Git

Fundamentals of Git

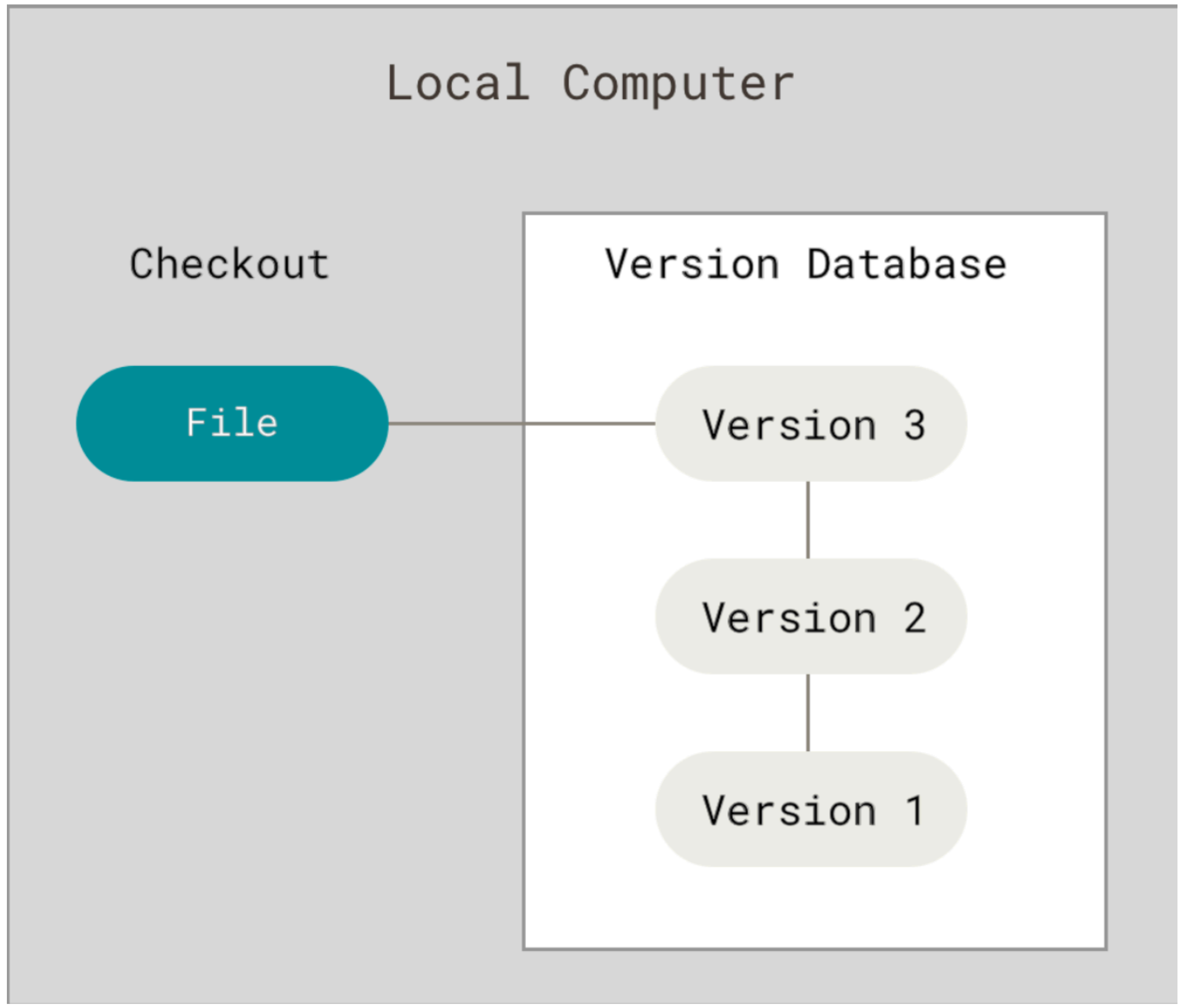
What is Version Control?

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

Different types of version control.

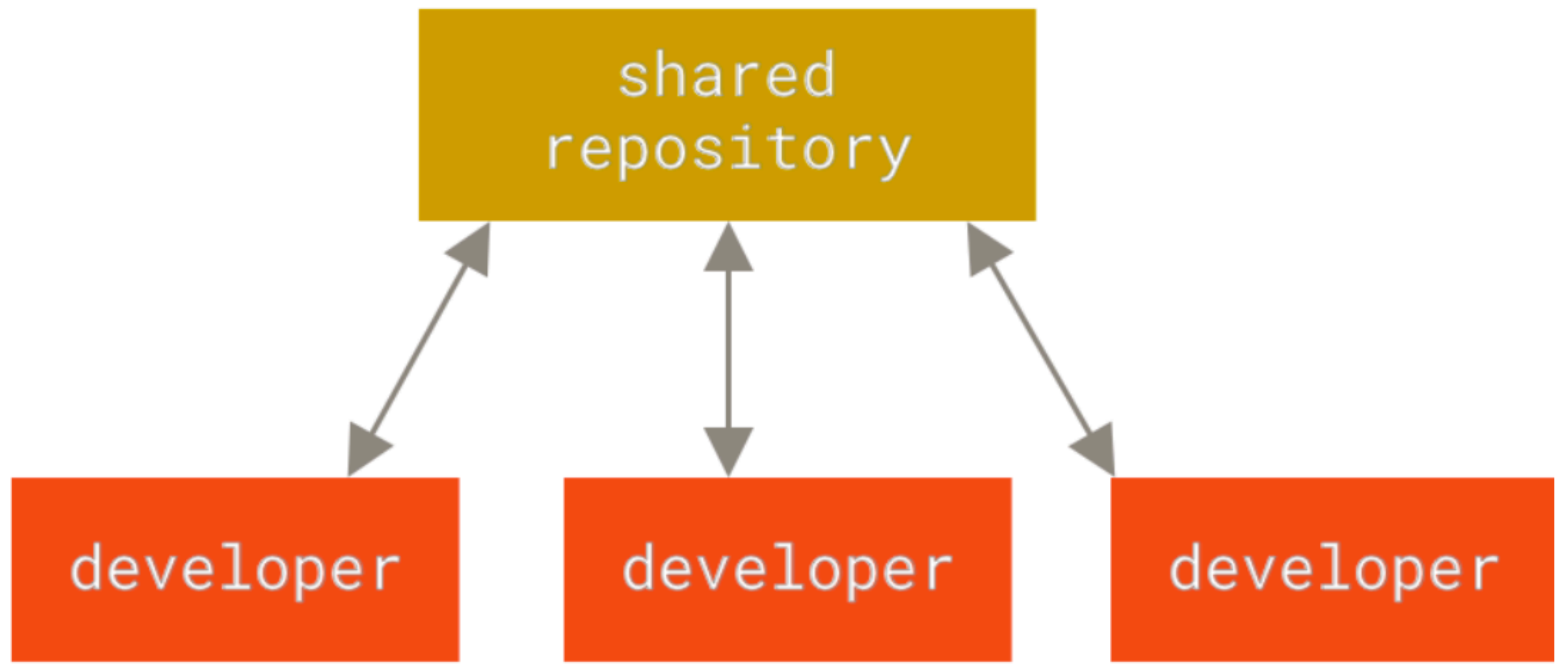
Local Version Control Systems.

- Many people's version control method of choice is to copy files Into another directory (perhaps a time-stamped directory, if they're clever).
- This approach is very common because it is so simple, but it is also incredibly error prone.
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.
- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



Centralised Version Control Systems.

- The next major issue that people encounter is that they need to collaborate with developers on other systems.
- To deal with this problem, Centralised Version Control Systems (CVCSs) were developed.
- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- For many years, this has been the standard for version control.
- This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing.
- Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client.



Disadvantages of Centralised Version Control Systems.

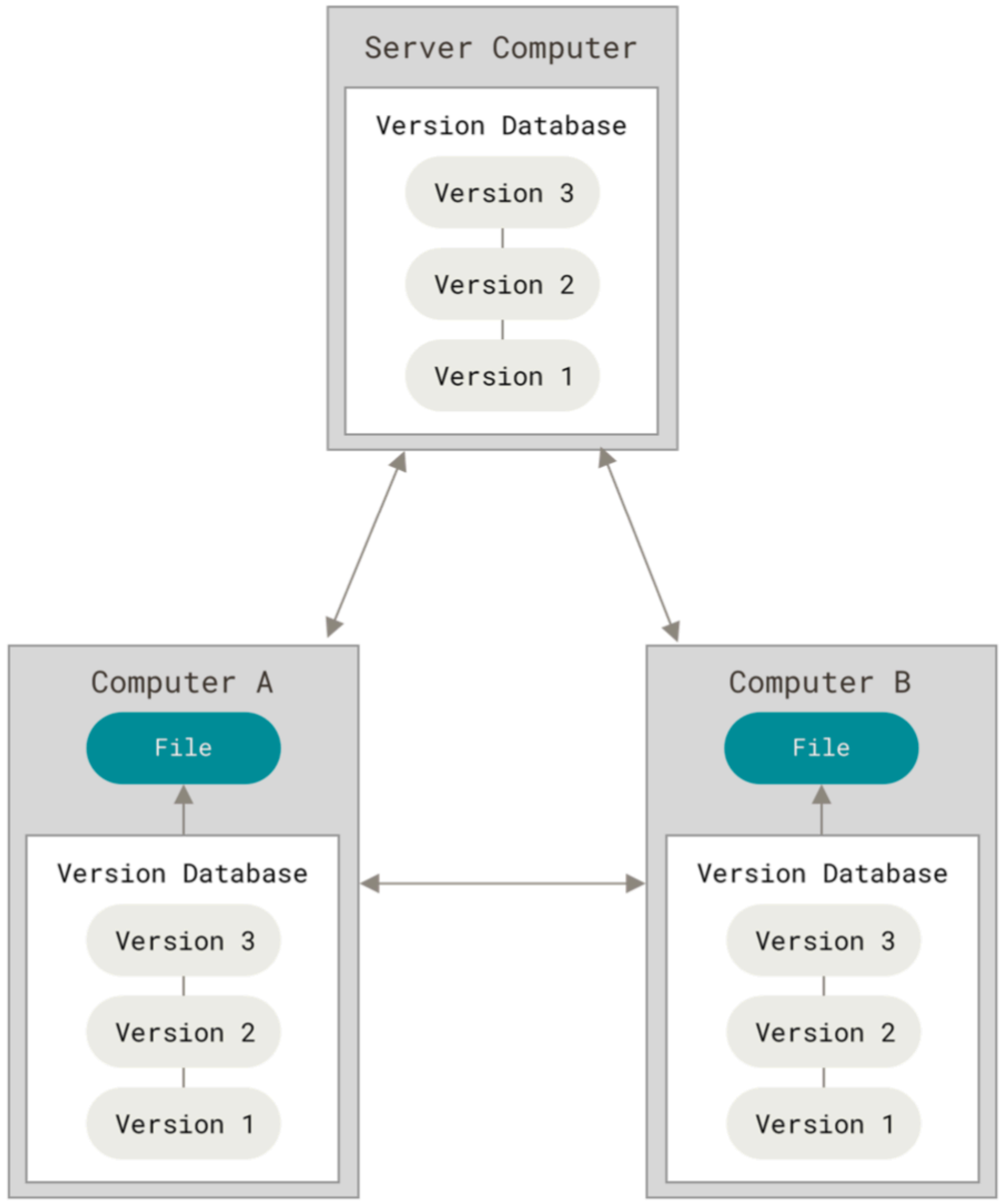
- This setup also has some serious downsides. The most obvious is the single point of failure that the centralised server represents.
- If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.
- If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines.
- Local VCS systems suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

- In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.

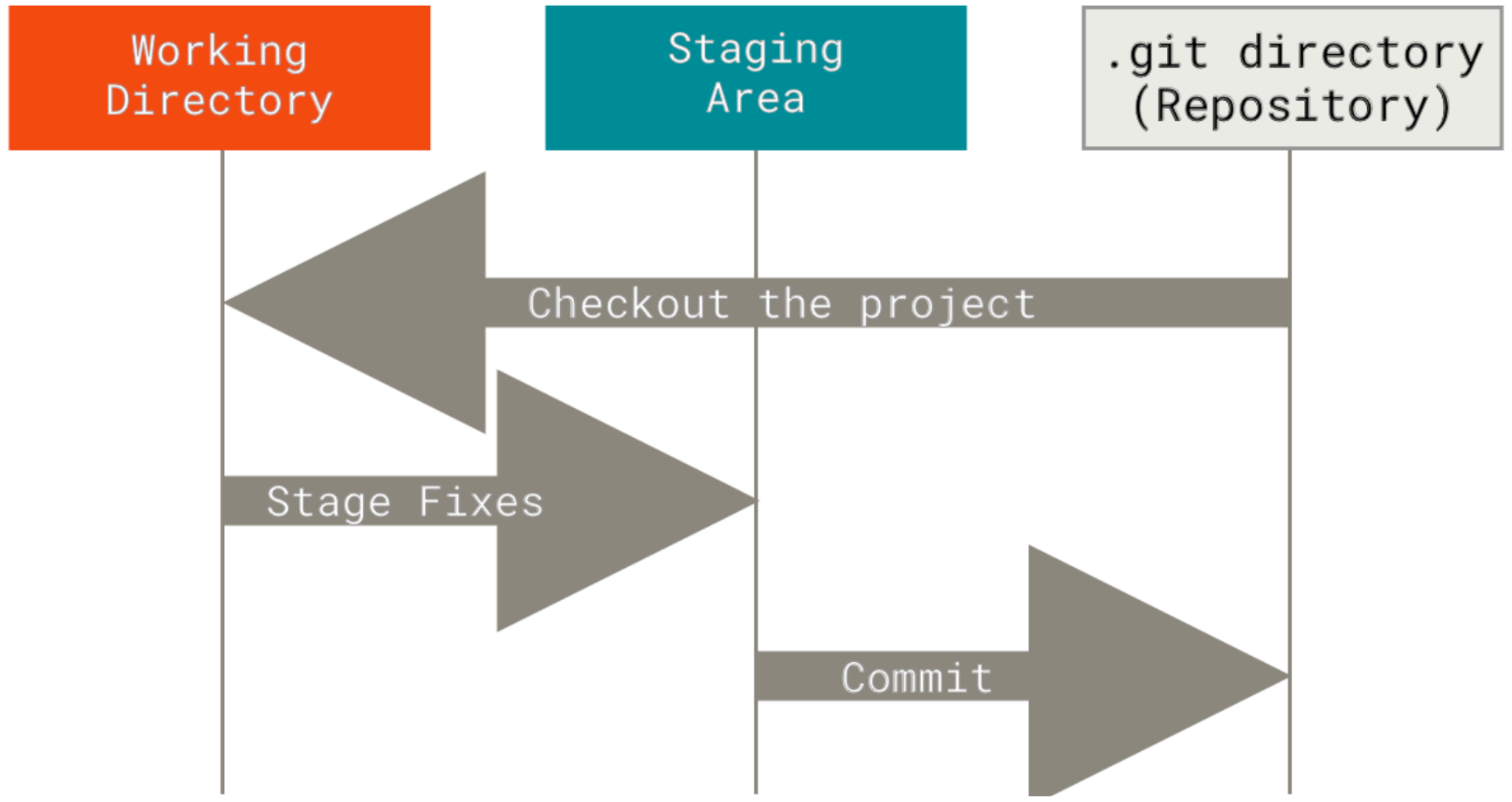
- Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.

- Every clone is really a full backup of all the data.



3 States of Git

- Git has three main states that your files can reside in modified, staged, and committed.
- **Modified** means that you have changed the file but have not committed it to your database yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** means that the data is safely stored in your local database.
- This leads us to the three main sections of a Git project: the working tree, the staging area, and the Git directory.



- The **working tree** is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- The **staging area** is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git parlance is the "index", but the phrase "staging area" works just as well.
- The **Git directory** is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.
- The basic Git workflow goes something like this:
    - You modify files in your working tree.
    - You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
    - You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.
- If a particular version of a file is in the Git directory, it's considered committed.
- And if it was changed since it was checked out but has not been staged, it is modified.

Configuring Git.

Configuring Git

- Download git from **https://git-scm.com/downloads**

- To get the version of the git installed, open gitbash and type the below command.

```
git --version
```

- Configuring username and password.

```
git config --global user.name ‘test user’
git config --global user.email ‘test.user@demo.com’
```

- To view your git configuration.

```
git config –list
```

cont…

- To get help, (Lists all the available commands)

```
git help
```

Creating git repository.

- You typically obtain a Git repository in one of two ways:
  1. You can take a local directory that is currently not under version control, and turn it into a Git repository.

  2. You can **clone** an existing Git repository from elsewhere.

- In either case, you end up with a Git repository on your local machine, ready for work.

Cont…

- Initialising a Repository in an Existing Directory.
- If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory. If you've never done this, it looks a little different depending on which system you're running:

```
for Linux:
$ cd /home/user/my_project

for macOS:
$ cd /Users/user/my_project

for Windows:
$ cd C:/Users/user/my_project

and type:
$ git init
```

- This creates a new subdirectory named **.git** that contains all of your necessary repository files.

Cont…

- At this point, nothing in your project is tracked yet.
- If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit.
- You can accomplish that with a few git add commands that specify the files you want to track, followed by a git commit:

```
$ git add test.py
$ git add LICENSE.txt
$ git commit -m 'Initial project version’
```

Cont…

- Cloning an Existing Repository
- If you want to get a copy of an existing Git repository — for example, a project you'd like to contribute to — the command you need is **git clone**.
- Git receives a full copy of nearly all data that the server has.
- Every version of every file for the history of the project is pulled down by default when you run git clone.
- You clone a repository with git clone <url>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

```
$ git clone https://github.com/libgit2/libgit2
```

- This creates a directory named libgit2, initialises a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version.
- If you go into the new libgit2 directory that was just created, you'll see the project files in there, ready to be worked on or used.

Git Commands

Git Commands

- git-log - Show commit logs

```
$ git log
commit c36d2103222cfd9ad62f755fee16b3f256f1cb21
Author: Test User <test.user@demo.com>
Date:   Tue Aug 26 22:09:26 2019 -0300
  Initial project version
```

- git-status - Show the working tree status

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Cont..

- git-add - Add file contents to the staging area

  ```
  $ git add .
  Adds all the files that are changed to the staging area.

  $ git add test.py
  Adds only the file test.py to the staging area.
  ```

- git-reset - removes the changes from staging area.

  ```
  $ git reset HEAD test.py (Removes the changes to the file from staging area)
  $ git checkout test.py (gets the remote repository version of the test.py file)
  ```

- Switching Branches.

  ```
  $ git checkout master (switches to master branch)
  $ git checkout development (switches to development branch)
  $ git checkout testing (switches to testing branch)
  ```

- git-diff - Show changes between commits, commit and working tree, etc

  ```
  $ git diff
  ```

Cont..

- git mv - renames the existing file

```
$ git mv test.py demo.py
(Renames test.py to demo.py and the changes are moved to staging area automatically by
git. To complete the task we need to commit to the git).
$ git commit -m “renamed file”
```

- git rm - deletes the existing file

```
$ git rm demo.py
(Deletes demo.py and changes are moved to staging area automatically by git. To
complete the task we need to commit to the git).
$ git commit -m “deleted file”
```

GitHub

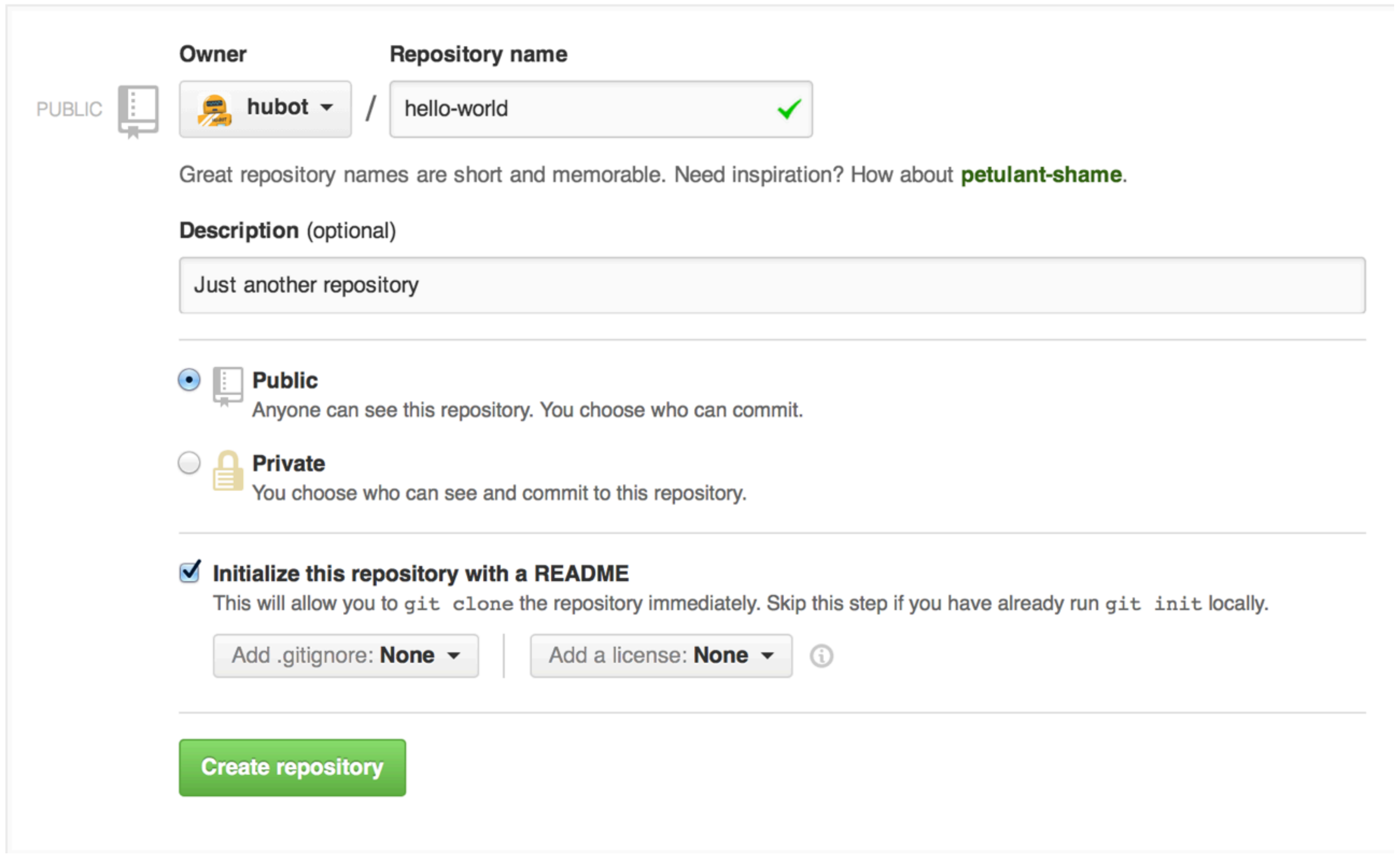
What is GitHub?

- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

- **Create a Repository.**

- A repository is usually used to organise a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.
- **To create a new repository**
    - In the upper right corner, next to your avatar or identicon, click and then select New repository.
    - Name your repository hello-world.
    - Write a short description.
    - Select Initialise this repository with a README.

Cont...


Owner

PUBLIC hubot / Repository name

hello-world

Great repository names are short and memorable. Need inspiration? How about **petulant-shame**.

Description (optional)

Just another repository

[✓ **Public**]
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

[✓] **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None**

Add a license: **None**

- Click **Create repository**.

Create a Github Branch.

- Branching is the way to work on different versions of a repository at one time.

- By default your repository has one branch named main which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to main.

- When you create a branch off the main branch, you're making a copy, or snapshot, of main as it was at that point in time. If someone else made changes to the main branch while you were working on your branch, you could pull in those updates.

To create a new branch

- Go to your new repository hello-world.
- Type a branch name, development, into the new branch text box.
- Select the blue Create branch box or hit "Enter" on your keyboard.

- **Configuring remote repository in local machine.** (setting up nick-name to remote repository)

```
$ git remote add remote https://github.com/testuser/hello-world.git
$ git remote -v (Shows the existing remote repositories)
```

- **Pushing the local changes to remote repository.**

```
$ git push remote master (master branch to remote)
$ git push remote development (development branch to remote)
```

- **Pulling the remote changes to local repository.** (remote branch to master)

```
$ git pull remote master
$ git pull remote development
```

- **Ignoring files Intentionally using file `.gitignore`**
    - A gitignore file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected.
    - Each line in a gitignore file specifies a pattern. When deciding whether to ignore a path, Git normally checks gitignore patterns from multiple sources, with the following order of precedence, from highest to lowest (within one level of precedence, the last matching pattern decides the outcome):

Sample gitignore file

```
$ cat .gitignore
   # Exclude the below patterns in the git repository
   *.html
   *.idea
   *.pyc
   *.config
   /foo/*.html
```

SSH Authentication

SSH Keys (Secure Shell)

- An SSH key is an access credential in the SSH protocol.
- Its function is similar to that of user names and passwords, but the keys are primarily used for automated processes and for implementing single sign-on by system administrators and power users.

Creating SSH Keys

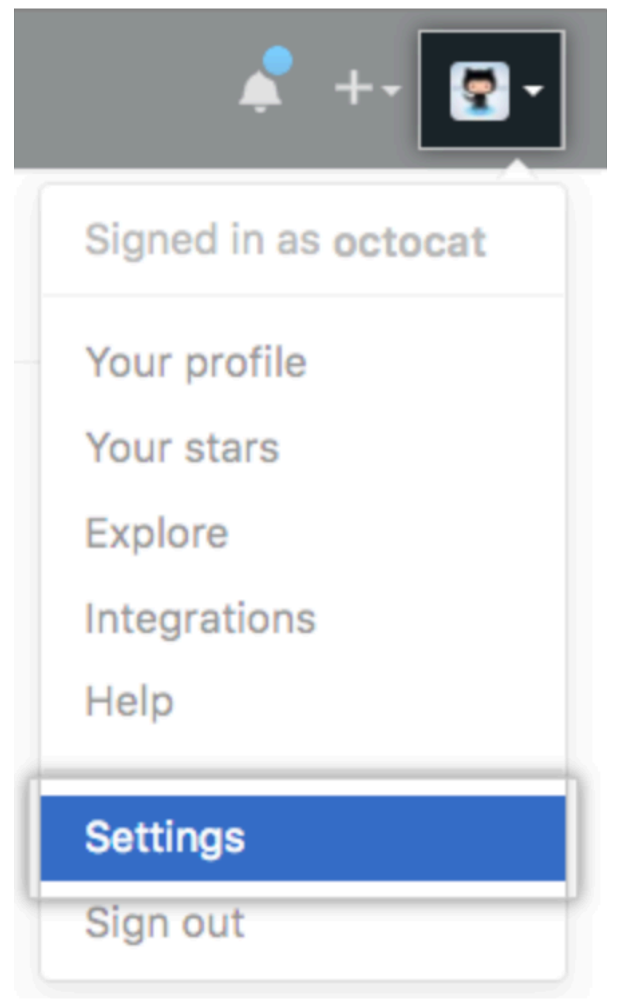
- Navigate to git repository in the file system and create a folder named .ssh
- Navigate to .ssh folder and generate the keys using below command.

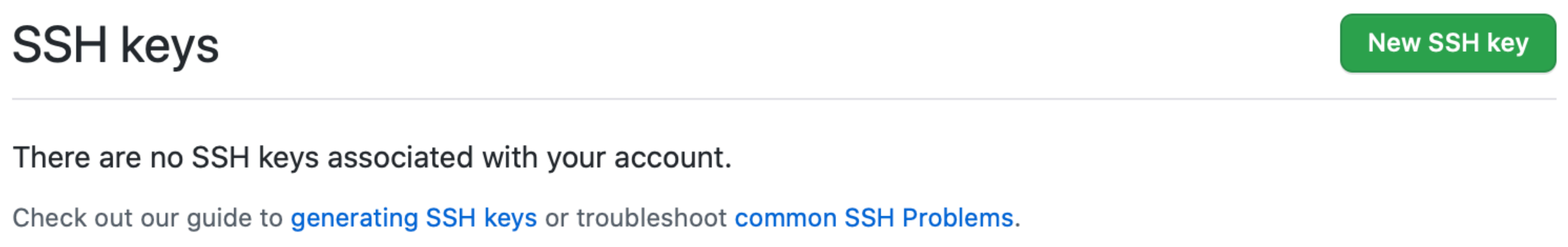
```
ssh-keygen -t rsa -C “test.user@demo.com”
```

- Once you execute the above command, you will be prompted to save the file.
- Hit enter without changing any path.
- You will be prompted with a pass phrase. You can either to choose one or you can ignore it by hitting enter key.
- Once you hit enter, your public and private keys will be generated inside two different files. (id_rsa and id_rsa_pub)

Adding a new SSH key to your GitHub account

- Copy the SSH key to your clipboard.
- In the upper-right corner of any page, click your profile photo, then click Settings.
- In the user settings sidebar, click SSH and GPG keys.
- Click New SSH key or Add SSH key.

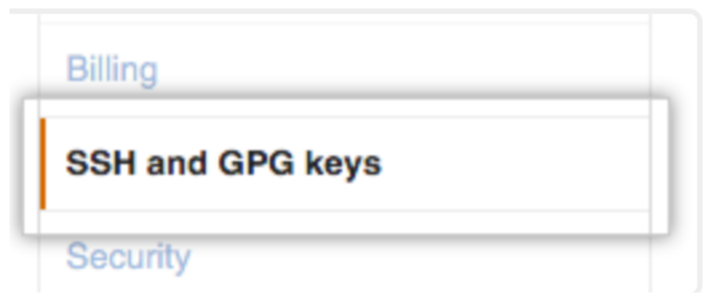



SSH keys

New SSH key

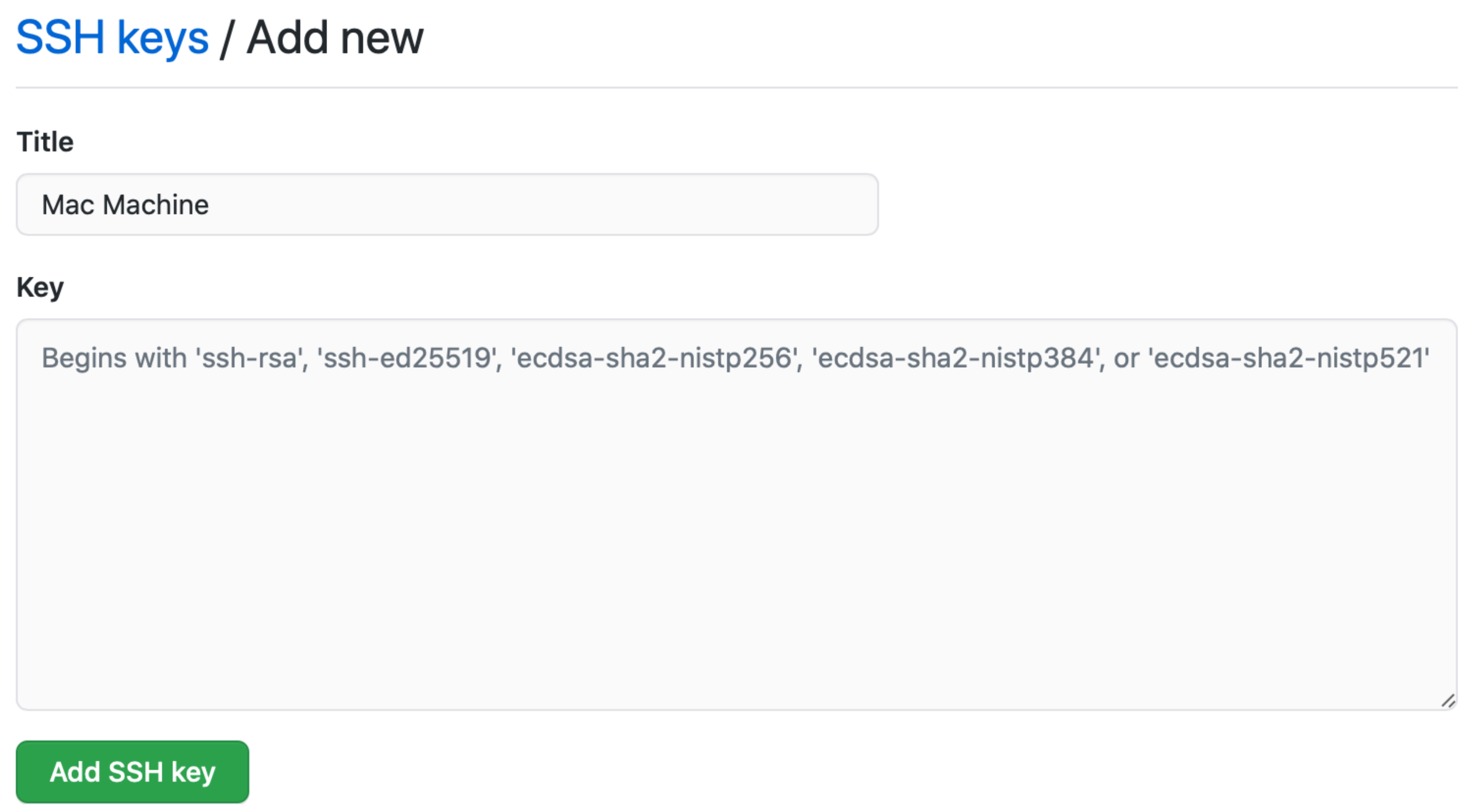
There are no SSH keys associated with your account.

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.



Cont..

- In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Pro”.



SSH keys / Add new

Title

Mac Machine

Key

Begins with 'ssh-rsa', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

- Paste your key into the "Key" field and click on “Add SSH Key”
- If prompted, confirm your GitHub Enterprise password.

END