this warning, call:
remote: hint:
remote: hint:   git config --global init.defaultBranch <name>
remote: hint:
remote: hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
remote: hint: 'development'. The just-created branch can be renamed via this command:
remote: hint:
remote: hint:   git branch -m <name>
remote: Initialized empty Git repository in /home/git/repositories/gitstuff.git/
To gitolite-vp:gitolite-admin.git
   1ad91fc..29cf12a  master -> master
```

Things to note:

1. We are using the `gitolite_admin_user` to send changes to the `gitolite_admin` repository.

2. We see an announcement about a newly created empty repository in

   `/home/git/repositories/gitstuff.git/`.

If we look at the `repositories/` directory on the remote server, we'll see the new `gitstuff` repository:

```
git@git:~ $ ls -la repositories/
total 20
drwx------  5 git  git  512 Jan 12 21:41 .
drwxr-xr-x  5 git  git  512 Jan 12 21:41 ..
drwx------  8 git  git  512 Jan 12 21:41 gitolite-admin.git
drwx------  7 git  git  512 Jan 12 21:41 gitstuff.git
drwx------  7 git  git  512 Jan 12 21:41 testing.git
```

And if we look in the `.ssh/authorized_keys` file, we'll see an entry for user `mv` which matches the public key we added.

```
git@git:~ $ cat .ssh/authorized_keys
# gitolite start
command="/usr/local/libexec/gitolite/gitolite-shell gitolite_admin_user",no-port-forwarding,
no-X11-forwarding,no-agent-forwarding,no-pty ssh-ed25519 AAAA...luP8
Gitolite admin user
command="/usr/local/libexec/gitolite/gitolite-shell mv",no-port-forwarding,
no-X11-forwarding,no-agent-forwarding,no-pty ssh-ed25519 AAAAC3.../Ww
User for git remote repositories
# gitolite end
```

4.8 Creating and Linking the User Repository

Our final task for user `mv` is to create their `gitstuff` repository, add some files, and link it to our remote server. This is all done on the git user workstation.

```
[mv@think /usr/home/mv/vp]$ mkdir gitstuff
[mv@think /usr/home/mv/vp]$ cd gitstuff
[mv@think /usr/home/mv/vp/gitstuff]$ cp ~/.gitignore_global gitignore_global.txt
```

Initialize our local repository. Files can already exist in this folder. We can use it as a working directory, too.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
```

```
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /usr/home/mv/vp/gitstuff/.git/
```

See the status of our new repository. `git status` is a good command to memorize, as it is useful all the time.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        gitignore_global.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Add the file for staging and eventual commitment to the repository, and then check the status again:

```
[mv@think /usr/home/mv/vp/gitstuff]$ git add gitignore_global.txt
[mv@think /usr/home/mv/vp/gitstuff]$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   gitignore_global.txt
```

Now that we've added our initial file(s), we commit them to the repository. This is basically our forever starting point.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git commit -m "Add gitignore global file."
[master (root-commit) 11c88d0] Add gitignore global file.
 1 file changed, 456 insertions(+)
 create mode 100644 gitignore_global.txt
```

Now comes the whole point of using `gitolite`, linking the remote `gitstuff` repository we created with `gitolite` to this new, local repository.

Set it using the `git remote add` command, taking care to use the `git .ssh/config` entry for our 'normal' user, neither the `gitremote` entry, nor the `gitolite-vp` entry.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git remote add origin git:gitstuff.git
[mv@think /usr/home/mv/vp/gitstuff]$
```

See if we can use it now to `git push` our committed changes to the empty remote.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin master

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

We missed a step with our remote and branch linkage. Follow the instructions. We only have to use the `--set-upstream` option one time.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git push --set-upstream origin master
Enter passphrase for key '/home/mv/.ssh/git_remote_user_mv':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 2.46 KiB | 2.46 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To git:gitstuff.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Done!

We can now clone add other users' public keys via the `gitolite-admin` mechanism and they'll be able to clone and use the `gitstuff` repository.

I can also copy the private key associated with the `mv.pub` key to another system (along with the corresponding `.ssh/config` entry for host `git`), and clone and work with the repository, from that system without having to create a separate user.

4.9 Summary: Add Other Users and Repositories

Using the `gitolite-admin` repository:

1. User generates ssh key pair.
2. User sends public key to admin user.
3. Admin adds public key to `gitolite-admin keydir/` directory.
4. Admin adds repository to `conf/gitolite.conf` file and sets user's access levels based on their key name.

5. Admin uses `git add`, `commit` and then git `push` to make the changes.
6. User follows the steps in section 4.8

5 Git Configuration Files

There are three locations for configuration files that affect most users. They are:

1. System - these are operating system wide settings. I've never needed to adjust these. They can be viewed with the following command:

   ```
   [mv@think /usr/home/mv/1mvgdocs/bsd]$ git config --system -l
   fatal: unable to read config file '/usr/local/etc/gitconfig':
    No such file or directory
   ```

   As you can see, there are no `system` level settings.

2. Global - these are user level settings that are stored in the user's home directory. There are at least two setting options that you'll set within this configuration level. Your name and email.

   ```
   [mv@think /usr/home/mv/1mvgdocs/bsd]$ git config --global -l
   user.name=Call me Vinny
   user.email=git@vinnythegeek.ca
   core.autocrlf=input
   core.excludesfile=/home/mv/.gitignore_global
   ```

   Other `global` settings that could be useful are command aliases.

3. Local - these are repository level settings that are stored within a `.git` folder in the `config` file (i.e. `.git/config`). The usual settings at this level are "remotes".

   ```
   [mv@think /usr/home/mv/1mvgdocs/bsd]$ git config --local -l
   core.repositoryformatversion=0
   core.filemode=true
   core.bare=false
   core.logallrefupdates=true
   core.sharedrepository=0640
   receive.denynonfastforwards=true
   remote.origin.url=sccs:bsd.git
   remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
   branch.master.remote=origin
   branch.master.merge=refs/heads/master
   pull.rebase=false
   ```

If you need to see where a configuration setting is coming from, you can use the `git config --list --show-origin` command:

```
[mv@think /usr/home/mv/1mvgdocs/bsd]$ git config --list --show-origin
file:/home/mv/.gitconfig        user.name=Call me Vinny
file:/home/mv/.gitconfig        user.email=git@vinnythegeek.ca
file:/home/mv/.gitconfig        core.autocrlf=input
file:/home/mv/.gitconfig        core.excludesfile=/home/mv/.gitignore_global
file:.git/config        core.repositoryformatversion=0
file:.git/config        core.filemode=true
file:.git/config        core.bare=false
file:.git/config        core.logallrefupdates=true
file:.git/config        core.sharedrepository=0640
file:.git/config        receive.denynonfastforwards=true
file:.git/config        remote.origin.url=sccs:bsd.git
file:.git/config        remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
file:.git/config        branch.master.remote=origin
file:.git/config        branch.master.merge=refs/heads/master
file:.git/config        pull.rebase=false
```

6 Git Working System Configuration

The workstation used to develop software and documentation needs to be setup. The first settings are your global git user name and email setting.

```
user@client:~ % git config --global user.name "Mark G."
user@client:~ % git config --global user.email git@vinnythegeek.ca
```

6.1 File Line Endings

The next config. item helps deal with line endings with Microsoft development systems. On an Unix-like system, setting the `core.autocrlf` to `input` is sensible.

```
user@client:~ % git config --global core.autocrlf input
```

6.2 Ignoring Files

There are a lot of files that you never want to store in a repository. Files that are automatically generated, files that contain sensitive information, useless operating system created files, and so on.

Change to your home directory, then create a `.gitignore_global` file. We'll fill it with many entries from the internet. We also need to set the `core.excludesfile` setting.

```
user@client:~ % touch .gitignore_global
user@client:~ % git config --global core.excludesfile ~/.gitignore_global
```

Good templates for ignore files can be found at Github. The specific link for LaTeX type files was at:

```
https://raw.githubusercontent.com/github/gitignore/master/TeX.gitignore
```

The contents of that file were added to the `.gitignore_global` file. Plus many more from:

```
https://raw.githubusercontent.com/github/gitignore/master/Global/macOS.gitignore
```

along with Python and Django related items:

```
https://raw.githubusercontent.com/github/gitignore/master/Python.gitignore
```

We also add:

```
envdir/

# Eclipse / IDE items
.project
.pydevproject
```

Since the sensitive django settings will be stored in environment variables defined in the above listed directory.

A copy of my `.gitignore_global` file can be found at TXT: .gitignore_global file Since it is named `gitignore_global.txt`, use your browser's "Save Page As" command to rename it to `.gitignore_global` and save it in your home directory.

6.3 `git status`

Find out what is happening in your repository with the `git status` command. It'll tell you about untracked files, modified files and files that are staged and ready for committing.

6.4 `git log`

Use this command to look at commit history.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git log
ESC[33mcommit 11c88d013a3bacc34fe9a7ffe948f51283d606acESC[mESC[33m
 (ESC[mESC[1;36mHEADESC[mESC[33m -> ESC[mESC[1;32mmasterESC[mESC[33m,
  ESC[mESC[1;31morigin/masterESC[m
ESC[33m)ESC[m
Author: Call me Vinny <git@vinnythegeek.ca>
Date:   Fri Jan 12 22:40:14 2024 -0800

    Add gitignore global file.
```

Looking closely at the output, we see a bunch of `ESC` sequences. This usually means colour is trying to be output. If the terminal doesn't support colour, or you prefer not to use it, set the `color.ui` config variable to `false`:

```
[mv@think /usr/home/mv/vp/gitstuff]$ git config --global color.ui false
```

The `ESC` sequences are now gone.

```
[mv@think /usr/home/mv/vp/gitstuff]$ git log
commit 11c88d013a3bacc34fe9a7ffe948f51283d606ac (HEAD -> master, origin/master)
Author: Call me Vinny <git@vinnythegeek.ca>
Date:   Fri Jan 12 22:40:14 2024 -0800

    Add gitignore global file.
```

6.5 Changing Remote

I had a repository that I used for a software engineering course. Its remote was:

```
mv@think:~/uvic/seng265/mvg$ git remote -v
origin ssh://mvg@git.seng.uvic.ca/seng/git/seng265/mvg (fetch)
origin ssh://mvg@git.seng.uvic.ca/seng/git/seng265/mvg (push)
```

A remote can be updated as follows:

```
mv@think:~/uvic/seng265/mvg$ git remote set-url origin git:seng265.git
mv@think:~/uvic/seng265/mvg$ git remote -v
origin git:seng265.git (fetch)
origin git:seng265.git (push)
```

We can set the name and email of the user at the local config level of the repository to override the global settings:

```
user@client:~/vp/gitstuff % git config --local user.name "Mark G."
user@client:~/vp/gitstuff % git config --local user.email mvg@example.com
```

7 Appendix: DNS Name for Service

We will create `git.vinnythegeek.ca` as the hostname for internal use only (at his time).

First, freeze the dynamic zone for `vinnythegeek.ca`:

```
root@dnsprimary:/usr/local/etc/namedb/dynamic # rndc freeze vinnythegeek.ca in xfer
```

Then edit the zone file, remembering to update the serial number.

```
root@dnsprimary:/usr/local/etc/namedb/dynamic # vi vinnythegeek.ca.ext.db

Serial number updated...

...
git      IN      A        10.24.7.22
         IN      AAAA     fdea:557c:9747:1060::2443
...
```

Thaw the zone on the primary DNS server, which will cause a reload of the zone.

```
root@dnsprimary:/usr/local/etc/namedb/dynamic # rndc thaw vinnythegeek.ca in xfer
```

Through the magic of zone transfers, the changes are propagated to the secondary / authoritative servers.