## **Start a new git repository:**

**GIT Repository from Bare Repository:**

1. Create a bare Repository using

**git init –bare project.git**

2. It creates “project.gti” folder which is the repository with bare contents to establish default requests “pull” and “push” operations

3. You can link **bare repository** with the contents from the remote user **repository** by linking with the command

**git remote add origin <repository-link>**

**ex:** git remote add origin [git@192.168.4.184](mailto:git@192.168.4.184):/home/git/project.git

### A new repo from scratch

Say you’ve just got some data from a collaborator and are about to start exploring it.

* Create a directory to contain the project.
* Go into the new directory.
* Type git init.
* Write some code.
* Type git add to add the files (see the [typical use page](http://kbroman.org/github_tutorial/pages/routine.html)).
* Type git commit -m “message”.

The first file to create (and add and commit) is probably a ReadMe file, either as plain text or with [Markdown](http://daringfireball.net/projects/markdown/), describing the project.

Markdown allows you to add a bit of text markup, like [hyperlinks](http://en.wikipedia.org/wiki/Hyperlink), bold/italics, or to indicate code with a monospace font. Markdown is easily converted to html for viewing in a web browser, and GitHub will do this for you automatically.

### A new repo from an existing project

Say you’ve got an existing project that you want to start tracking with git.

* Go into the directory containing the project.
* Type git init.
* Type git add to add all of the relevant files.
* You’ll probably want to create a .gitignore file right away, to indicate all of the files you don’t want to track. Use git add .gitignore, too.
* Type git commit.

### Connect it to github

You’ve now got a local git repository. You can use git locally, like that, if you want. But if you want the thing to have a home on github, do the following.

* Go to [github](http://github.com/).
* Log in to your account.
* Click the [new repository](https://github.com/new) button in the top-right. You’ll have an option there to initialize the repository with a README file, but I don’t.
* Click the “Create repository” button.

Now, follow the second set of instructions, “Push an existing repository…”

$ git remote add origin git@github.com:username/new\_repo

$ git push -u origin master

Actually, the first line of the instructions will say

$ git remote add origin https://github.com/username/new\_repo

But I use git@github.com:username/new\_repo rather than https://github.com/username/new\_repo, as the former is for use with [ssh](http://en.wikipedia.org/wiki/Secure_Shell) (if you set up ssh as I mentioned in “[Your first time](http://kbroman.org/github_tutorial/pages/first_time.html)”, then you won’t have to type your password every time you push things to github). If you use the latter construction, you’ll have to type your github password every time you push to github.

**PUSH Operation**

**git push origin master**

**PULL operation**

**git pull**

### Link:<https://www.linux.com/LEARN/HOW-RUN-YOUR-OWN-GIT-SERVER>

### [ssh with no password, with ssh-keygen key](https://www.garron.me/en/bits/ssh-key-keygen-login-no-password.html)

## Link: <https://www.garron.me/en/bits/ssh-key-keygen-login-no-password.html>

## SSH keys

Log into ssh servers (Mac OS X or Linux) without passwords, using ssh key. How to generate them with ssh-keygen and install on the server — Arch Linux Wiki

## Generate ssh keys

To generate your private and public keys for ssh login with no password we will use ssh-keygen in the client computer (The one that will log into the server using ssh)

ssh-keygen -t rsa

It will ask you for a passphrase, is up to you to use one or not. Here are the pros and cons.

Pros

* Extra security
* If you lose your private key, no one will be able to use it.

Cons

* It can be tedious to be typing the pass phrase each time you want to log into the server
* You can not use the key for scripts, as they will not be able to type the pass phrase.

I prefer not to use a passphrase, anyway my home folder in my laptop is encrypted.

## Install the ssh key

If you are on Arch Linux, Ubuntu or Debian use:

ssh-copy-id username@remote-server.org

If you do not have the command ssh-copy-id, for example if you are on a Mac.

cat ~/.ssh/id\_rsa.pub | ssh username@machine "cat >> ~/.ssh/authorized\_keys"

Run ssh-add on the client machine, that will add the SSH key to the agent. Confirm with ssh-add -l (again on the client) that it was indeed added.

Use root as username to install your key for the root user, you can run for as many users in the server as you want or need.

Now test that the work by loggin into the server, you should not be promped for a password.

## Secure your server

Now that you know it works, it is time to secure your server. Edit the file /etc/ssh/sshd\_config and look for: PermitRootLogin and change it from yes or no to: without-password. You will now will only be able to log into the server using root user, by using the keys.

### [Backup git bare repository](https://www.garron.me/en/bits/backup-git-bare-repo.html)

###### Link: <https://www.garron.me/en/bits/backup-git-bare-repo.html>

## Introduction

Git is great to version control your work, and it is also good that is does not have a central repository, but a distributed authority.

Anyway, when you create bare repository to sync the job between different contributors, you are somehow in a "virtual" way creating a centralized authority for git. Usually contributors will clone this bare repository work with it, make changes, commit them and then upload (push) them to the bare repository.

Before going further, let's see what is a bare repository.

A git repository is usually divided in two parts, one is the history of your project, and it is made by a lot of files in a hidden folder named .git. The other part is your working folder, which is the current state of your job, as was created by git, with the info in the .git folder. In a bare repository, there is no working folder, just the history of your repository. In other words, all the info needed to create a working folder in its current state or any state it may have had in the past.

Bare repositories are very important when a lot of people is working on the same project. All of them can clone the bare repository with:

git clone [bare repository url or folder]

Work on it, commit changes and push the changes back to the bare repository. You can read more about this in this [introduction to git](https://www.garron.me/en/articles/git-101-basics-introduction-basic-use-commands.html) article.

Now that we know how important bare repositories are, you may want to know how to back it up? Being git almost a backup system by it own, you may assume that all clones are backups, and that is true. But there is another way to be sure you have a complete backup of your bare repository. git clone --mirror.

The –mirror switch will do this:

–mirror

Set up a mirror of the source repository. This implies –bare. Compared to –bare, –mirror not only maps local branches of the source to local branches of the target, it maps all refs (including remote branches, notes etc.) and sets up a refspec configuration such that all these refs are overwritten by a git remote update in the target repository.

## Backup a bare repository with git clone –mirror

Assumptions

* You have access via ssh to your bare repository
* You are doing the backups in another server
* You will be doing periodic backups automatically

To use this command and backup your bare repository in a timely basis, do this.

We will do this in a different server, so the backup is a real backup. Let's create a place to store it.

mkdir ~/my-git-backup

cd ~/my-git-backup

Then, clone the bare respository.

git clone --mirror user@server:/url-to-repo.git

Enter the ssh password, and you have your backup, it is time to enable automatic and periodic backups.

Prepare your server, where the backup is stored to access the one where the original bare repository is via [ssh with no password](https://www.garron.me/en/bits/ssh-key-keygen-login-no-password.html)

## Removing a remote

Use the git remote rm command to remove a remote URL from your repository.

The git remote rm command takes one argument:

* A remote name, for example, destination

### Example

These examples assume you're [cloning using HTTPS](https://help.github.com/articles/which-remote-url-should-i-use/" \l "cloning-with-https-urls-recommended), which is recommended.

git remote -v

# View current remotes

origin https://github.com/OWNER/REPOSITORY.git (fetch)

origin https://github.com/OWNER/REPOSITORY.git (push)

destination https://github.com/FORKER/REPOSITORY.git (fetch)

destination https://github.com/FORKER/REPOSITORY.git (push)

git remote rm destination

# Remove remote

git remote -v

# Verify it's gone

origin https://github.com/OWNER/REPOSITORY.git (fetch)

origin https://github.com/OWNER/REPOSITORY.git (push)

Note: git remote rm does not delete the remote repository from the server. It simply removes the remote and its references from your local repository.