Install git

OSX:

```
1. xcode-select --install
2. /usr/bin/ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
3. brew install git
```

Linux:

1. For Redhat/CentOS: `yum install git`
2. For Debian/Ubuntu: `apt-get install git`

Windows:

1. https://github.com/git-for-windows/git/releases/download/v2.21.0.windows.1/Git-2.21.0-64-bit.exe
2. Use the "Git Bash" program instead of "Command Prompt"
3. Setup editor: `git config --global core.editor '"C:/Program Files (x86)/Notepad++/notepad++.exe"'`

Verify Installation

```
git --version
```

Output:

```
git version 2.20.1 (Apple Git-117)
```

Run the following:

```
git config --global user.name "user1"
git config --global user.email "user1@meetup.com"
```

To verify, run:

```
git config --list
```

Output:

```
user.name=user1
user.email= user1@meetup.com
```

Install gogs

(local git server, I am assuming not everyone will have git enterprise)

Install docker (I assume this is already done as there was a docker meetup before)
Save the contents below to this file: `~/Downloads/docker-compose-gogs.yaml`:

```
version: "2"

services:
  gogs:
    restart: unless-stopped
    image: gogs/gogs
    volumes:
      - ~/Downloads/gogs:/data
    ports:
      - "22:22"
      - "3000:3000"
```

To verify installation, run:

```
docker-compose -f ~/Downloads/docker-compose-gogs.yaml up
```

To Stop gogs, run:

```
docker-compose -f ~/Downloads/docker-compose-gogs.yaml down
```

Next steps will be done during meetup.

Create SSH keypair to interact with git

Your existing keys will be overwritten. Before proceeding, check for existing keys as follows:

```
ls -al ~/.ssh/
```

if you see these two files, you already have a key pair:

```
-rw-------   1 user  group   1679 Aug 18  2016 id_rsa
-rw-r--r--   1 user  group    405 Aug 18  2016 id_rsa.pub
```

Make sure "id_rsa" has 600 permission.

If not, follow instraction from here:
https://help.github.com/en/enterprise/2.16/user/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

```
ssh-keygen -t rsa -b 4096 -C "user1@meetup.com"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

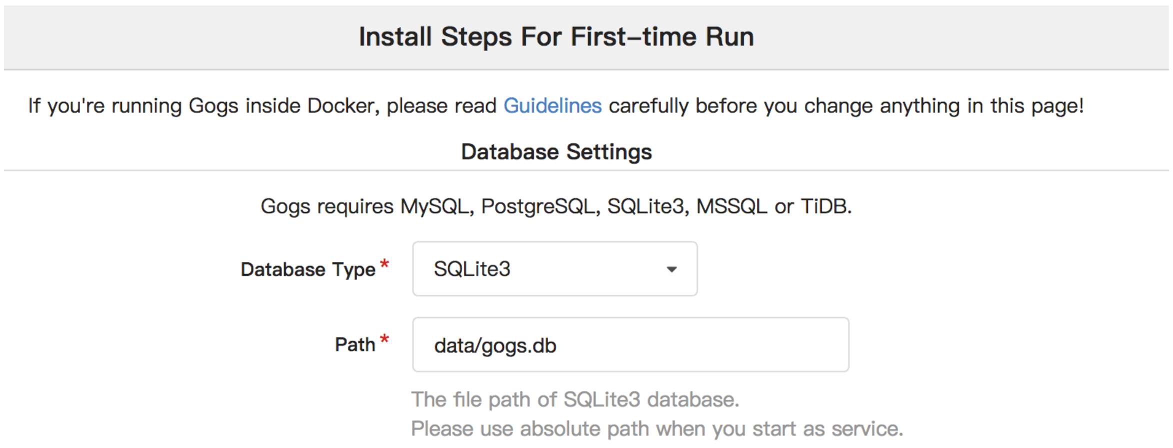
```
ssh git@localhost
```

```
Hi there, You've successfully authenticated, but Gogs does not provide shell access.
If this is unexpected, please log in with password and setup Gogs under another user.
Connection to localhost closed.
```

Create an Organization & Repo

http://localhost:3000

Org Name: `my_org`
Repo Name: `my_repo`



Install Steps For First-time Run

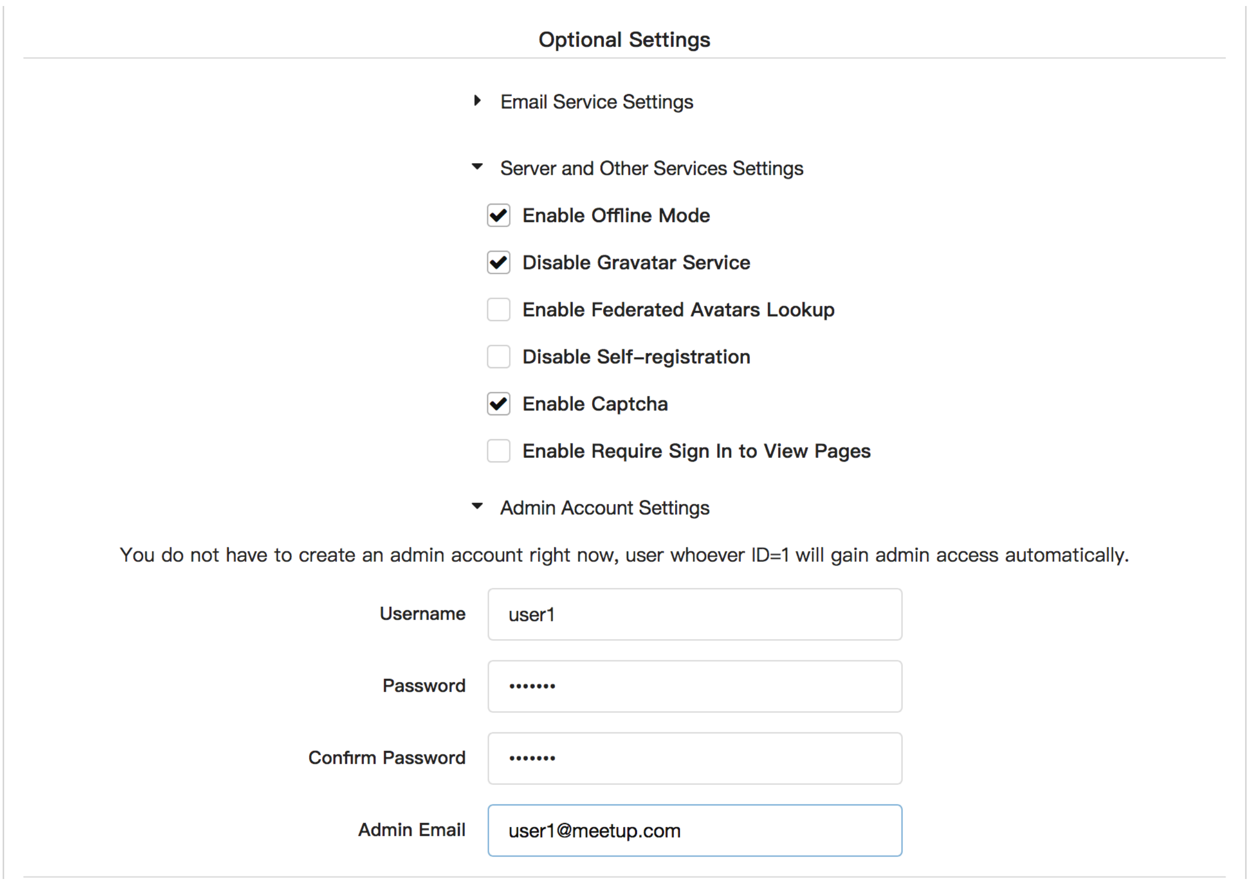
If you're running Gogs inside Docker, please read Guidelines carefully before you change anything in this page!

Database Settings

Gogs requires MySQL, PostgreSQL, SQLite3, MSSQL or TiDB.

Database Type *	SQLite3
Path *	data/gogs.db

The file path of SQLite3 database.
Please use absolute path when you start as service.



Optional Settings

▸ Email Service Settings

▾ Server and Other Services Settings

☑ Enable Offline Mode

☑ Disable Gravatar Service

☐ Enable Federated Avatars Lookup

☐ Disable Self-registration

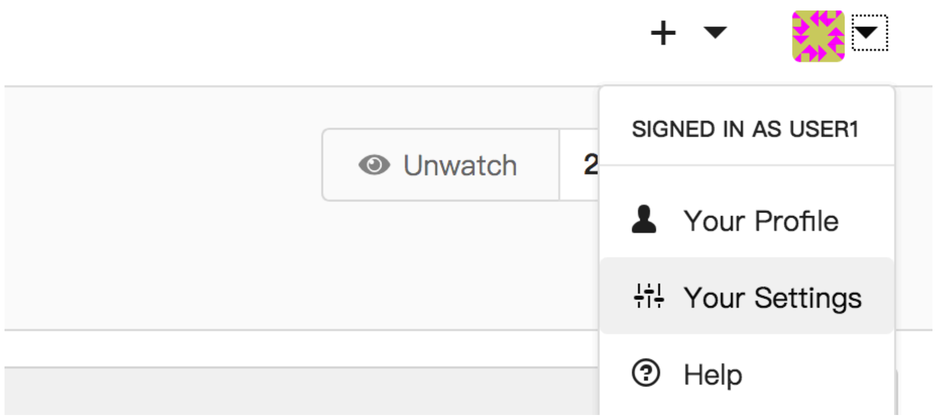
☑ Enable Captcha

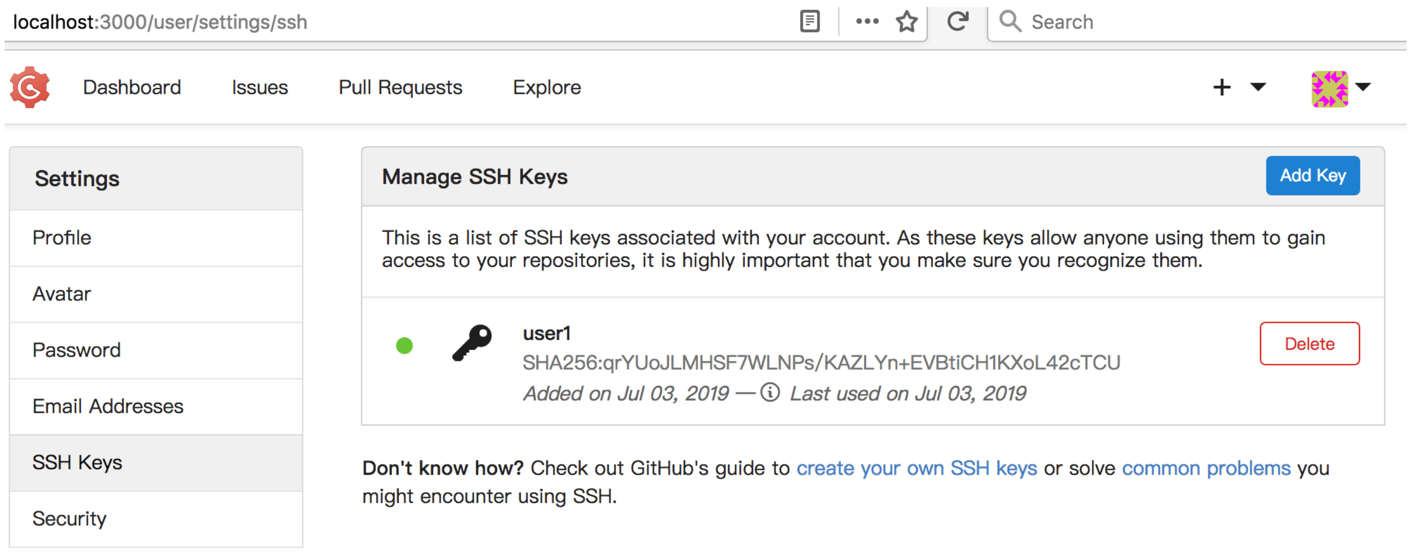
☐ Enable Require Sign In to View Pages

▾ Admin Account Settings

You do not have to create an admin account right now, user whoever ID=1 will gain admin access automatically.

Username	user1
Password	•••••••
Confirm Password	•••••••
Admin Email	user1@meetup.com





This is a list of SSH keys associated with your account. As these keys allow anyone using them to gain access to your repositories, it is highly important that you make sure you recognize them.

user1
SHA256:qrYUoJLMHSF7WLNPs/KAZLYn+EVBtiCH1KXoL42cTCU
Added on Jul 03, 2019 — Last used on Jul 03, 2019

Don't know how? Check out GitHub's guide to create your own SSH keys or solve common problems you might encounter using SSH.

We will create local repos step by step:

```
mkdir ~/ git-repos
cd ~/ git-repos
mkdir my_repo
cd my_repo

echo "My repo for learning git" > README.md
git init     ## Is there is a .git hidden directory under my_repo now?
git add README.md
git commit -m "My first commit"
git remote add origin git@localhost:my_org/my_repo.git
git push -u origin master
```

The .git folder is very important. git stores all local changes to files under this folder.

Git Concepts

What is git?

1. A distributed version control. Keeps record of your changes.
2. Started by Linux Torvald in 2005.
3. It is not the only version control, of course.
4. Most of the content below is inspired by a tutorial called "Ry's Git Tutorial" written by Ryan Hodson. You can find a copy at https://johnmathews.eu/rys-git-tutorial.html.

Why use it?

1. Allows for collaborative development.
2. Allows you to know who & when made the changes.
3. Allow you to go back in time.

What is so great about git?

1. Users keep entire code & history locally without internet access, except pulling & pushing code.
2. Fast branching & merging
3. PR review
4. Built in Wiki (Enterprise)

What kind of files are tracked with git?

Most of git's tools are designed to work with plain text files. But git supports any type of file, like xls. For a binary file, diff/merge will not work.

What are git tracking branches & workflow?

When working with git, it convenient to arrange the flow (from dev to prod) in a series of branches. Here are a few very common branches and the flow associated with it. The names are only convention.

master

This is the default branch automatically created during repo creation. This can also be the branch from which code gets deployed to prod.

release/stage

This branch is where code changes from one or more developers are merged for a release and deployed for UAT. After UAT, code from here gets merged into master branch for prod deployment.

feature

These are branches created by individual developers for each feature/bugfix devs are working now. Once unit testing is complete, code from multiple feature branch can be merged into stage for UAT deployment.

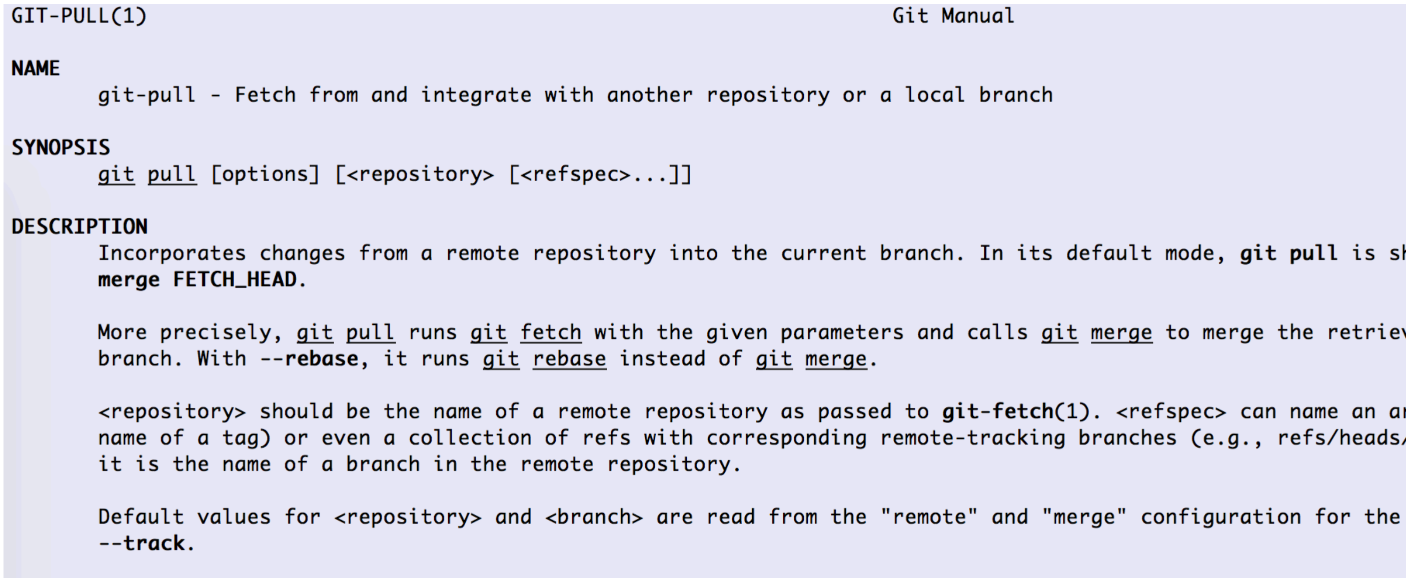
Once in a while, individual developers can merge their working code with the updates from the stage branch with "git pull" command.

On hands:

For the on hand part, we will use the following userids:
admin2/admin123
user1/user123
user2/user123

Getting help from command line:

```
git-pull --help
```



```
GIT-PULL(1)                                                    Git Manual

NAME
       git-pull - Fetch from and integrate with another repository or a local branch

SYNOPSIS
       git pull [options] [<repository> [<refspec>...]]

DESCRIPTION
       Incorporates changes from a remote repository into the current branch. In its default mode, git pull is sh
       merge FETCH_HEAD.

       More precisely, git pull runs git fetch with the given parameters and calls git merge to merge the retriev
       branch. With --rebase, it runs git rebase instead of git merge.

       <repository> should be the name of a remote repository as passed to git-fetch(1). <refspec> can name an ar
       name of a tag) or even a collection of refs with corresponding remote-tracking branches (e.g., refs/heads/
       it is the name of a branch in the remote repository.

       Default values for <repository> and <branch> are read from the "remote" and "merge" configuration for the
       --track.
```

Adding the First File

Create `index.html` in `my_repo` directory with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>A Colorful Website</title>
  <meta charset="utf-8" />
</head>
<body>
  <h1 style="color: #07F">A Colorful Website</h1>
  <p>This is a website about color!</p>

  <h2 style="color: #C00">News</h2>
  <ul>
    <li>Nothing going on (yet)</li>
  </ul>
</body>
</html>
```

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

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

	index.html

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

An untracked file is one that is not under version control. Git doesn't automatically track files because there are often project files that we don't want to keep under revision control. To keep a project small and efficient, you should only track source files and omit anything that can be generated from those files.

Let's make a smal change in your README.md

```
echo "Some text" >> README.md
```

```
git diff
diff --git a/README.md b/README.md
index 6079123..b0ae87b 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 My repo for learning git
+Some text
```

Git diff is useful to find out what you have changed in the repository before you actually commit. By default, this produces color-coded output.

Ignoring Files with `.gitignore`

If you want to make sure a file isn't accidentally checked in, you can add it into the .gitignore file. For example, if you use intelliJ to manage source, it will create a .idea file, which is local to you and not needed to be checked in into the remote.

```
echo "My scrap file" > scrap.txt
```

```
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 checkout -- <file>..." to discard changes in working directory)
```

```
        modified:   README.md

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

        index.html
        scrap.txt

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

echo "scrap.txt" > .gitignore
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 checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

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

        .gitignore
        index.html

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

We will checkin the `.gitignore` file to remote. Note that other developers may add their own entries to this file.

At this point, we have one modified file, and two new file in the local repo. We will push all of this files to remote.

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

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        modified:   README.md
        new file:   index.html

git commit -m "Added index & .gitignore"
[master c0454d2] Added index & .gitignore
 3 files changed, 20 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 index.html

git push origin master
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 586 bytes | 586.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To localhost:my_org/my_repo.git
   5159f65..c0454d2  master -> master
```

Now let's assume, the changes above were commited already in past by someone else, and you are adding a feature for the first time. Let rename the directory:

```
cd ~/ git-repos
mv my_repo my_repo_bak
git clone git@localhost:my_org/my_repo.git
Cloning into 'my_repo'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 12 (delta 1), reused 0 (delta 0)
Receiving objects: 100% (12/12), 1.27 KiB | 1.27 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Now git will bring down the repo from remote server and create the directory.

Let's repeat this process by adding a new file:

```
vi blue.html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>The Blue Page</title>
  <meta charset="utf-8" />
</head>
<body>
  <h1 style="color: #00F">The Blue Page</h1>
  <p>Blue is the color of the sky.</p>
</body>
</html>

vi index.html
- <li>Nothing going on (yet)</li>
+ <li>Check out our new file: <a href="blue.html">Blue</a></li>

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 checkout -- <file>..." to discard changes in working directory)

	modified:   index.html

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

	blue.html

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

git add .

git commit -m "Added blue file"
[master f19976a] Added blue file
 2 files changed, 13 insertions(+), 1 deletion(-)
 create mode 100644 blue.html

git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 576 bytes | 576.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To localhost:my_org/my_repo.git
   c0454d2..f19976a  master -> master
```

DEMO: We can view all of the changes we made using the UI.

We can also view the changes locally:

```
git log
commit f19976af580e404f7f93a3b6ec6bdf51250a5fea (HEAD -> master, origin/master)
Author: user1 <user1@meetup.com>
Date:   Thu Jul 4 09:16:27 2019 -0500

    Added blue file

commit c0454d2d6f551e1b246f3a677f3fd20c5831c7ef
Author: user1 <user1@meetup.com>
Date:   Thu Jul 4 08:26:07 2019 -0500

    Added index & .gitignore

commit 5159f65ad534cf83f0999478e6c08e61705a9d77
Author: user1 <user1@meetup.com>
Date:   Thu Jul 4 07:49:38 2019 -0500

    My first commit
```

A compact version of the log command is:

```
git log --oneline
f19976a (HEAD -> master, origin/master) Added blue file
c0454d2 Added index & .gitignore
5159f65 My first commit
```

View an old revision (command line):

```
git checkout c0454d2
Note: checking out 'c0454d2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at c0454d2... Added index & .gitignore
```

The output above has a lot of information about a detached HEAD state. You can ignore it for now. All you need to know is that the above command turns your my_repo directory into an exact (virtual) replica of the second snapshot (`c0454d2`) we committed.

```
git status
HEAD detached at c0454d2
nothing to commit, working tree clean

git log --oneline
c0454d2 (HEAD) Added index & .gitignore
5159f65 My first commit

ls -al
total 32
drwxr-xr-x   7 daich  staff  238 Jul  4 09:41 ./
drwxr-xr-x   4 daich  staff  136 Jul  4 08:46 ../
drwxr-xr-x  12 daich  staff  408 Jul  4 09:42 .git/
-rw-r--r--   1 daich  staff   10 Jul  4 08:14 .gitignore
-rw-r--r--   1 daich  staff   35 Jul  4 07:59 README.md
-rw-r--r--   1 daich  staff  320 Jul  4 09:41 index.html
-rw-r--r--   1 daich  staff   14 Jul  4 08:04 scrap.txt
```

The `blue.html` file is not shown. To revert back:

```
git checkout master
Previous HEAD position was c0454d2... Added index & .gitignore
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
```

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

nothing to commit, working tree clean

git log --oneline
f19976a (HEAD -> master, origin/master) Added blue file
c0454d2 Added index & .gitignore
5159f65 My first commit

ls -al
total 40
drwxr-xr-x   8 daich  staff  272 Jul  4 09:42 .
drwxr-xr-x   5 daich  staff  170 Jul  4 09:50 ..
drwxr-xr-x  12 daich  staff  408 Jul  4 09:50 .git
-rw-r--r--   1 daich  staff   10 Jul  4 08:14 .gitignore
-rw-r--r--   1 daich  staff   35 Jul  4 07:59 README.md
-rw-r--r--   1 daich  staff  214 Jul  4 09:42 blue.html
-rw-r--r--   1 daich  staff  350 Jul  4 09:42 index.html
-rw-r--r--   1 daich  staff   14 Jul  4 08:04 scrap.txt
```

Branching & pull request
A branch is a replica of the code, isolated from the **master** branch. It allows the developer the ability to add/remove functionality, experiment, etc. and depending on the result, the developer can either merge the code into the **master** branch (or any other branch for that matter) or simply delete it.

```
git branch
* master
```

Let's add a new branch for the feature we will be working:

```
git checkout -b my_branch
Switched to a new branch 'my_branch'

git branch
  master
* my_branch
```

We will add a new file.

```
vi orange.html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>The Orange Page</title>
  <meta charset="utf-8" />
</head>
<body>
  <h1 style="color: #F90">The Orange Page</h1>
  <p>Orange is so great it has a
  <span style="color: #F90">fruit</span> named after it.</p>
</body>
</html>
```

We will alos update index.html

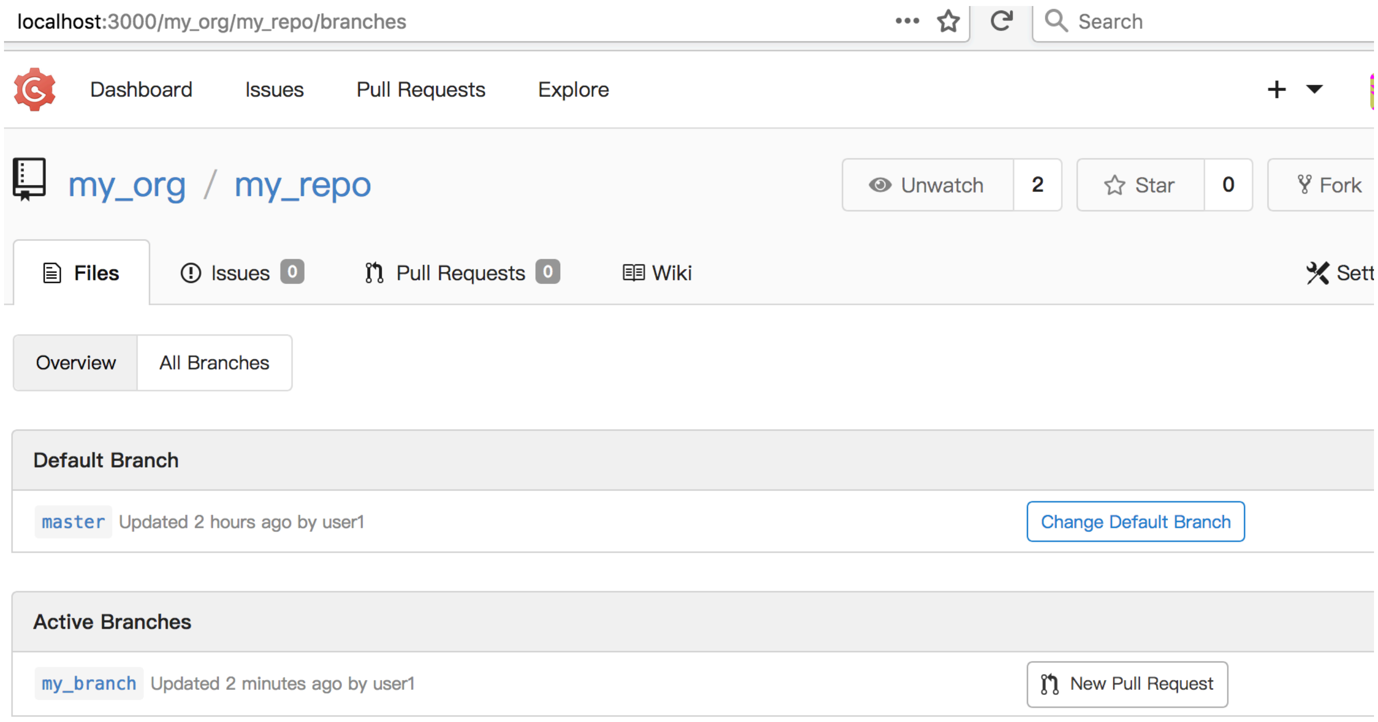
```
vi index.html
+ <li>Also check out: <a href="orange.html">Orange</a></li>

git status
On branch my_branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

	modified:   index.html

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

	orange.html

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

git diff
diff --git a/index.html b/index.html
index 7c0cd03..394eb5f 100644
--- a/index.html
+++ b/index.html
@@ -11,6 +11,7 @@
   <h2 style="color: #C00">News</h2>
   <ul>
     <li>Check out our new file: <a href="blue.html">Blue</a></li>
+    <li>Also check out: <a href="orange.html">Orange</a></li>
   </ul>
 </body>
 </html>
```

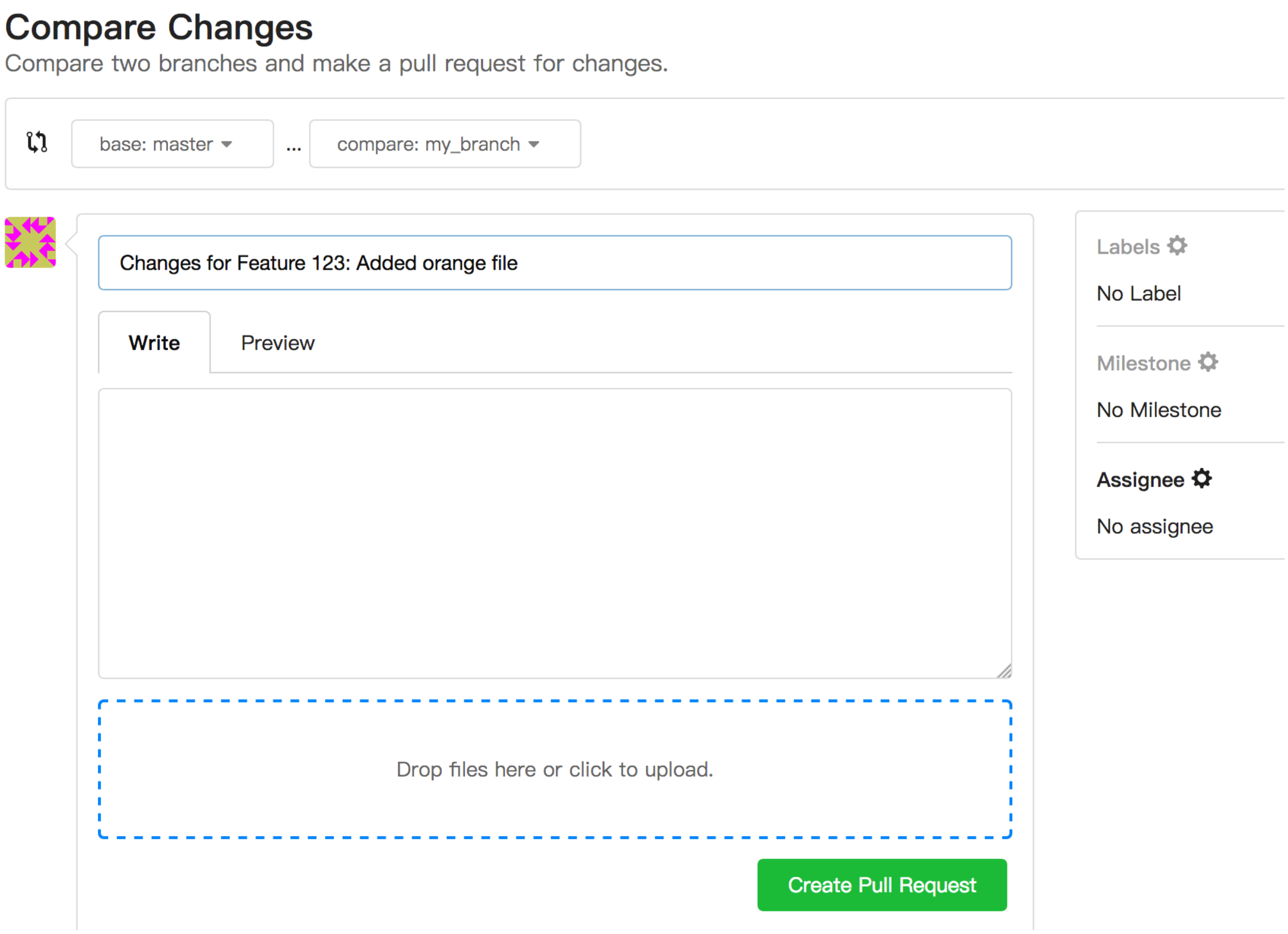
All of these changes are happenniing in my_branch.

```
git add .
git status
On branch my_branch
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

	modified:   index.html
	new file:   orange.html
```

```
git commit -m "Added orange file"
[my_branch 4df8f16] Added orange file
 2 files changed, 14 insertions(+)
 create mode 100644 orange.html

git push origin my_branch
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 552 bytes | 552.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To localhost:my_org/my_repo.git
 * [new branch]      my_branch -> my_branch
```

Now we will merge using pull request from the UI.





Compare Changes

Compare two branches and make a pull request for changes.

Changes for Feature 123: Added orange file

http://localhost:3000/my_org/my_repo/pulls/1

To view all branches:

```
git branch -a
```

To delete a branch locally:

```
git branch -d
git branch -D
```
(force)

Recover deleted file

```
rm index.html
```

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

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

	deleted:    index.html
```

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

git ls-files --deleted
index.html

git checkout -- index.html

git ls-files --deleted
ls -al
```

Recovering from accidental local commit

```
echo "Oops" > index.html
** git diff

git add index.html

git commit -m "More changes to index file"
[master b874323] More changes to index file
 1 file changed, 1 insertion(+), 18 deletions(-)

git log --oneline
7923ca4 (HEAD) More changes to index file
f19976a (origin/master, master) Added blue file
c0454d2 Added index & .gitignore
5159f65 My first commit


git revert 7923ca4

---------------------------------  vi  -------------------------------------
Revert "More changes to index file"

This reverts commit 7923ca4ecef1b1eca74ac4a0022374586c7d57af.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   modified:   index.html
#

~
----------------------------------------------------------------------------

[master 0471852] Revert "More changes to index file"
```

```
 1 file changed, 18 insertions(+), 1 deletion(-)



git log --oneline
0471852 (HEAD -> master) Revert "More changes to index file"
7923ca4 More changes to index file
f19976a (origin/master, origin/HEAD) Added blue file
c0454d2 Added index & .gitignore
5159f65 My first commit

cat index.html
```