GIT

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda

- Introduction to Git
  - What is version control?
  - What is GIT?
  - Centralized VCS
  - Distributed VCS (Git)
  - Centralized VCS vs Distributed VCS
  - How does git work? (Concepts)
    - Snapshots
    - Commit
    - Repositories
    - Branches
      - HEAD
      - MASTER
      - Branching off of the master branch
      - Merging
- Environment Setup
  - Configure your environment (GitLab)
    - Create a GitLab account
    - Login to your GitLab account
    - Create A GitLab Project
    - Configure SSH Access To The GitLab Project
    - Generating a new SSH key pair
    - Adding a SSH key to your GitLab account
  - Install GIT (Windows/Mac)
  - GitLab Visual Studio Code Extension
    - Create your Personal Access Token
    - Install GitLab Workflow Extension
    - Add token to GitLab Workflow Extension

Agenda

- Create Local Git repository
- Add new files to the repository
- Staging Environment
- Add files to the Staging Environment
- Create a commit
- Create a new branch
- Push a branch to GitLab
- Clone a Git Repository
  - Create A New GitLab Project
- Create a Pull Request (PR)
  - Pull from GitLab
- Merge a Pull Request
  - Merge conflicts

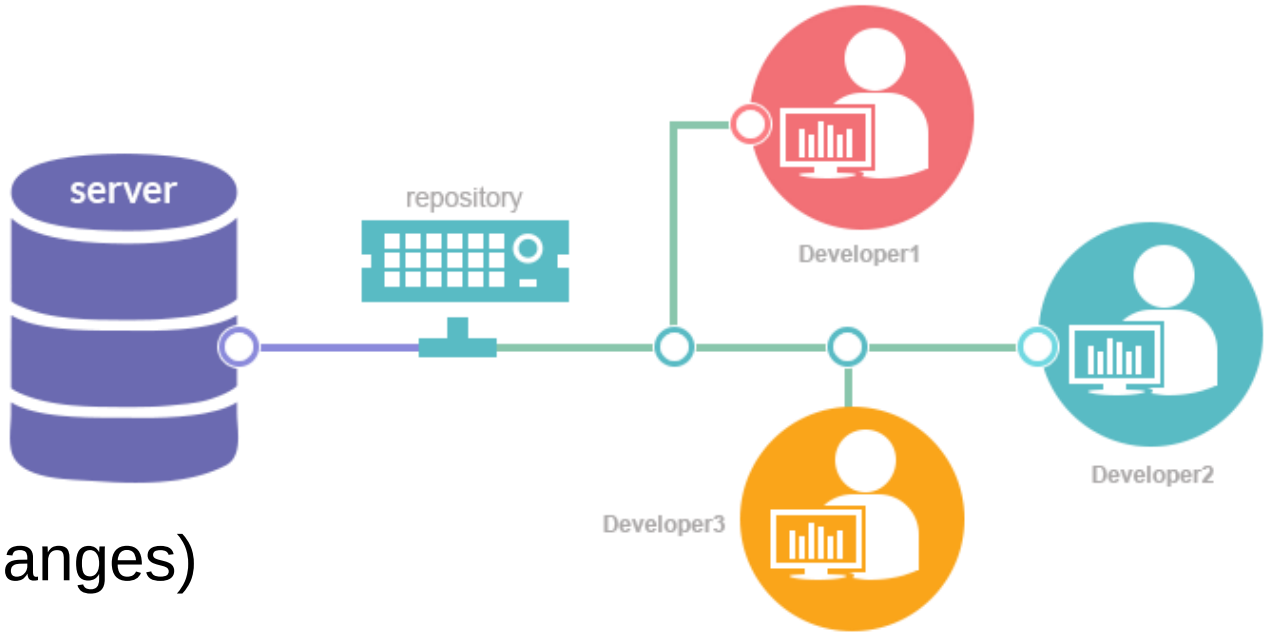
Introduction to Git

What is version control?

- A system for managing changes made to documents and other computer files

- **What kinds of files can we use it with?**
  - Source code
  - Documentation
  - Short stories
  - Binary files (music and pictures)

- **What should we use it for?**
  - Text files
  - Projects that have lots of revisions (changes)
  - Collaborating



What is GIT?

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

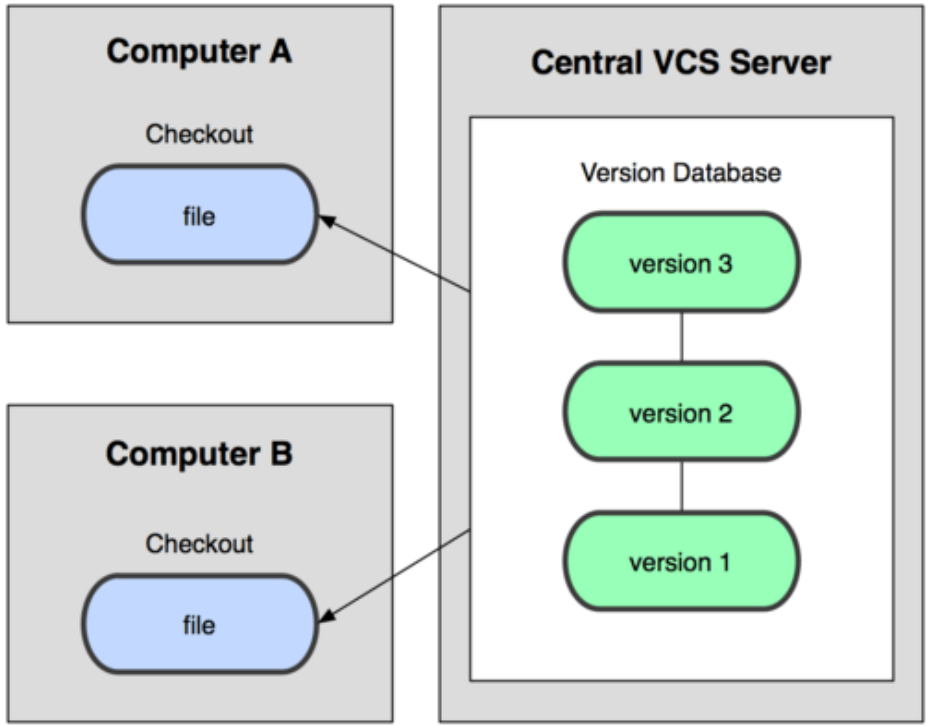
What is GIT? cont.

- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

- Git isn't the only version control system, there are different choices available:
  - CVS
  - SVN
  - Perforce
  - Mercurial (Hg)
  - Bazaar
  - And more

Centralized VCS

- In Subversion, CVS, Perforce, etc., a central server repository (repo) holds the "official copy" of the code.
  - The server maintains the sole version history of the repo.

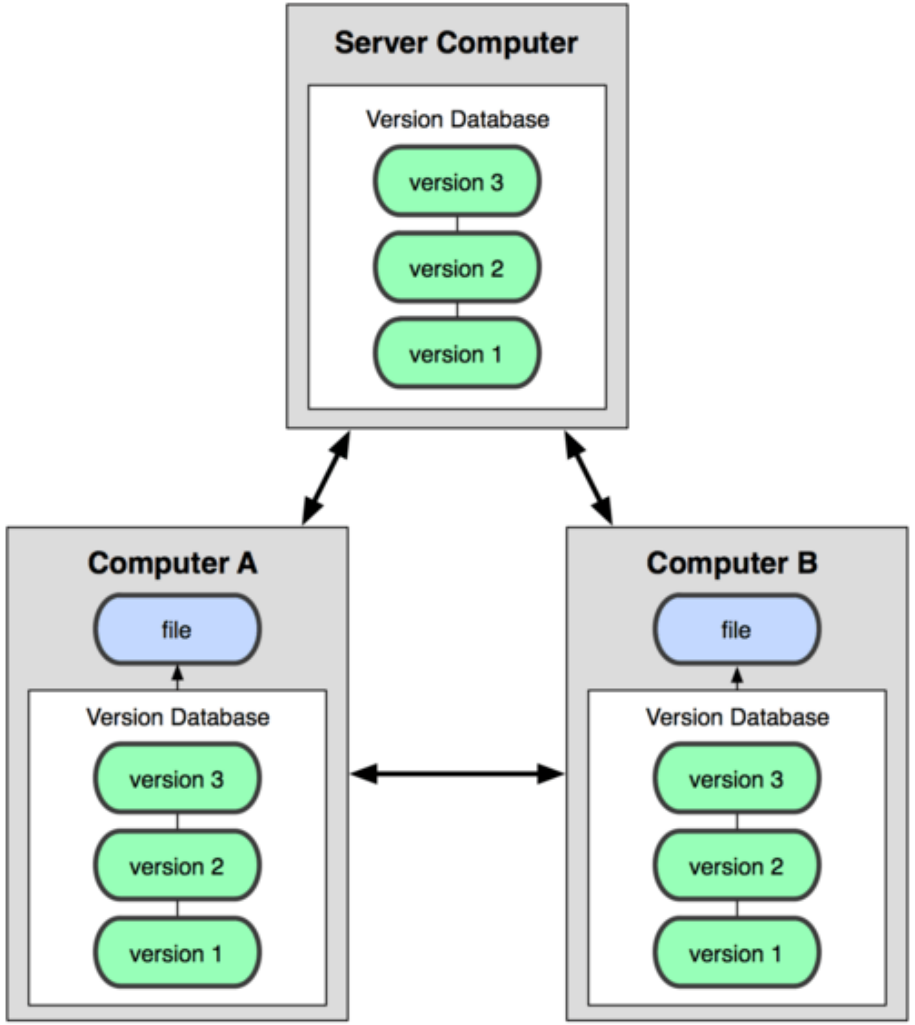


- You make "checkouts" of it to your local copy.
  - You make local modifications.
  - Your changes are not versioned.

- When you're done, you "check in" back to the server.
  - Your check-in increments the repo's version.

Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout" from a central repo.
  - you "clone" it and "pull" changes from it.
- Your local repo is a complete copy of everything on the remote server.
  - yours is "just as good" as theirs.
- Many operations are local:
  - check in/out from local repo.
  - commit changes to local repo.
  - local repo keeps version history.
- When you're ready, you can "push" changes back to server.



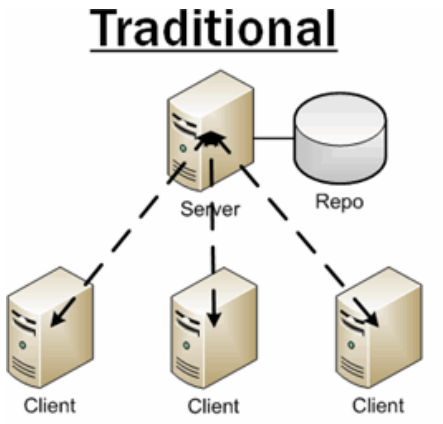
Centralized VCS vs Distributed VCS

Centralized VCS

- One central repository.

- Must be capable of connecting to repo.

- Need to solve issues with group members making different changes on the same files.



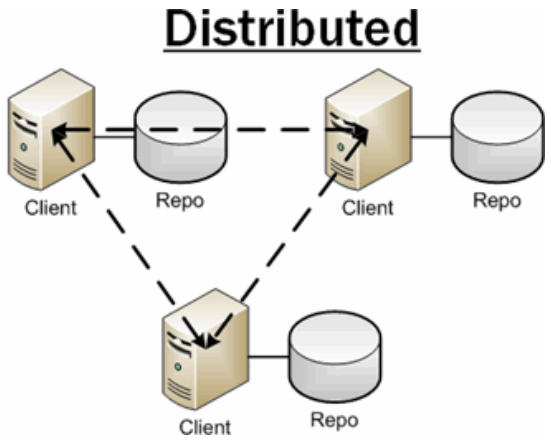
Distributed VCS

- Everyone has a working repo.

- Faster.

- Connectionless.

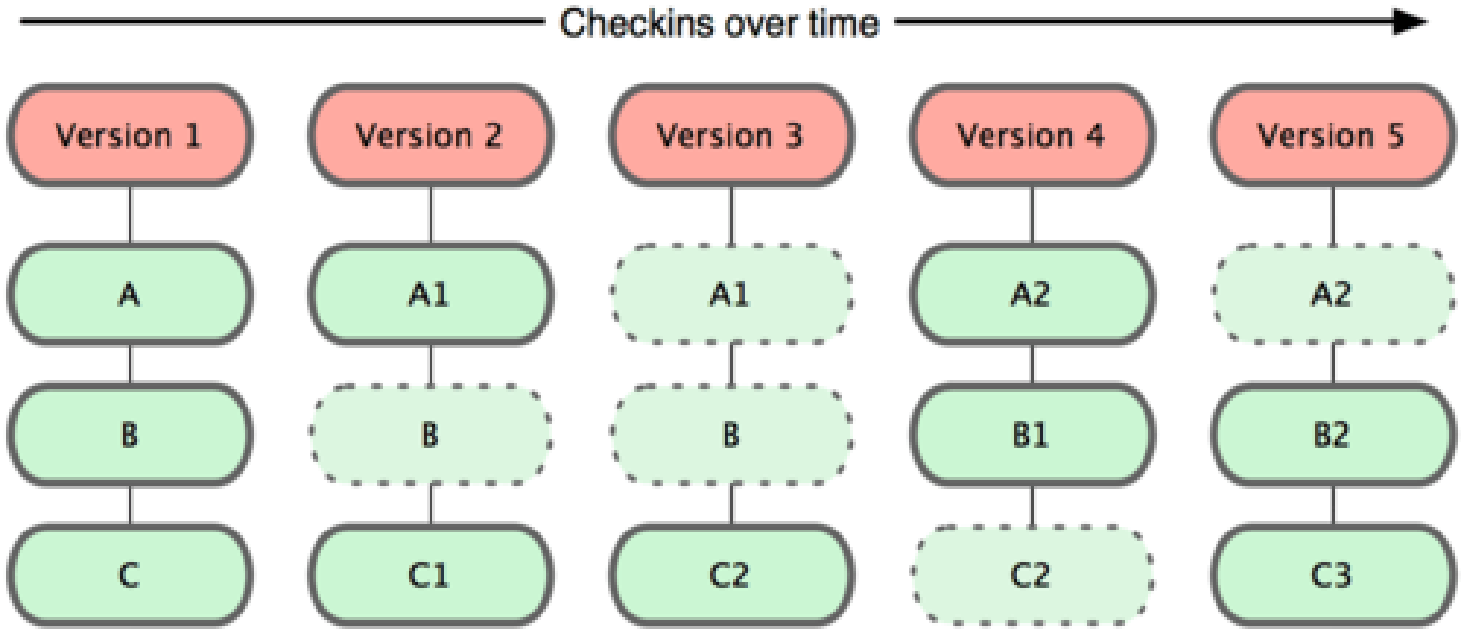
- Still need to resolve issues, but it's not an argument against DVCS.



How does git work? (Concepts)

Snapshots

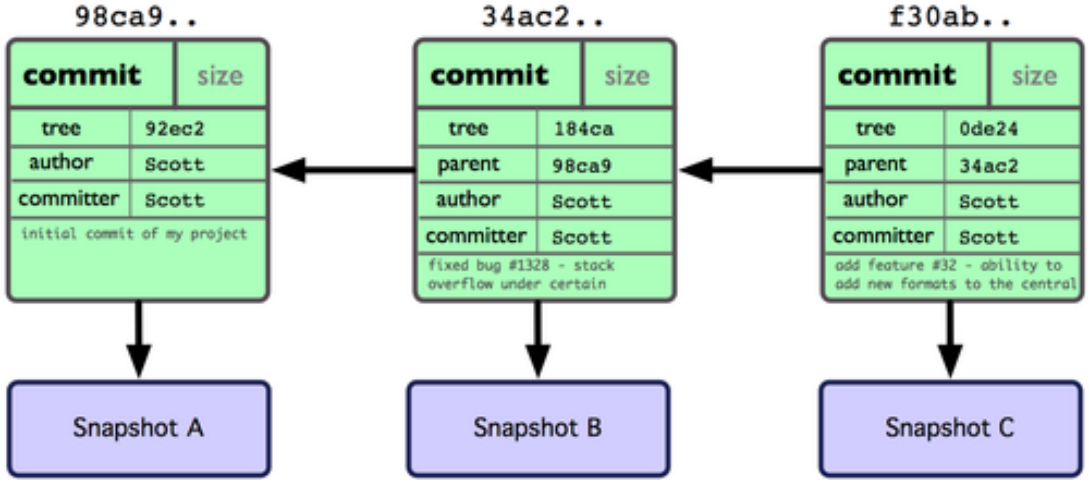
- The way git keeps track of your code history.
- Essentially records what all your files look like at a given point in time.
- You decide when to take a snapshot, and of what files.
- Have the ability to go back to visit any snapshot.
  - Your snapshots from later on will stay around, too.



How does git work? (Concepts)

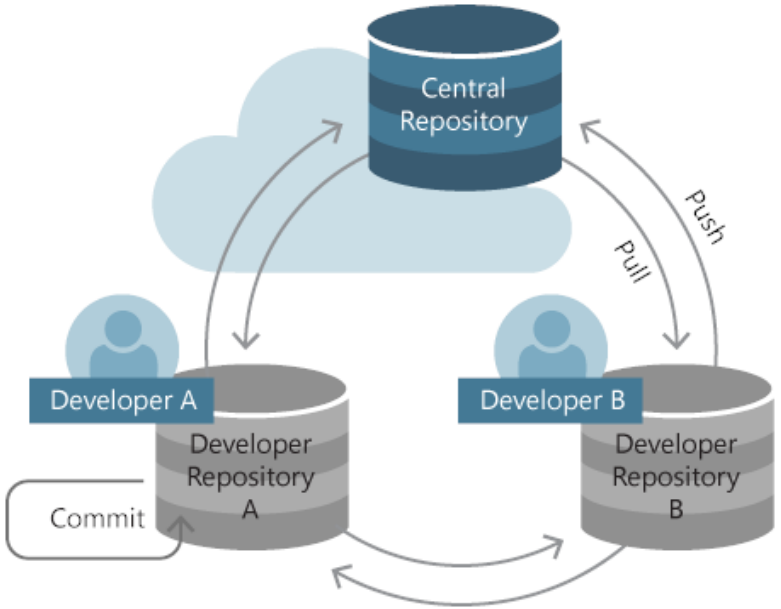
Commit

- The act of creating a snapshot
- Essentially, a project is made up of a bunch of commits
- Commits contain three pieces of information:
    1. Information about how the files changed from previously
    2. A reference to the commit that came before it
        - Called the *"parent commit"*
    3. A hash code name
        - Will look something like: *fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e*



How does git work? (Concepts)

Repositories



- Often shortened to ‘repo’.
- A collection of all the files and the history of those files .
  - Consists of all your commits.
  - Place where all your hard work is stored.
- Can live on a local machine or on a remote server.
- The act of copying a repository from a remote server is called cloning.
- Cloning from a remote server allows teams to work together.
- The process of downloading commits that don’t exist on your machine from a remote repository is called pulling changes.
- The process of adding your local changes to the remote repository is called pushing changes.

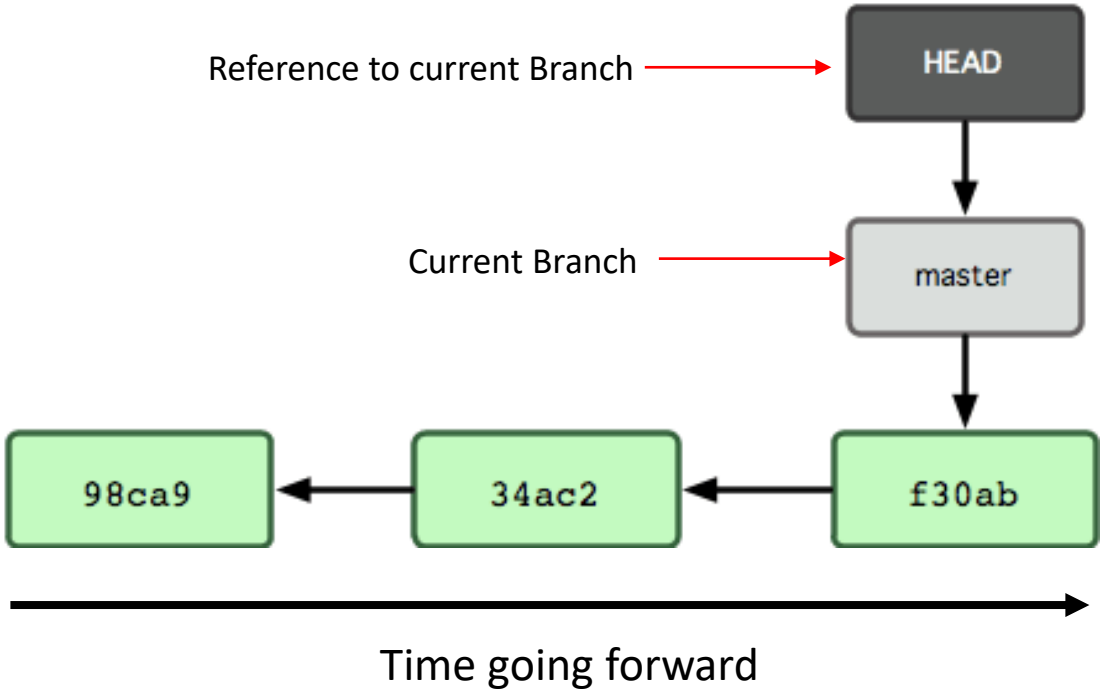
How does git work? (Concepts)

Branches

- All commits in git live on some branch.
- But there can be many, many branches.
- The main branch in a project is called the master branch.

So, what does a typical project look like?

- A bunch of commits linked together that live on some branch, contained in a repository.



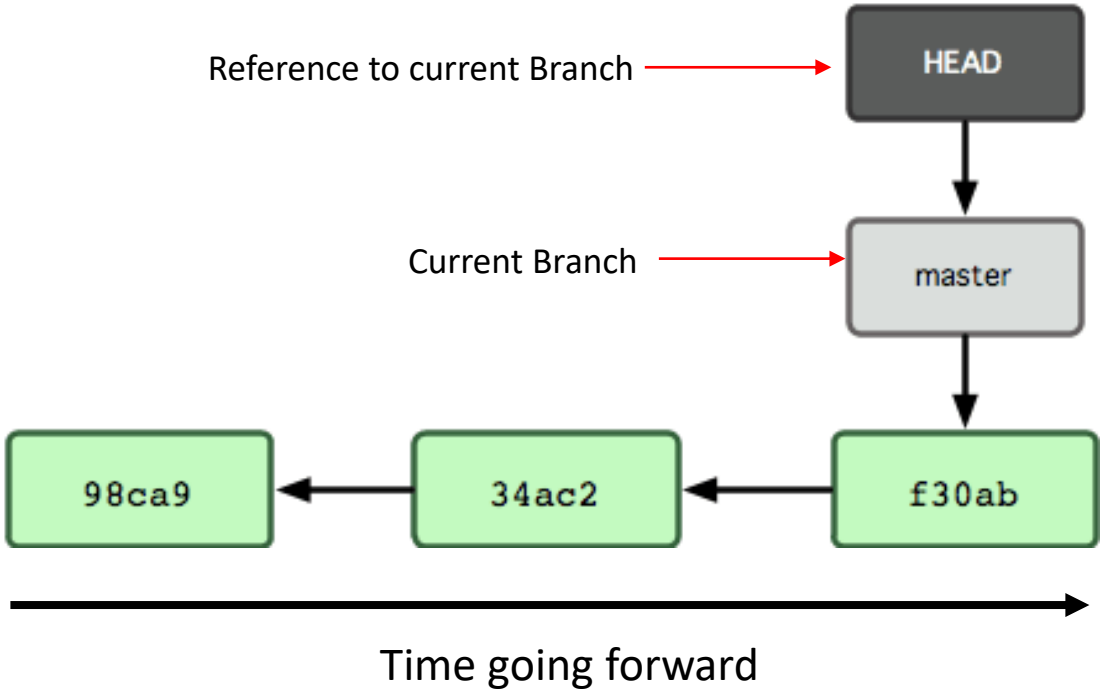
How does git work? (Concepts)

What is *HEAD*?

- A reference to the most recent commit.

What is *MASTER*?

- The main branch in your project (Doesn't have to be called *master*, but almost always is).



How does git work? (Concepts)

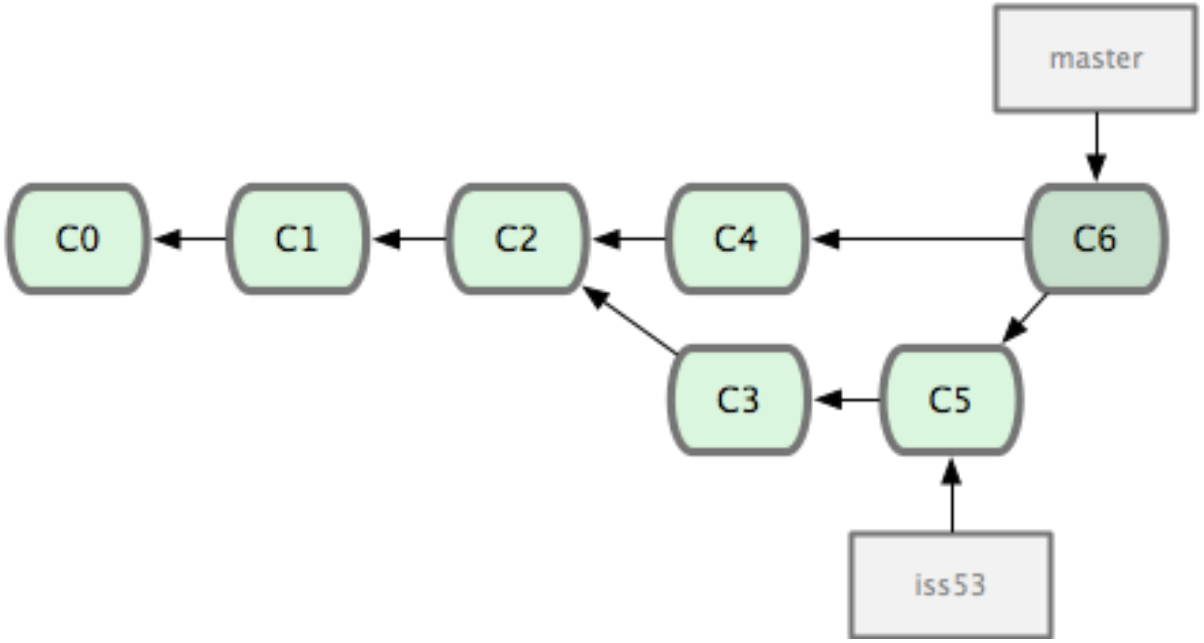
Branching off of the master branch

- The start of a branch points to a specific commit.

- When you want to make any changes to your project you make a new branch based on a commit.

Merging

- Once you're done with your feature, you merge it back into master.



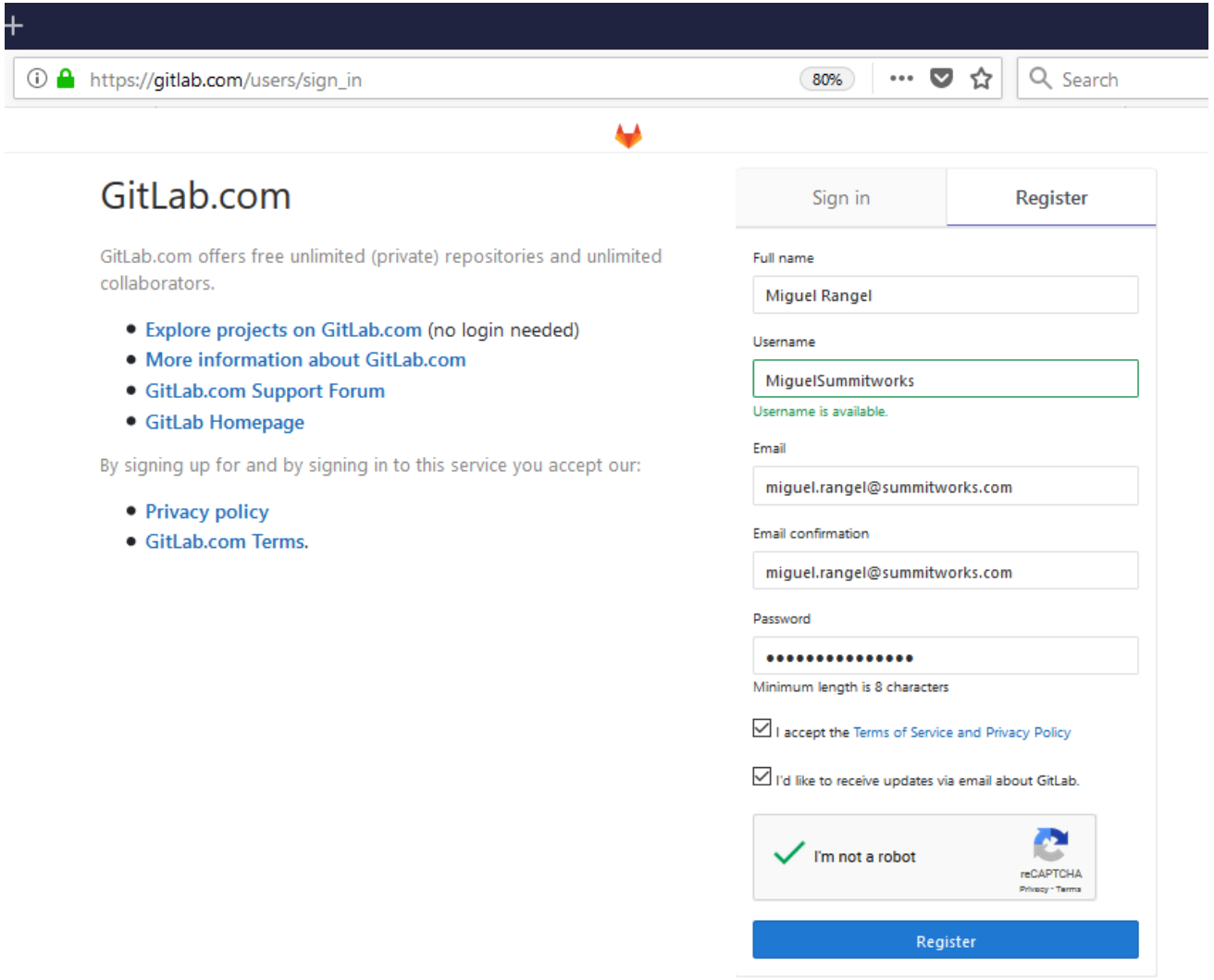
Environment Setup

Configure your environment (GitLab)

Create a GitLab account

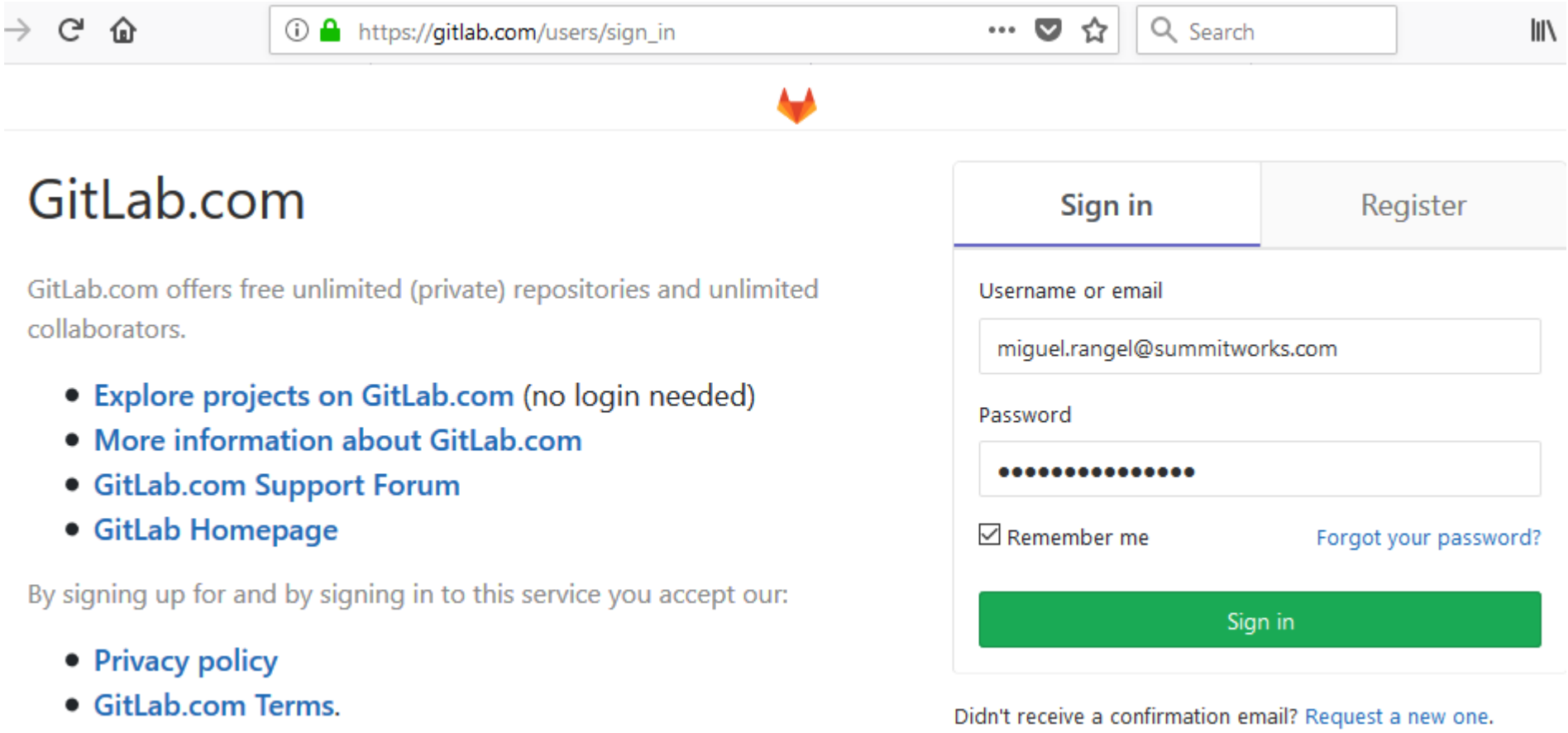
- Create a new user account on GitLab.

  https://gitlab.com/users/sign_in



Configure your environment (GITLab)

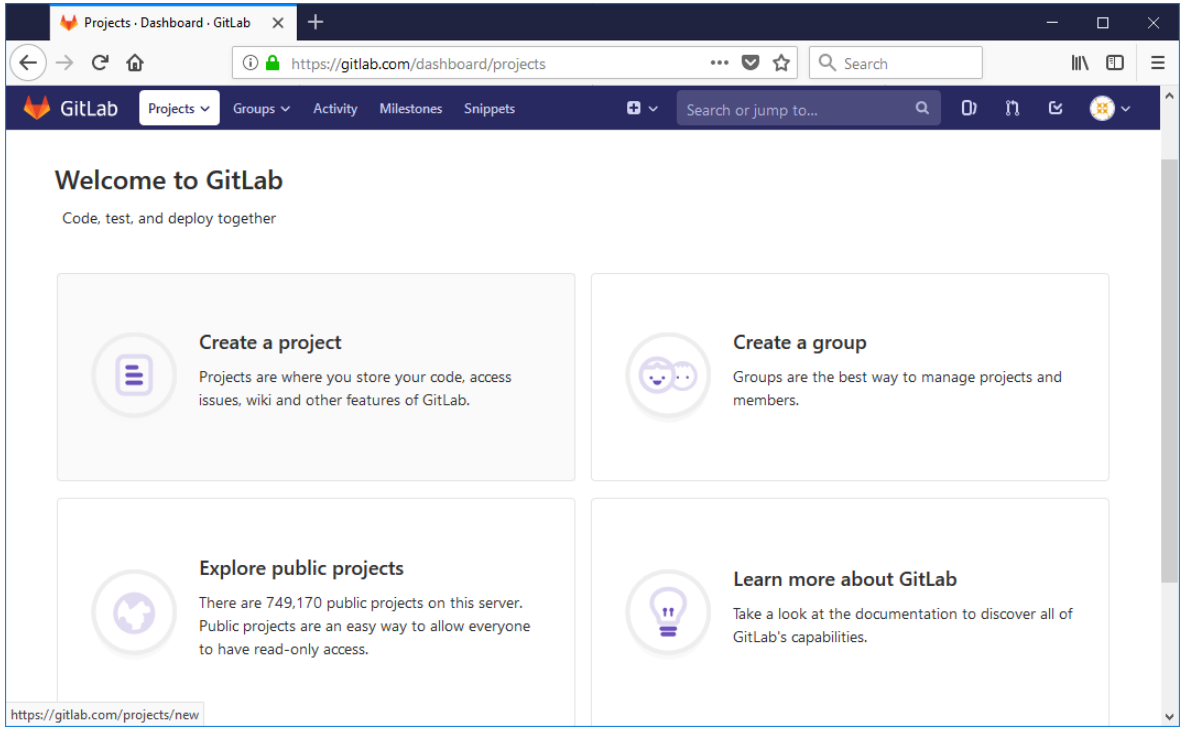
Login to your GitLab account



Configure your environment (GITLab)

Create A GitLab Project

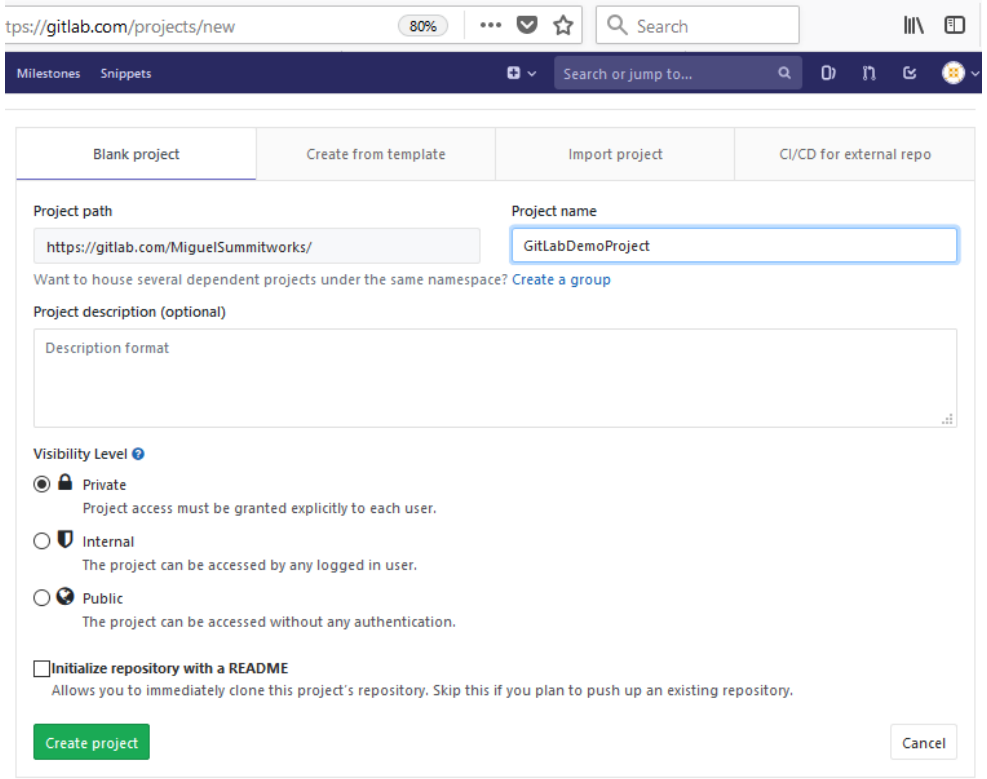
- Now, lets create a Git repository for your project source code using GitLab. This is where you and your team will push code changes.
- Click the “Create a Project" button.



Configure your environment (GITLab)

Create A GitLab Project

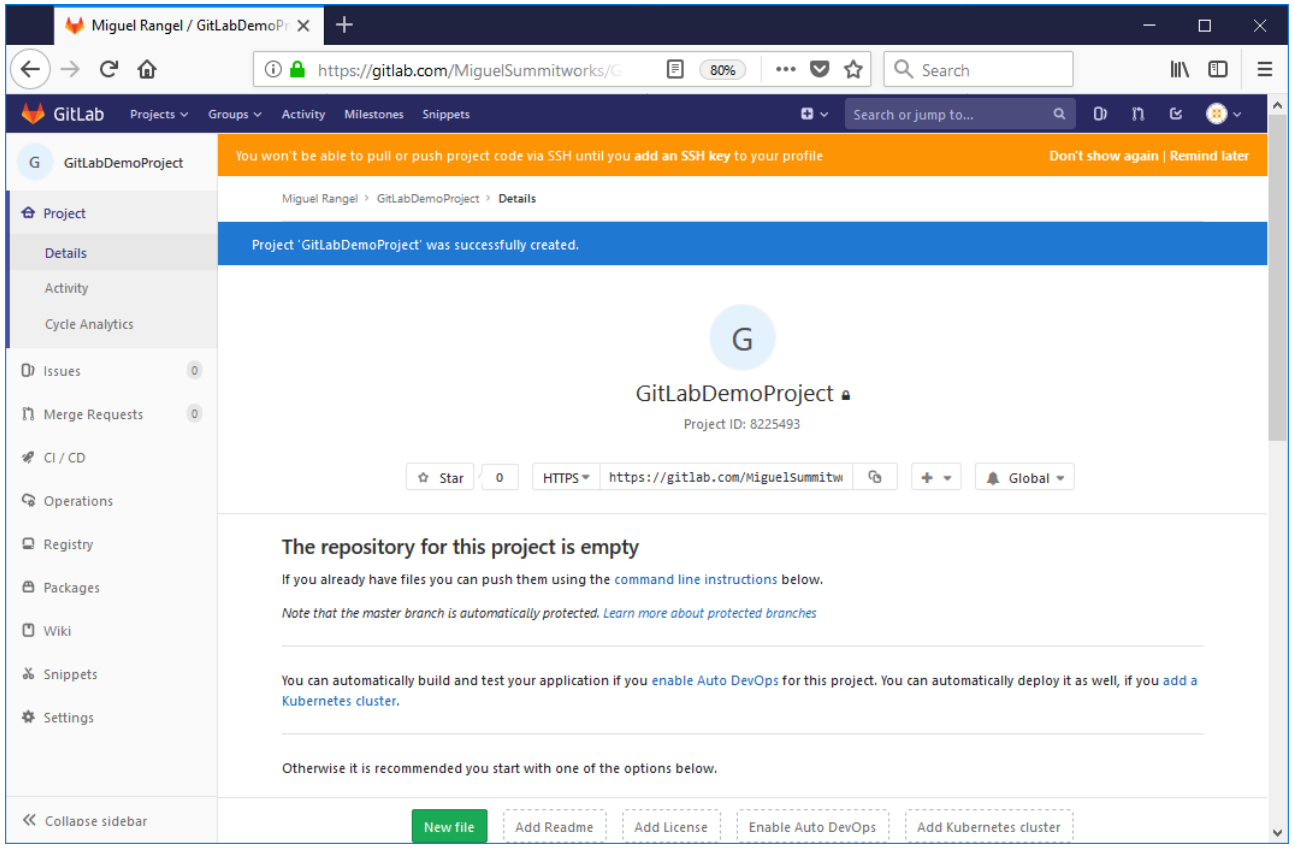
- On the "New Project" page, enter a name for the project and set the visibility level. The project in this example is set to "Public", but you can also choose to make your project "Private" or "Internal". Click "Create Project" once done.



Configure your environment (GITLab)

Create A GitLab Project

- GitLab will initialize an empty Git repository for the project, as shown below. Note the SSH URL to the repository, as you will need it in the next step.



Configure your environment (GITLab)

Configure SSH Access To The GitLab Project

- Your Git client will need access to the project repository, so this is a good time to configure that access.
- Click on “add an SSH key” link.

You won't be able to pull or push project code via SSH until you **add an SSH key** to your profile

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to generate one or use an existing key.

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_rsa.pub' and begins with 'ssh-rsa'. Don't use your private SSH key.

Typically starts with "ssh-rsa ..."

Title

SummitworksKey

Name your individual key via a title

Add key

Configure your environment (GITLab)

Windows

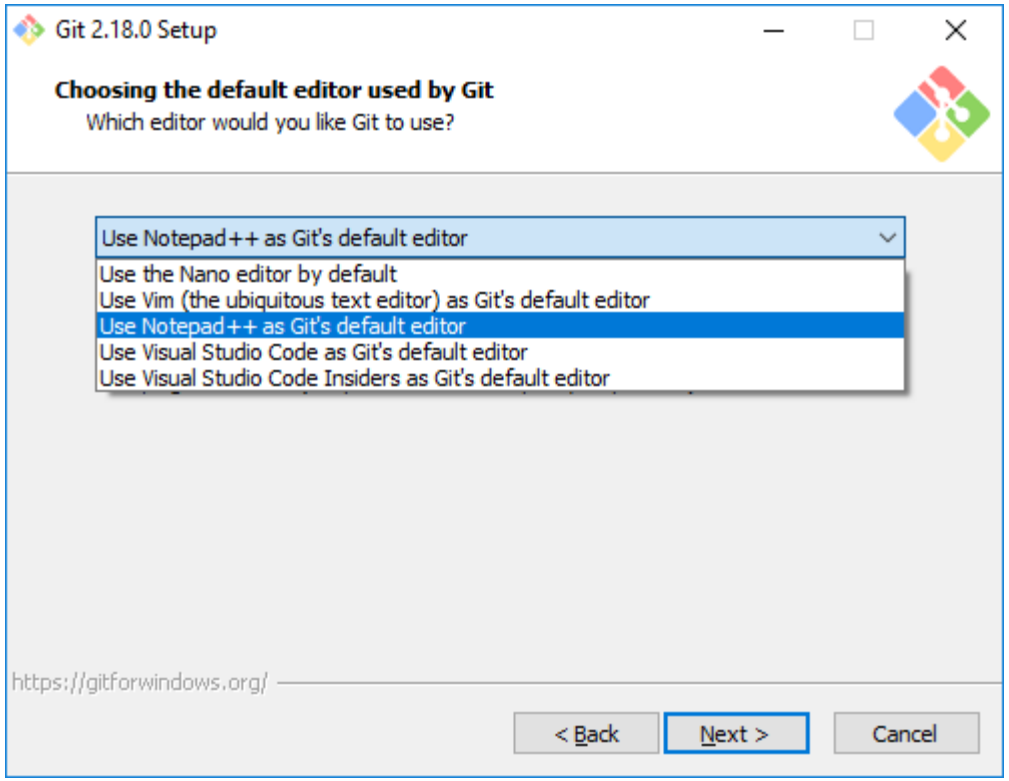
- Install 'Git for Windows' from https://git-for-windows.github.io

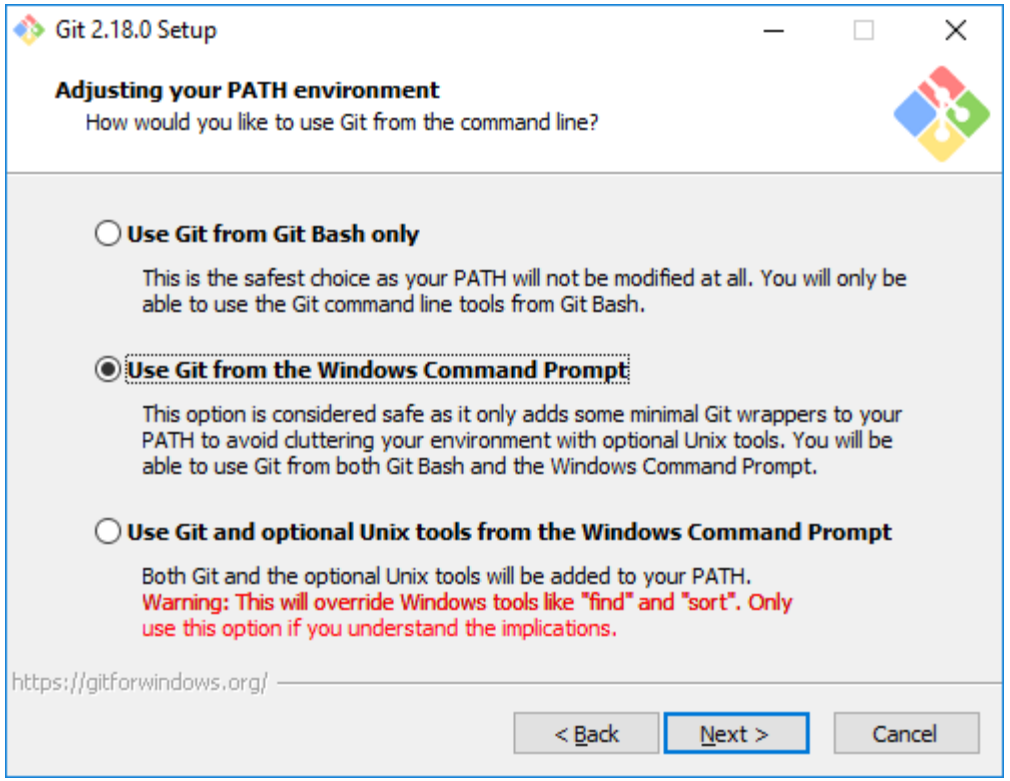
Mac

- Type ‘git' in the terminal application.
- If it's not installed, it will prompt you to install it.

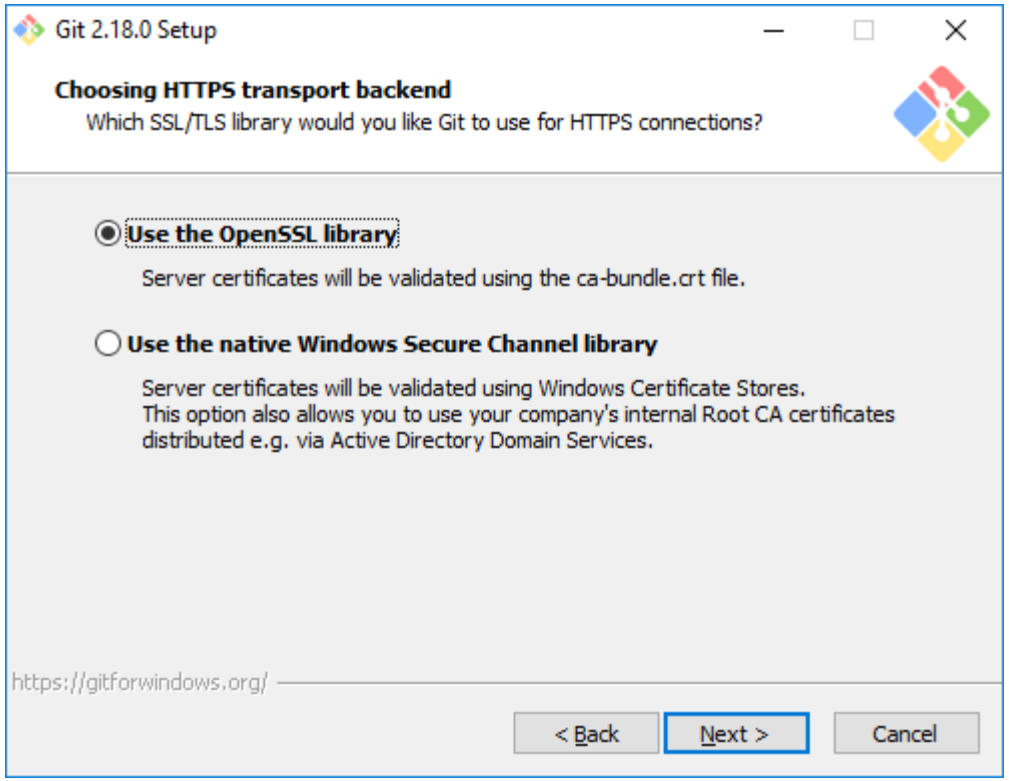
Configure your environment (GITLab)

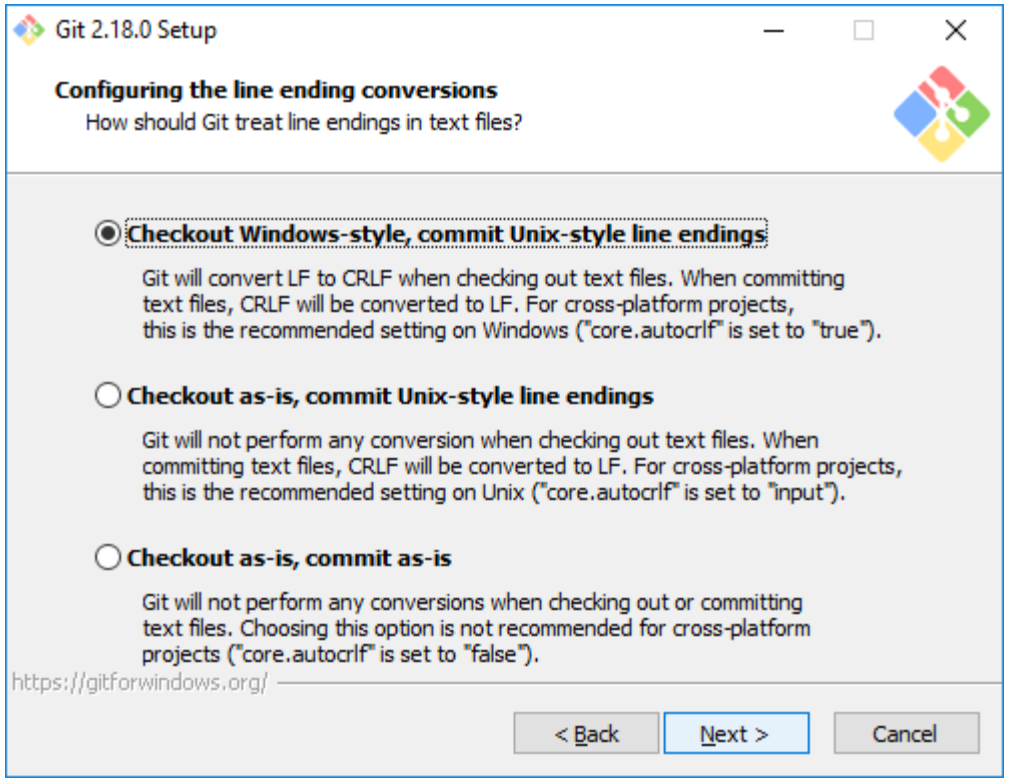
Configure your environment (GITLab)



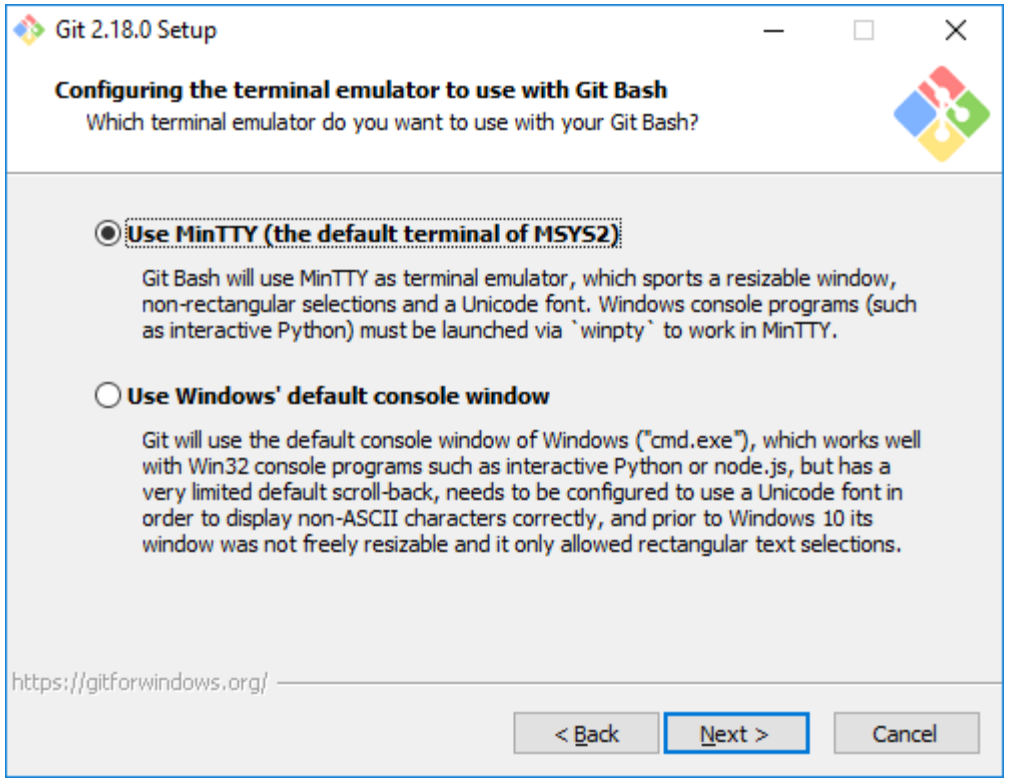


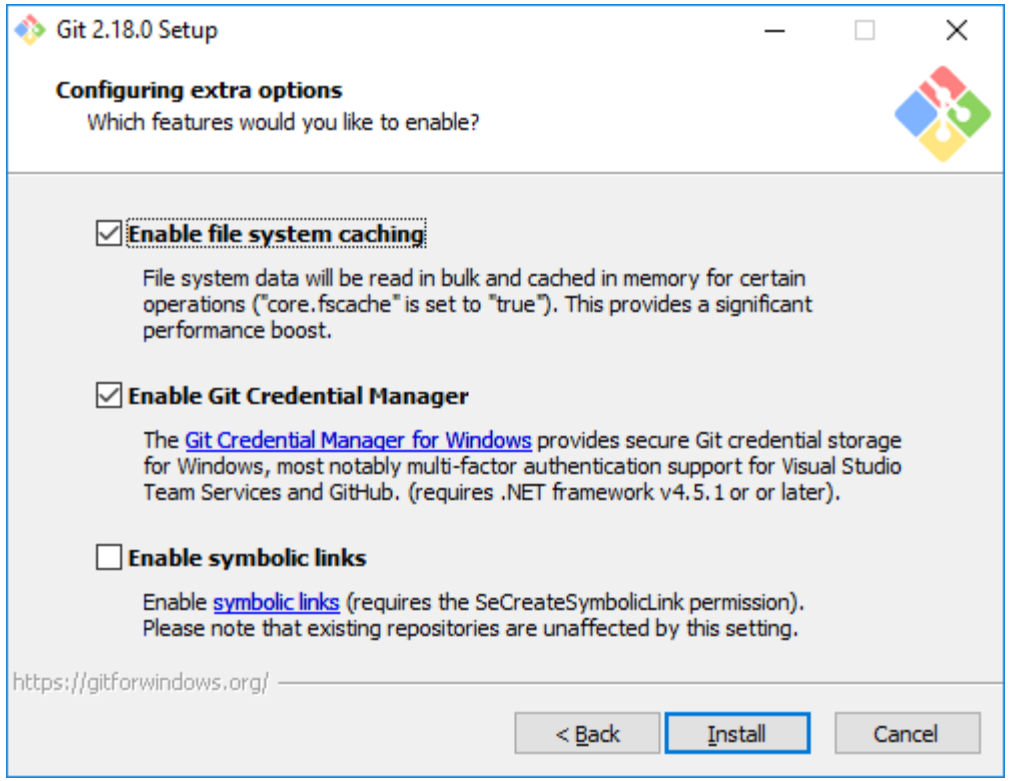
Configure your environment (GITLab)



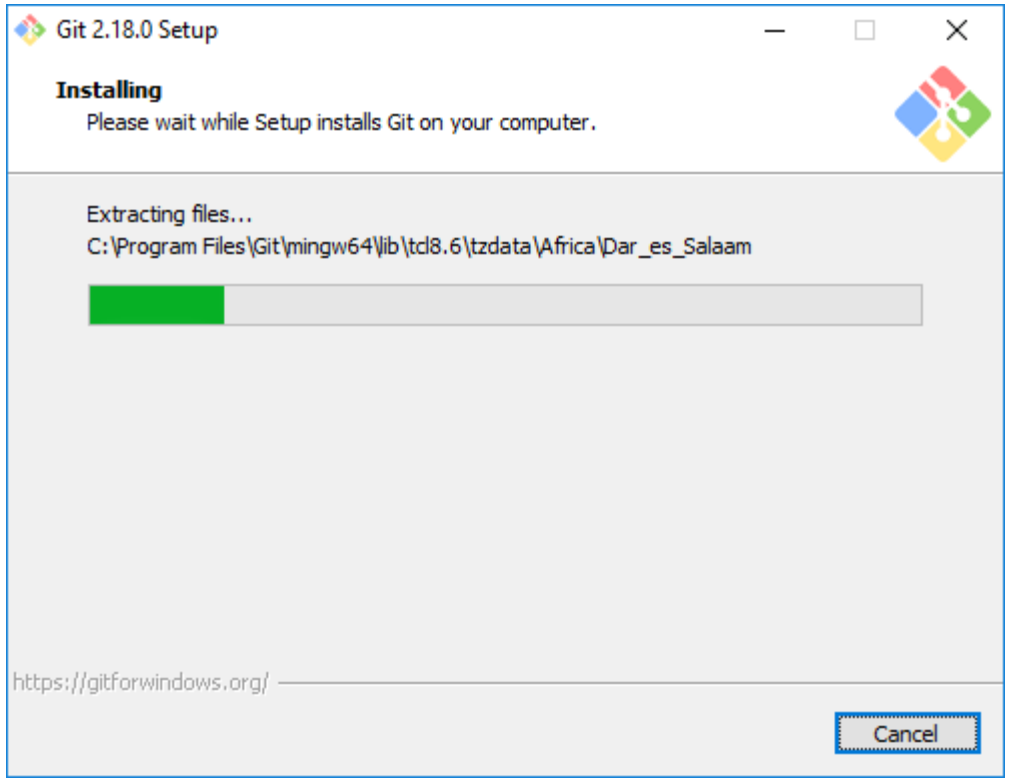


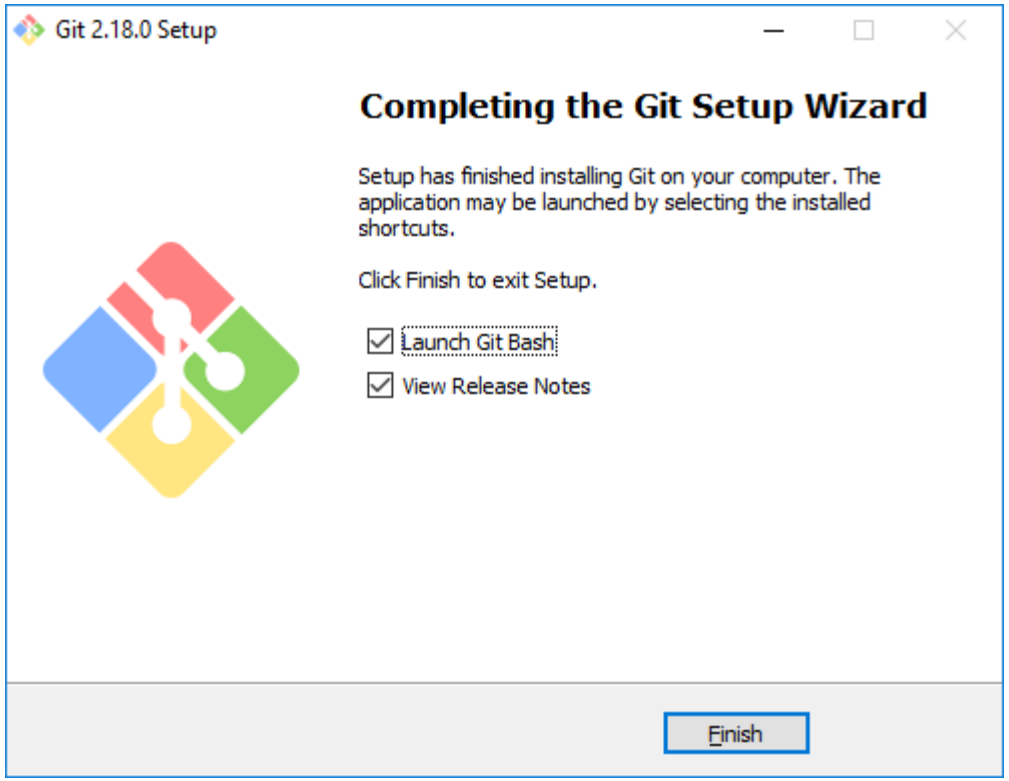
Configure your environment (GITLab)





Configure your environment (GITLab)





Configure your environment (GITLab)

Generating a new SSH key pair

- To generate a new SSH key pair, go to “Home” and navigate to “Git” folder, then open “Git Bash”.

- Use the following command:

```
ssh-keygen -o -t rsa -C "your.email@example.com" -b 4096
```

Configure your environment (GITLab)

Next, you will be prompted to input a file path to save your SSH key pair to.

Use the suggested path by pressing enter.

MINGW64:/c/Users/Oxezno

```
Oxezno@LAPTOP-6P6E1FTI MINGW64 ~ (master)
$ ssh-keygen -o -t rsa -C "miguel.rangel@summitworks.com" -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Oxezno/.ssh/id_rsa): |
```

Once you have input a file path you will be prompted to input a password to secure your SSH key pair. It is a best practice to use a password for an SSH key pair, but it is not required and you can skip creating a password by pressing enter.

MINGW64:/c/Users/Oxezno

```
Oxezno@LAPTOP-6P6E1FTI MINGW64 ~ (master)
$ ssh-keygen -o -t rsa -C "miguel.rangel@summitworks.com" -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Oxezno/.ssh/id_rsa):
/c/Users/Oxezno/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Oxezno/.ssh/id_rsa.
Your public key has been saved in /c/Users/Oxezno/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:xsnbs12iFlW74uwAoYKuDBr31TpqvybDv2sBWgBHigY miguel.rangel@summitworks.com
The key's randomart image is:
+---[RSA 4096]----+
|Eoo              |
|oo.            .  |
|o. .    .    . .  |
|.  .o  + o  . .  |
|  .o... S  .   .  |
| ..  ..o +. . .  |
|o o.  ..o ++...  |
|++ .=.+.  .*oo   |
|o. .oOB= .o.o    |
```

Configure your environment (GITLab)

Adding a SSH key to your GitLab account

The next step is to copy the public SSH key as we will need it afterwards.

To copy your public SSH key to the clipboard, use the appropriate code below:

macOS:

```
pbcopy < ~/.ssh/id_rsa.pub
```

GNU/Linux (requires the xclip package):

```
xclip -sel clip < ~/.ssh/id_rsa.pub
```

Windows Command Line:

```
type %userprofile%\.ssh\id_rsa.pub | clip
```

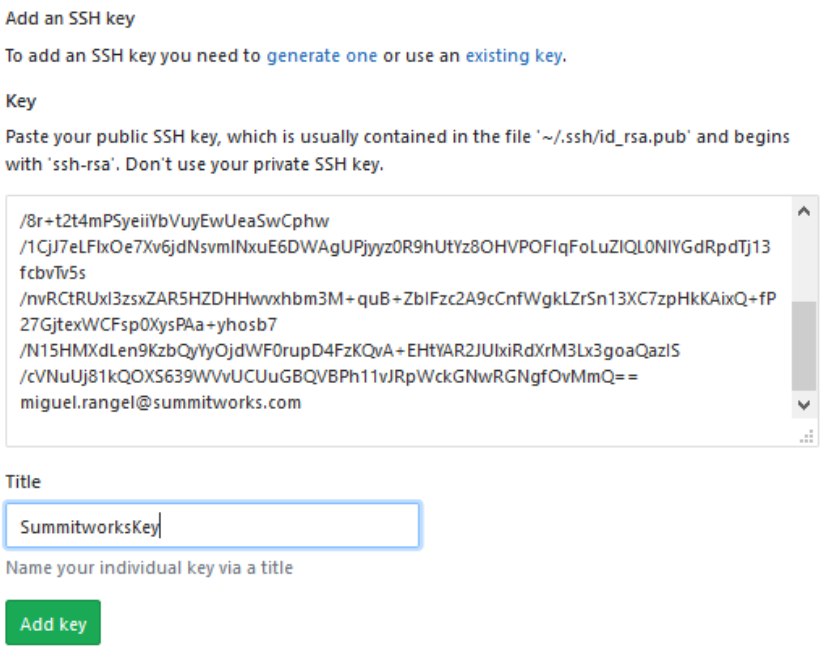
Git Bash on Windows / Windows PowerShell:

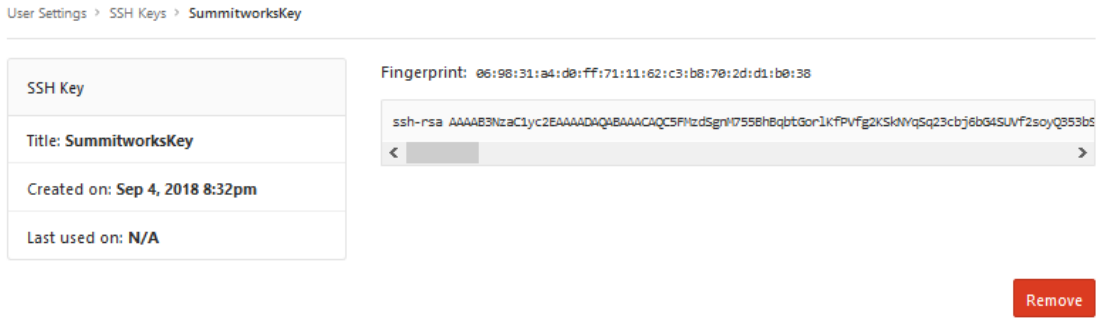
```
cat ~/.ssh/id_rsa.pub | clip
```

Configure your environment (GITLab)

The final step is to add your public SSH key to GitLab.

Navigate to the 'SSH Keys' tab in your 'Profile Settings'. Paste your key in the 'Key' section and give it a relevant 'Title'. Use an identifiable title like 'Work Laptop - Windows 7' or 'Home MacBook Pro 15'.





GitLab Visual Studio Code Extension

Setup

To use this extension, you need to create a GitLab Personal Access Token and give it to the extension.

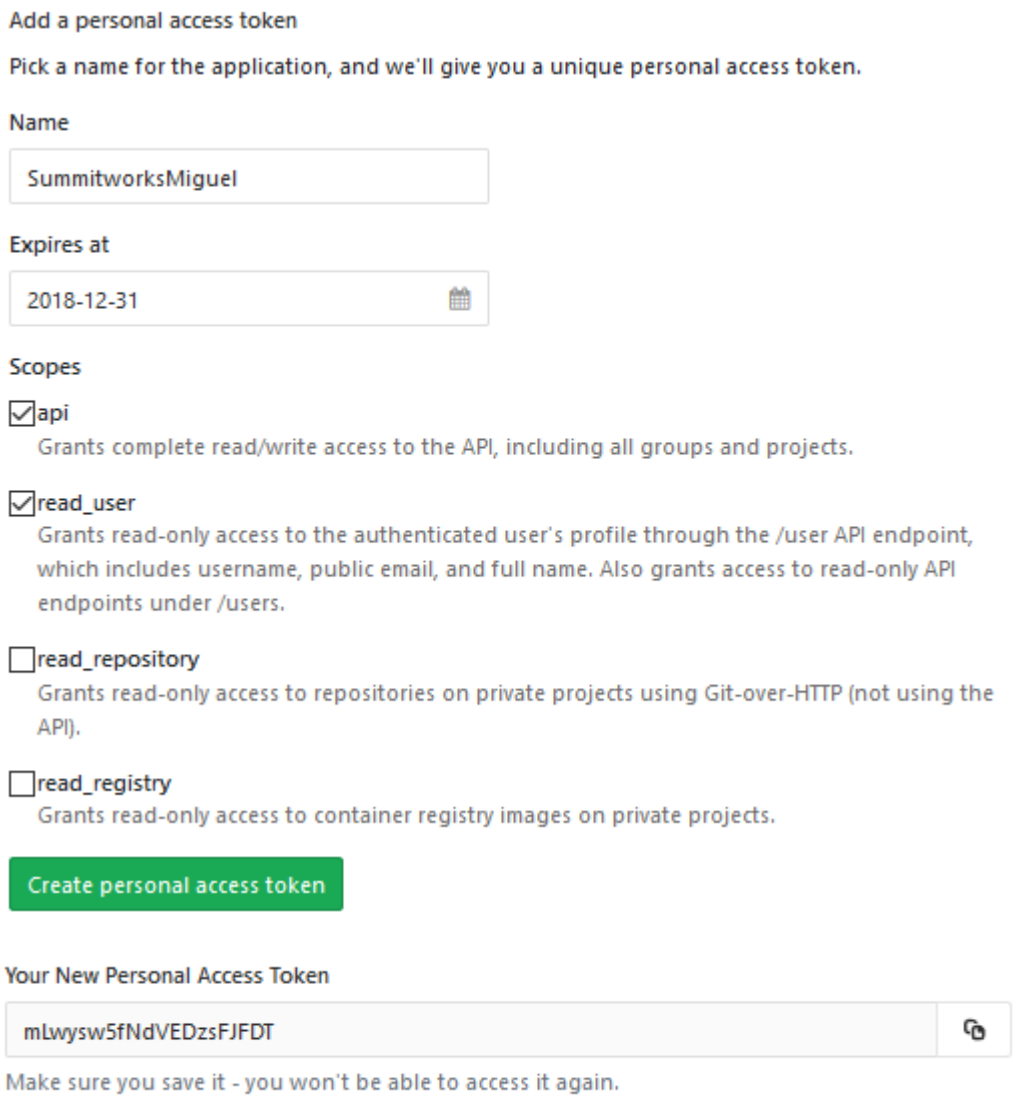
Step 1: Create your Personal Access Token

- Navigate to *https://gitlab.com/profile/personal_access_tokens.*

On “Personal Access Token" form

- Give a name to your token.
- Select and expiry date.
- Select "api" and "read_user" permissions.
- Hit "Create personal access token" button.

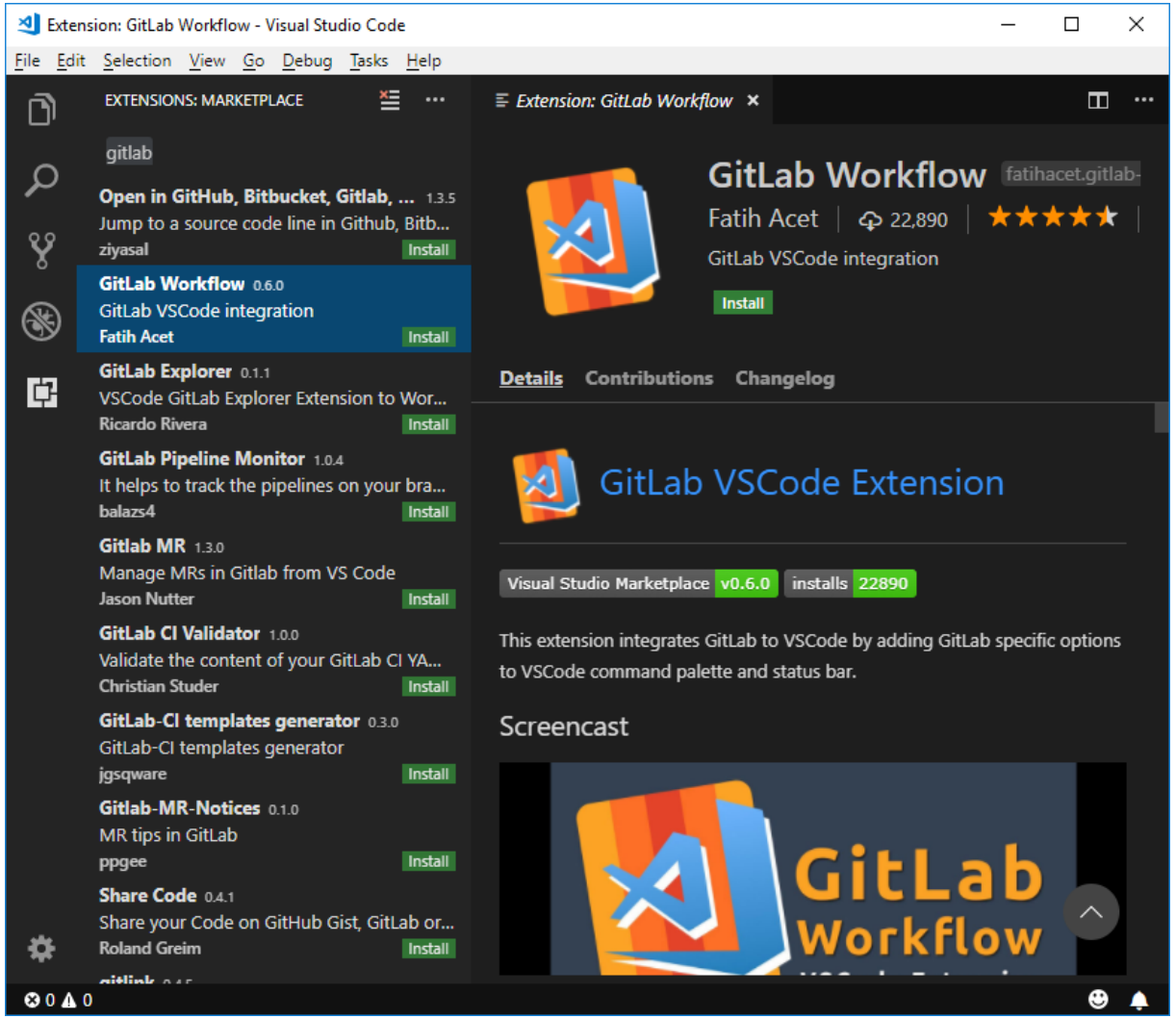
- Copy the token. *Remember you won't be able to see value of this token ever again for security reasons.*



GitLab Visual Studio Code Extension

Step 2: Install GitLab Workflow Extension

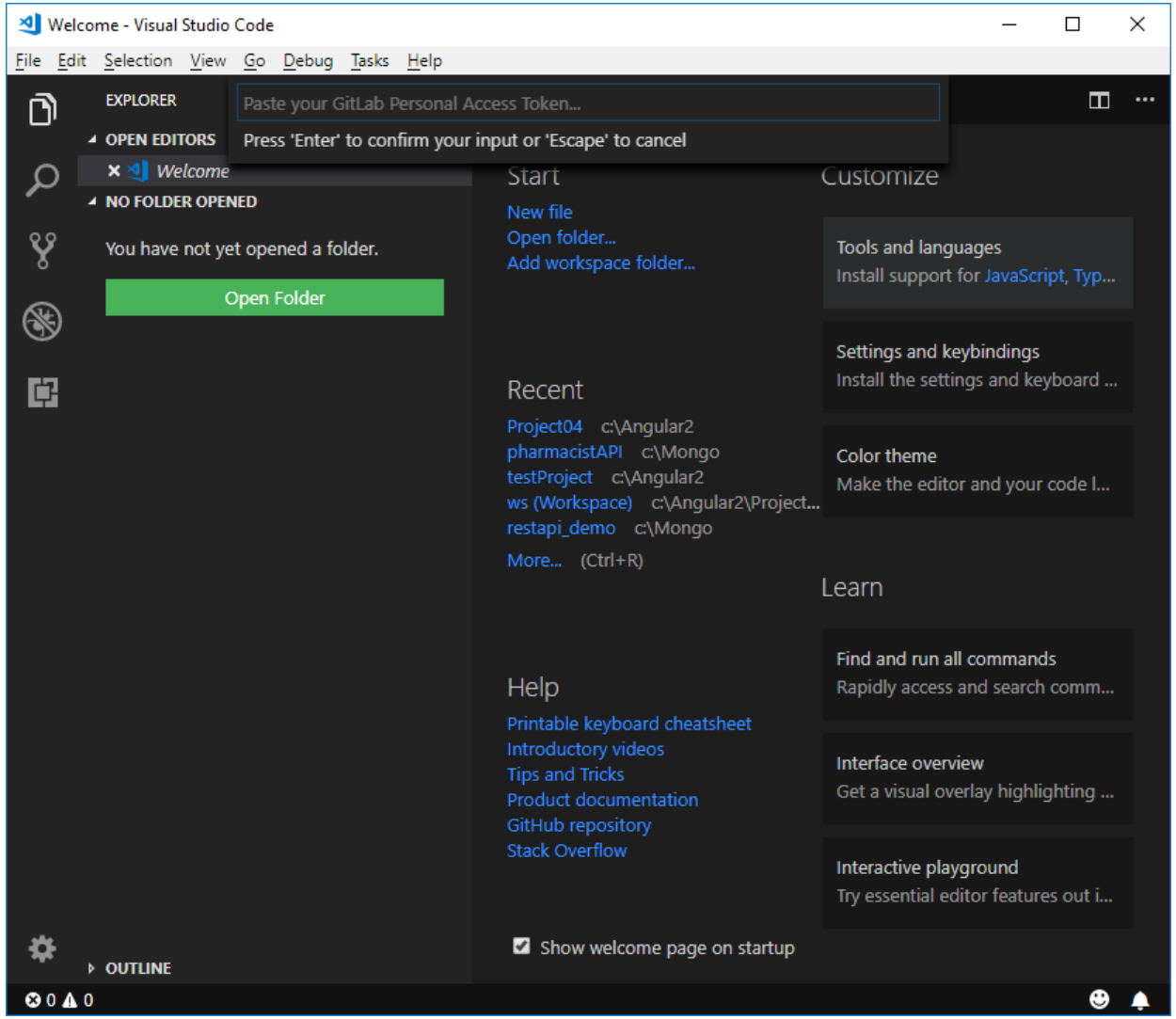
- Open Visual Studio Code
- Open up Extensions by pressing Cmd+ Shift + X
- Search for "GitLab" and hit Enter.
- Select GitLab Workflow and click on “install” button.



GitLab Visual Studio Code Extension

Step 3: Add token to GitLab Workflow Extension

- Open up Command Palette by pressing Cmd + Shift + P
- Search for "GitLab: Set GitLab Personal Access Token" and hit Enter.
- Enter the URL to the Gitlab instance the PAT should apply to and hit Enter.
- Extension will ask for your PAT. Paste your PAT and hit Enter. *It won't be visible and accessible to others.*

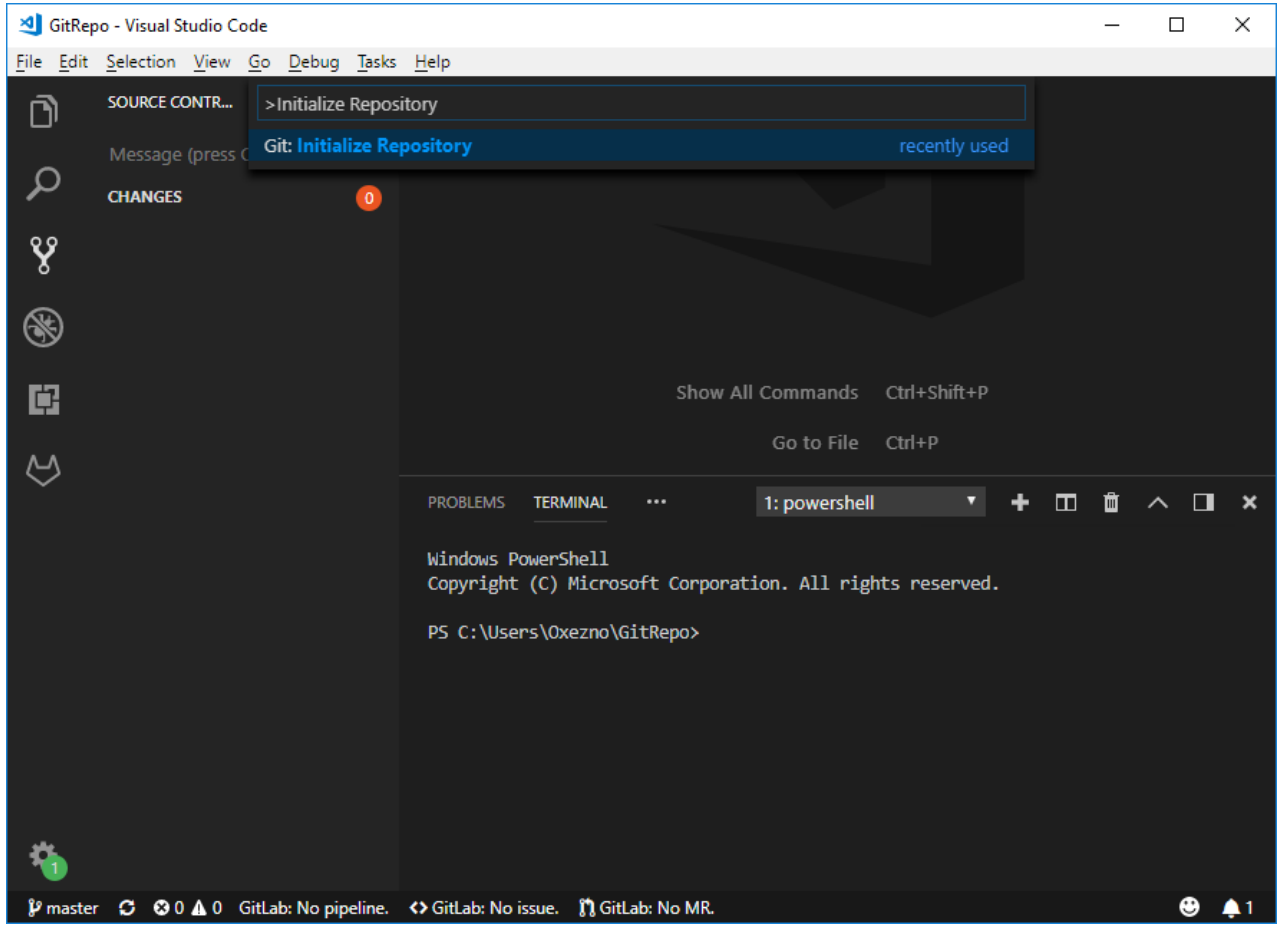


Create Local Git repository

Create Local Git repository

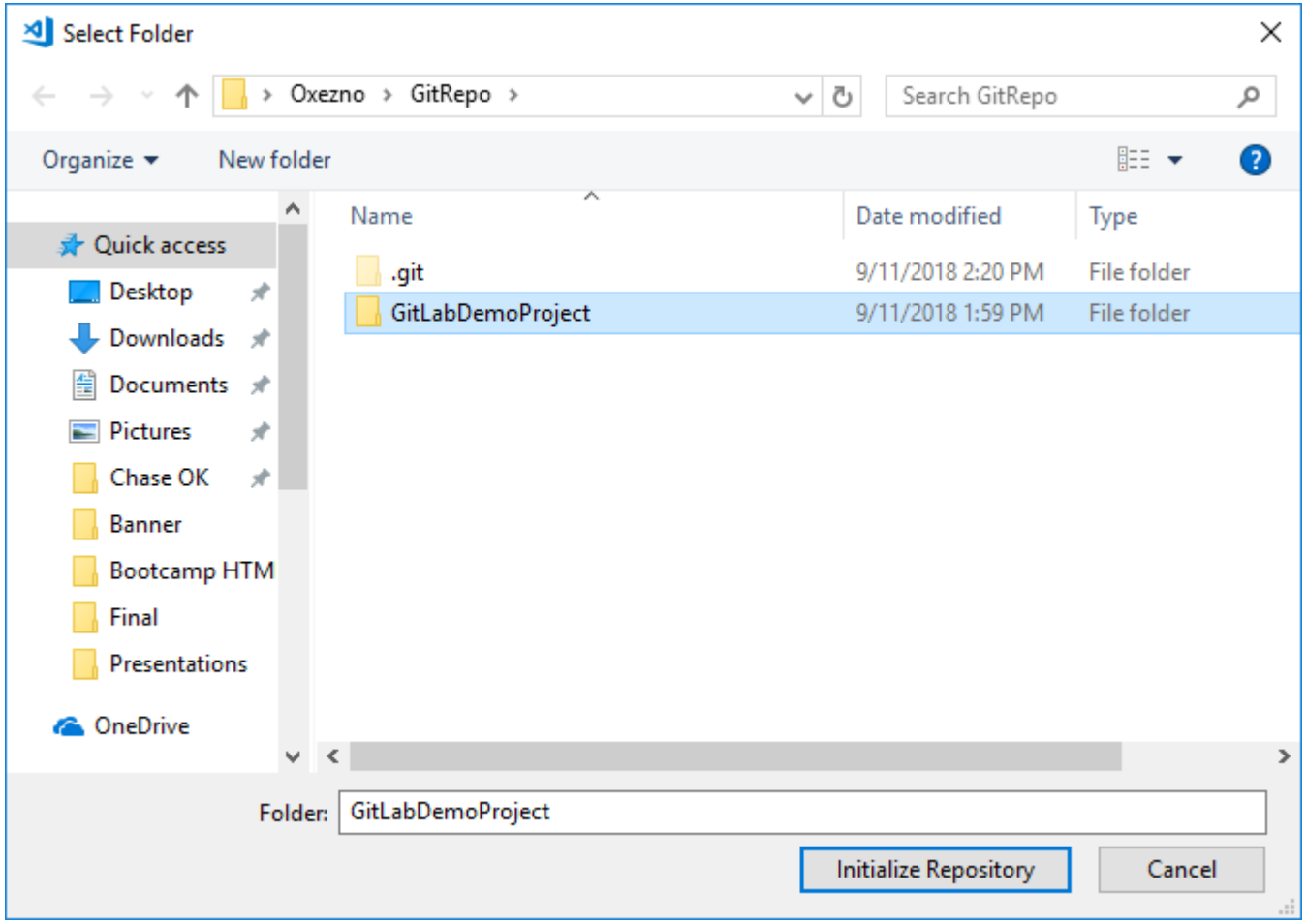
- Open up Command Palette by pressing Cmd+ Shift + P

- Search for “Initialize Repository" and hit Enter.



Create Local Git repository

- Create a new folder for Local Git Repository
- Select the folder and Click on “Initialize Repository”.



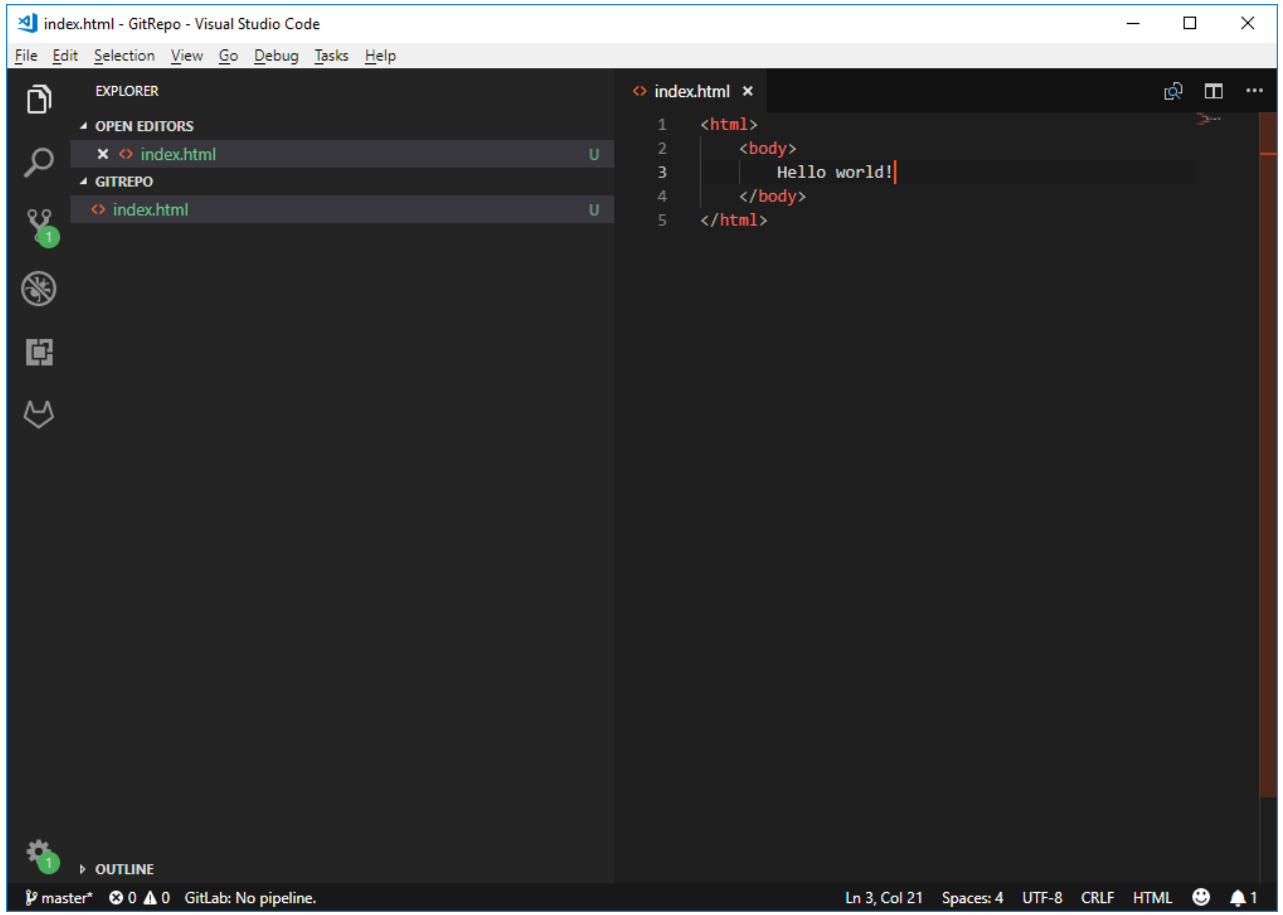
Add new files to the repository

Add new files to the repository

Lets create a HTML file.

- Click on Explorer (Ctrl + Shift + E).
- Create a new file into your Repository folder with the name index.html.

```
<html>
    <body>
        Hello world!
    </body>
</html>
```



Staging Environment

Staging Environment

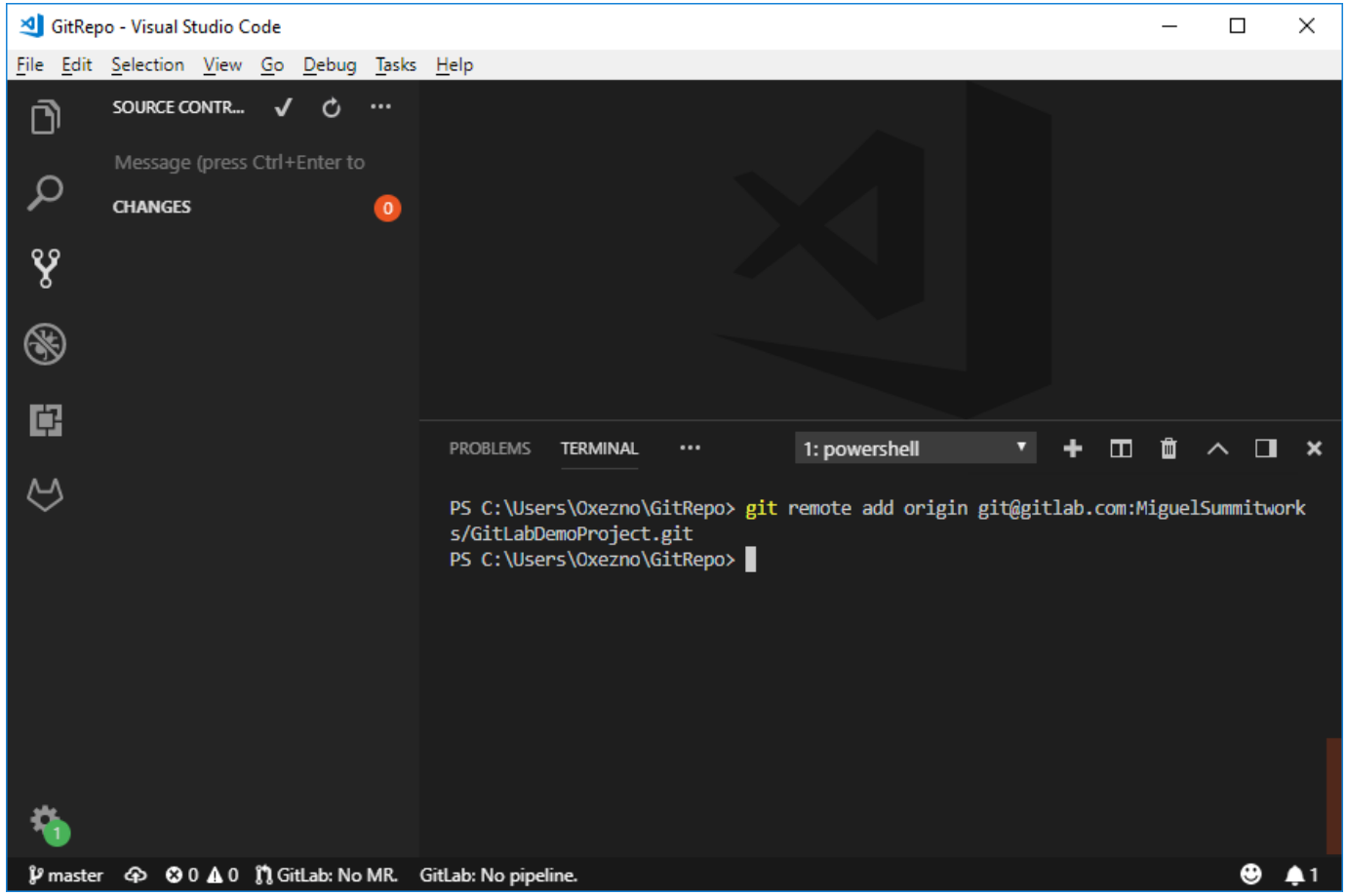
- You need to have a Git origin set up. You can get the required URL from the repo host. Once you have that URL, you need to add it to the Git settings by running a couple of command line actions.

```
cd existing_folder
git init
git remote add origin git@gitlab.com:MiguelSummitworks/GitLabDemoProject.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Staging Environment

- Open up Command Palette by pressing Cmd+ Shift + `

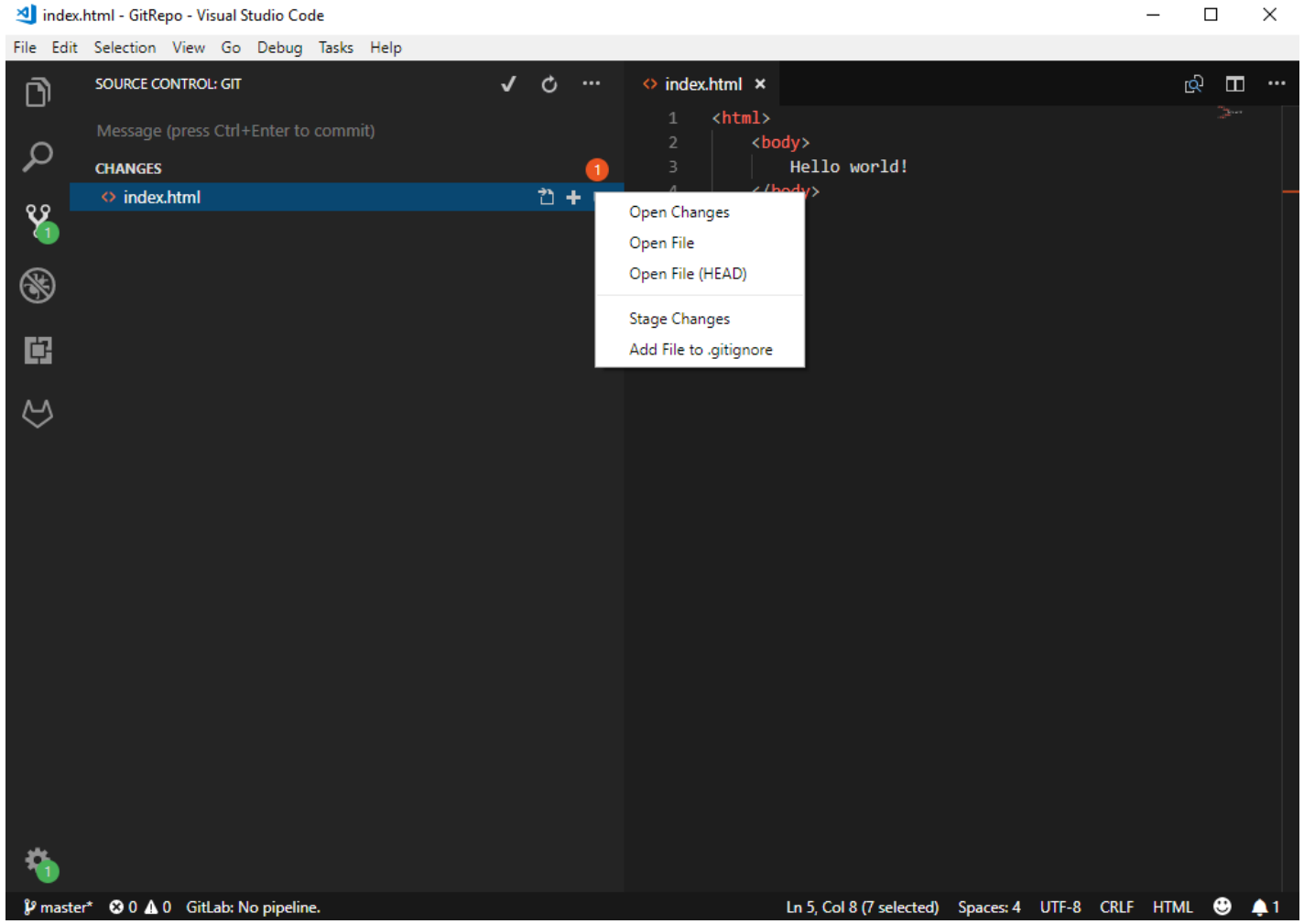
- Paste the URL from the repo host.



Add files to the Staging Environment

Add files to the Staging Environment

- Click on Source Control (Ctrl + Shift + G).
- Right click on your index.html file.
- Click on stage changes

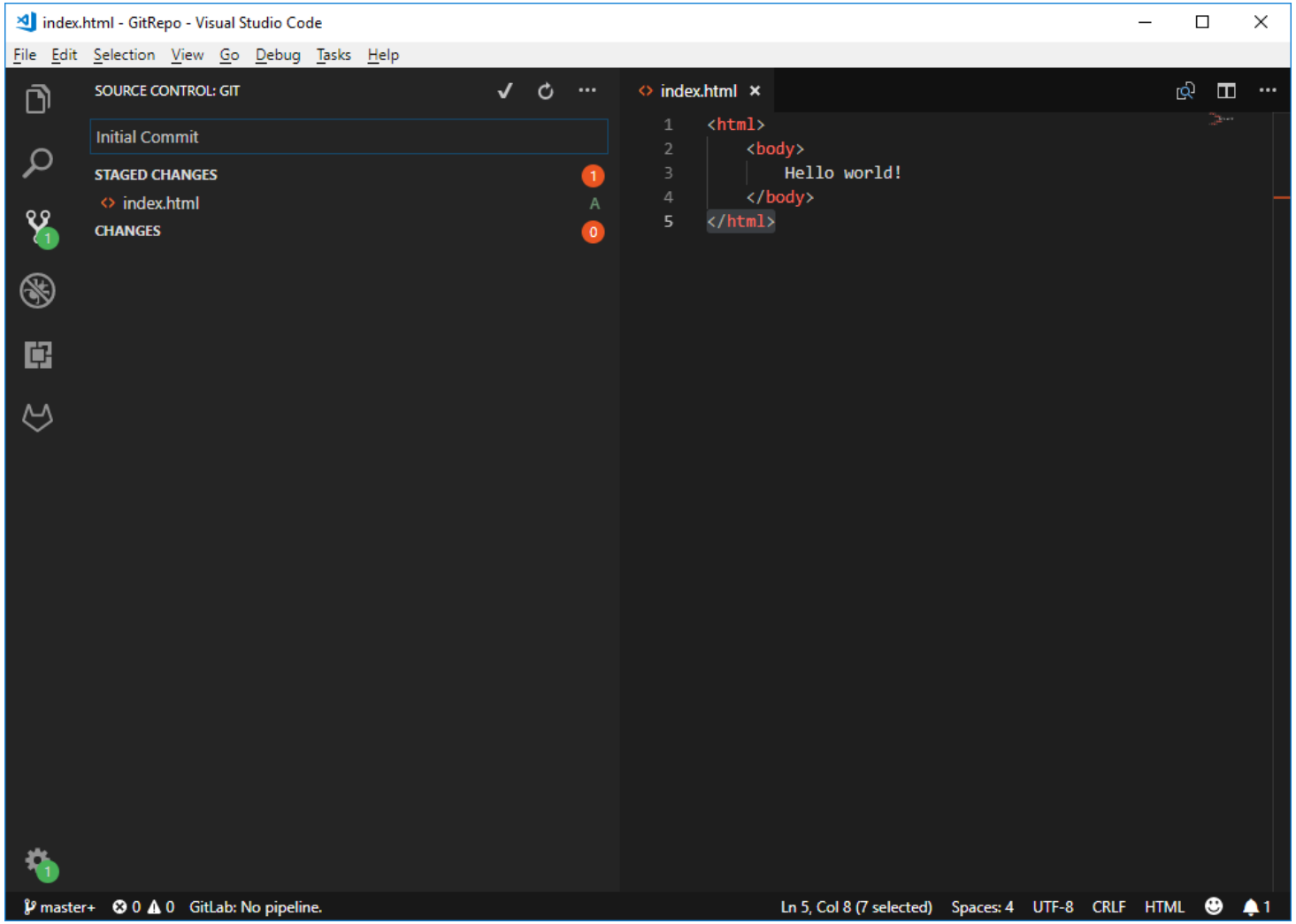


Create a commit and push

Create a commit

Now lets start with the Commit.

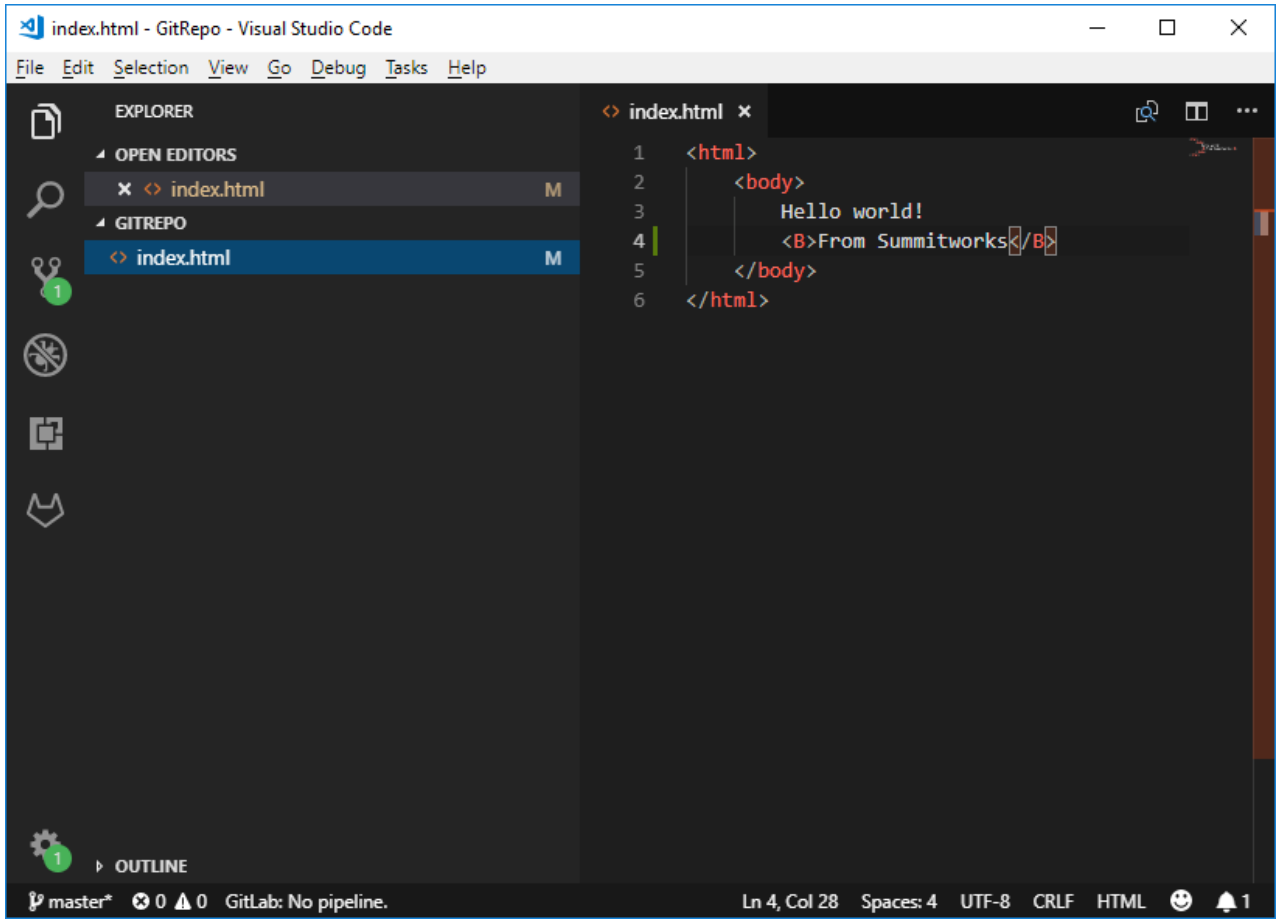
- You need to add a commit message.
- Click on the Commit check icon.



Create a commit

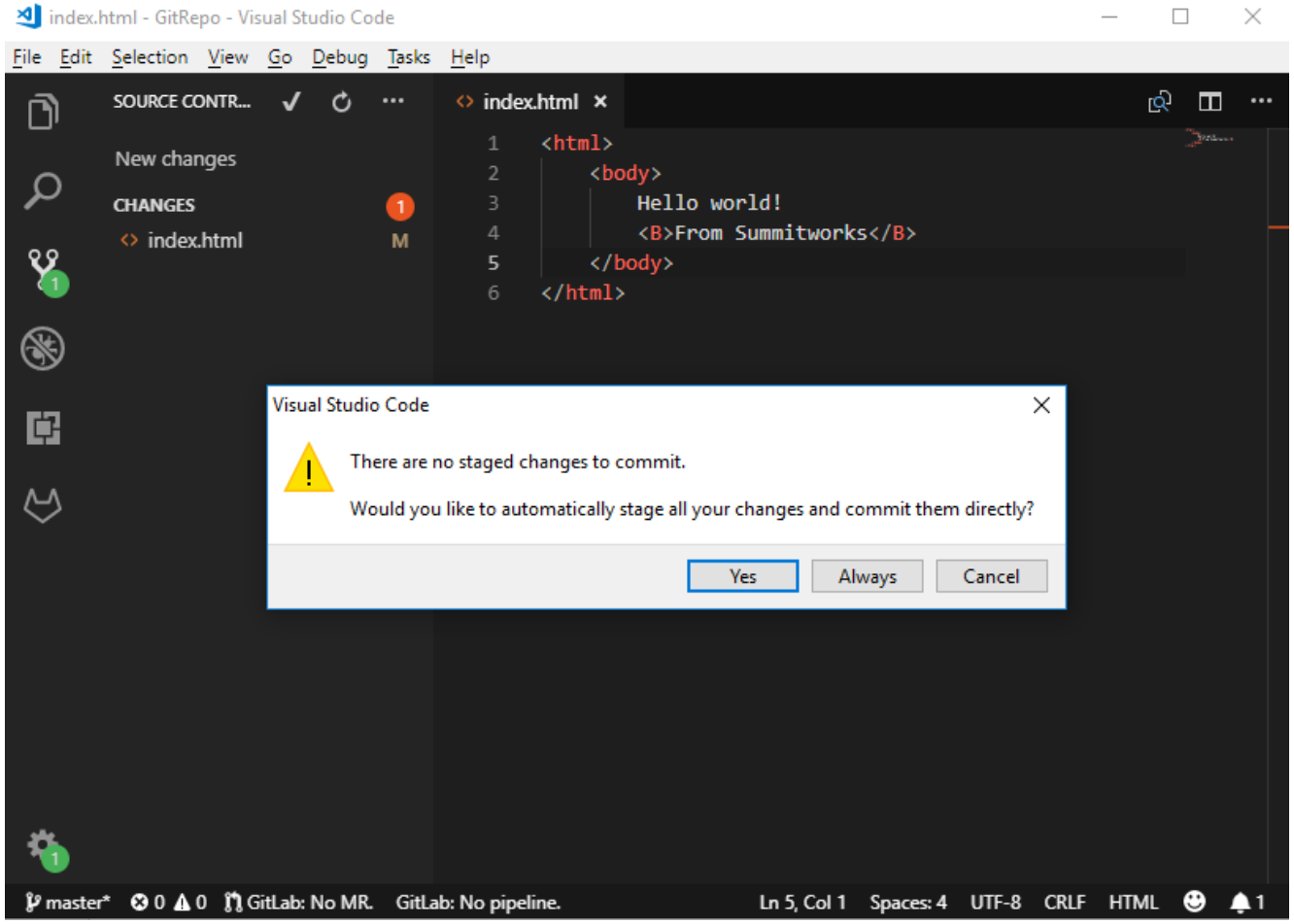
- Lets make some changes to our index.html file

```
<html>
    <body>
        Hello world!
        <B>From Summitworks</B>
    </body>
</html>
```



Create a commit

- Now Commit the changes.
- You can choose to Stage Changes automatically.
- Now click on “Push”

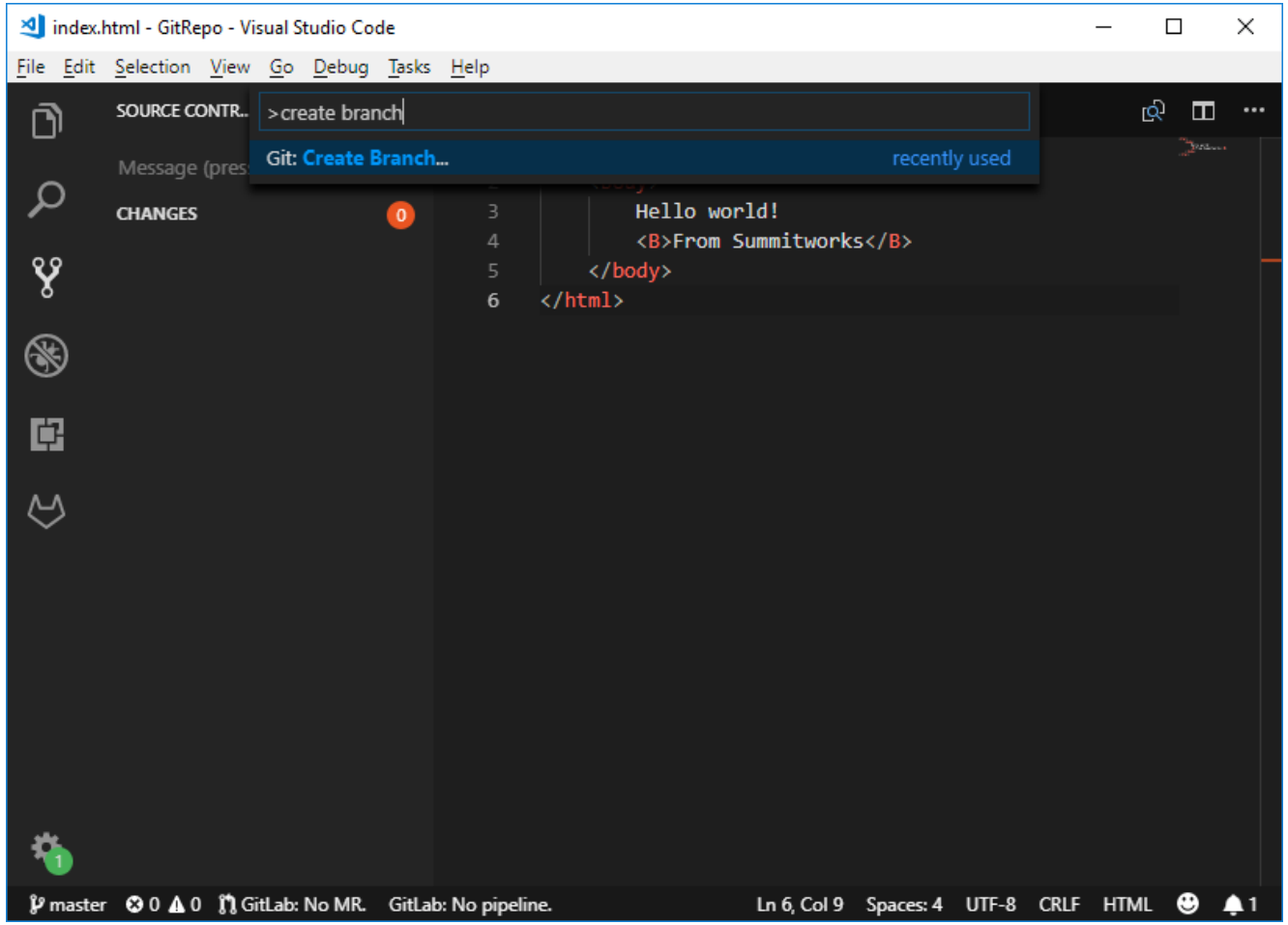


Create a new branch

Create a new branch

Creating a Branch

- Open up Command Palette by pressing Cmd + Shift + P.
- Search for “Create Branch" and hit Enter.
- Provide a branch name and hit enter.
- Now you can switch between branches clicking on the left bottom icon.

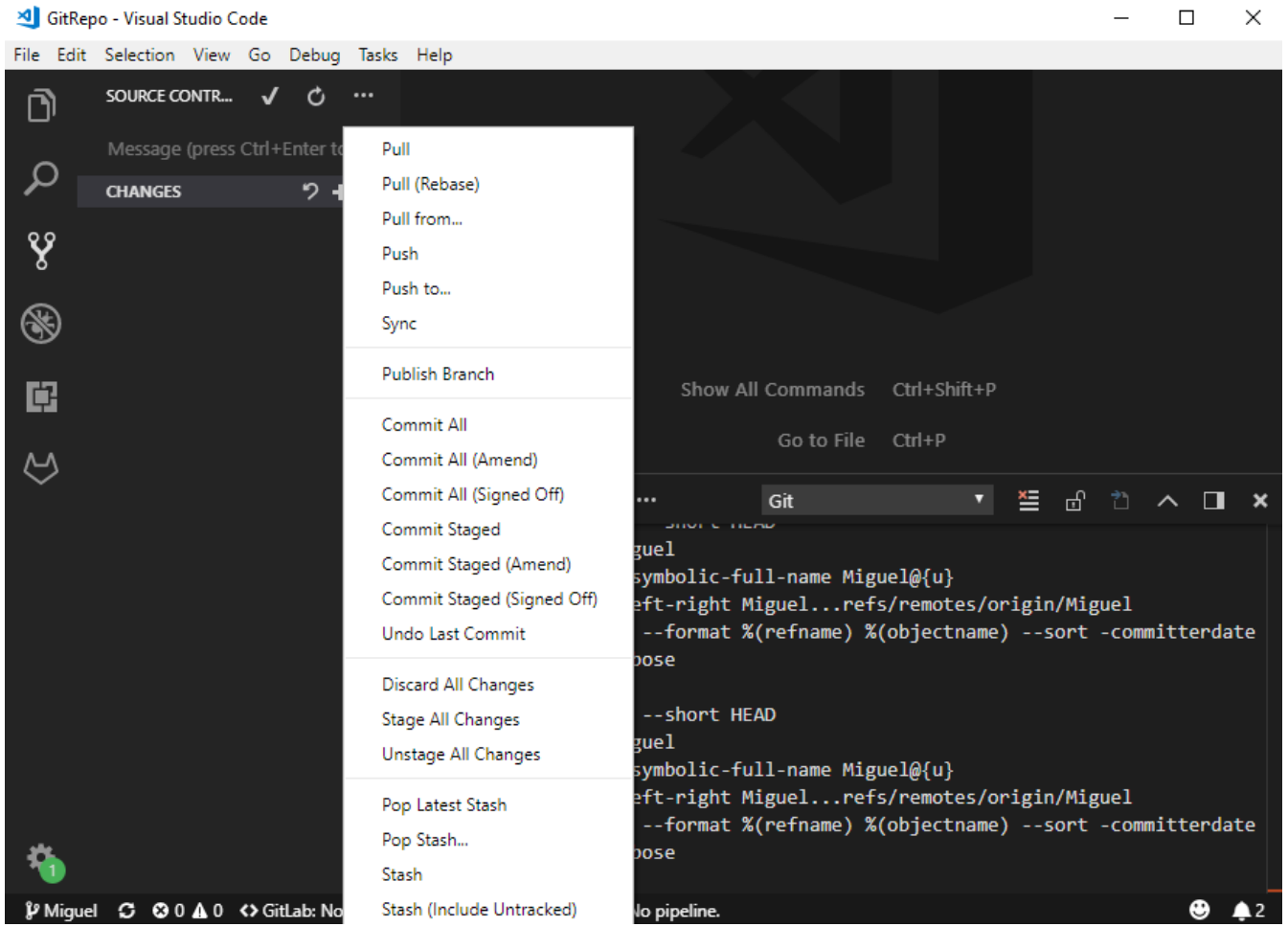


Push a branch to GitLab

Push a branch to GitLab

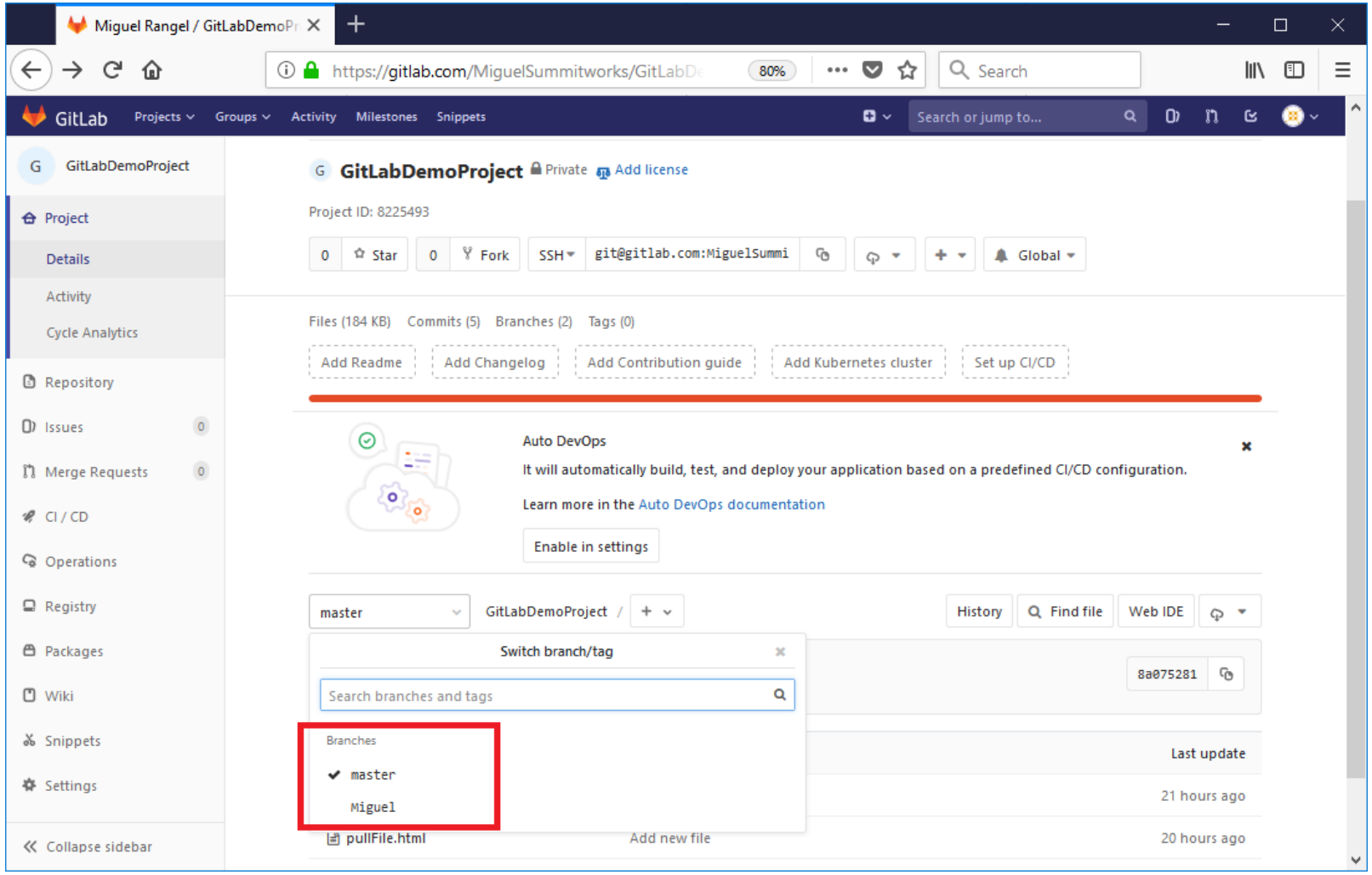
Push a Branch

- Open “More Actions” icon.
- Click on “Publish Branch” and then click on “Push”.



Push a branch to GitLab

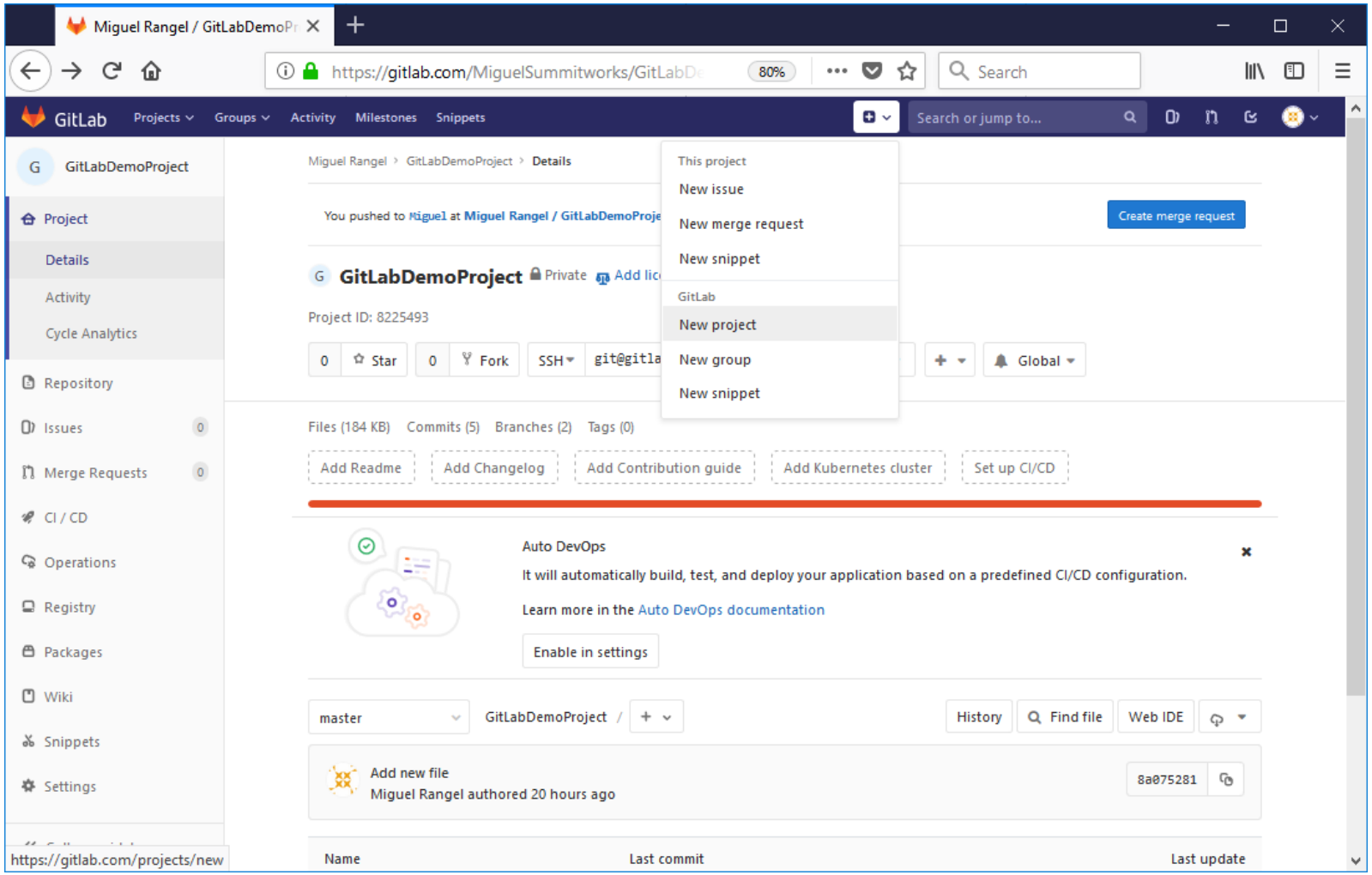
- Navigate to your GitLab account and check if the branch was pushed.



Clone a Git Repository

Clone a Git Repository

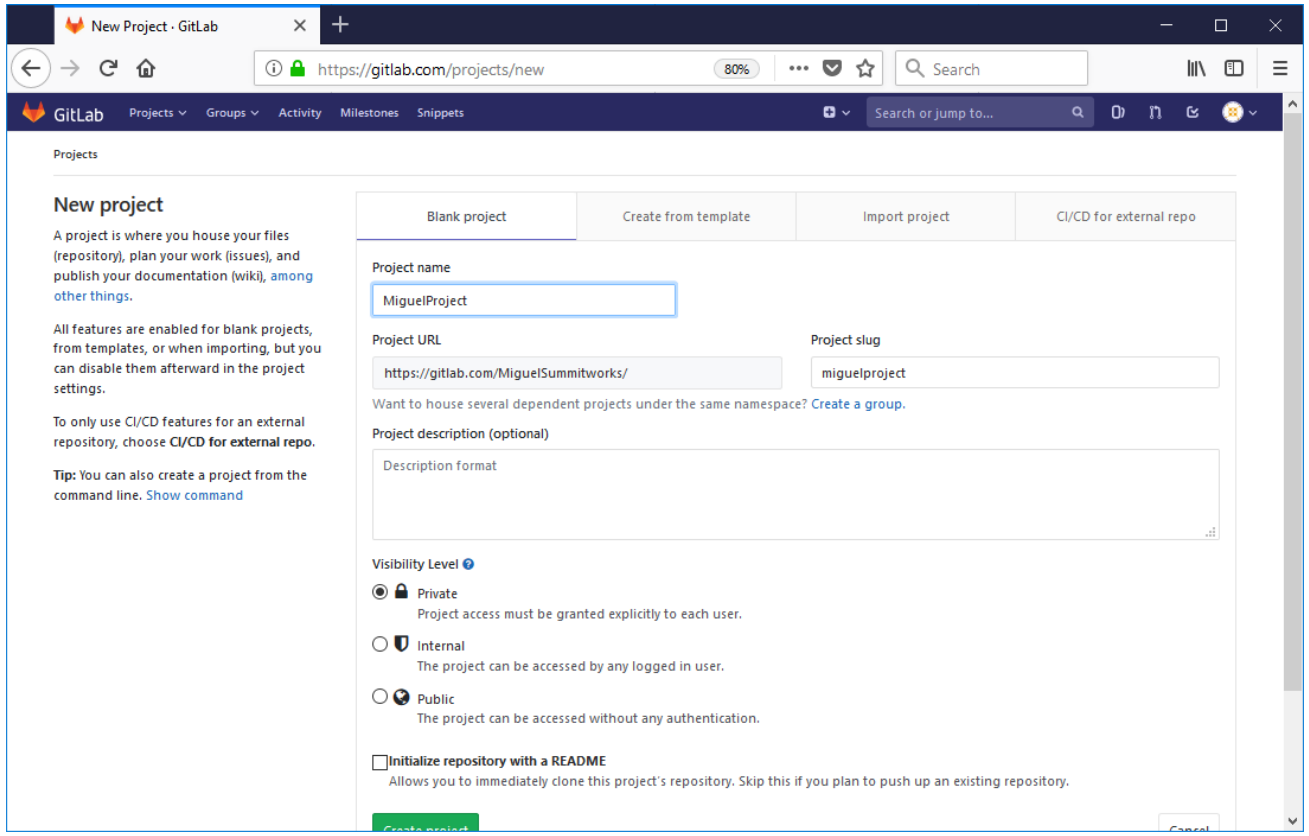
- Navigate to your GitLab account
- Click on “+” icon on the top.
- Click on “New Project”.



Clone a Git Repository

Create A New GitLab Project

On the "New Project" page, enter a name for the project and set the visibility level. The project in this example is set to "Public", but you can also choose to make your project "Private" or "Internal". Click "Create Project" once done.



Clone a Git Repository

You need to have a Git origin set up. You can get the required URL from the repo host. Once you have that URL, you need to add it to the Git settings by running a couple of command line actions.

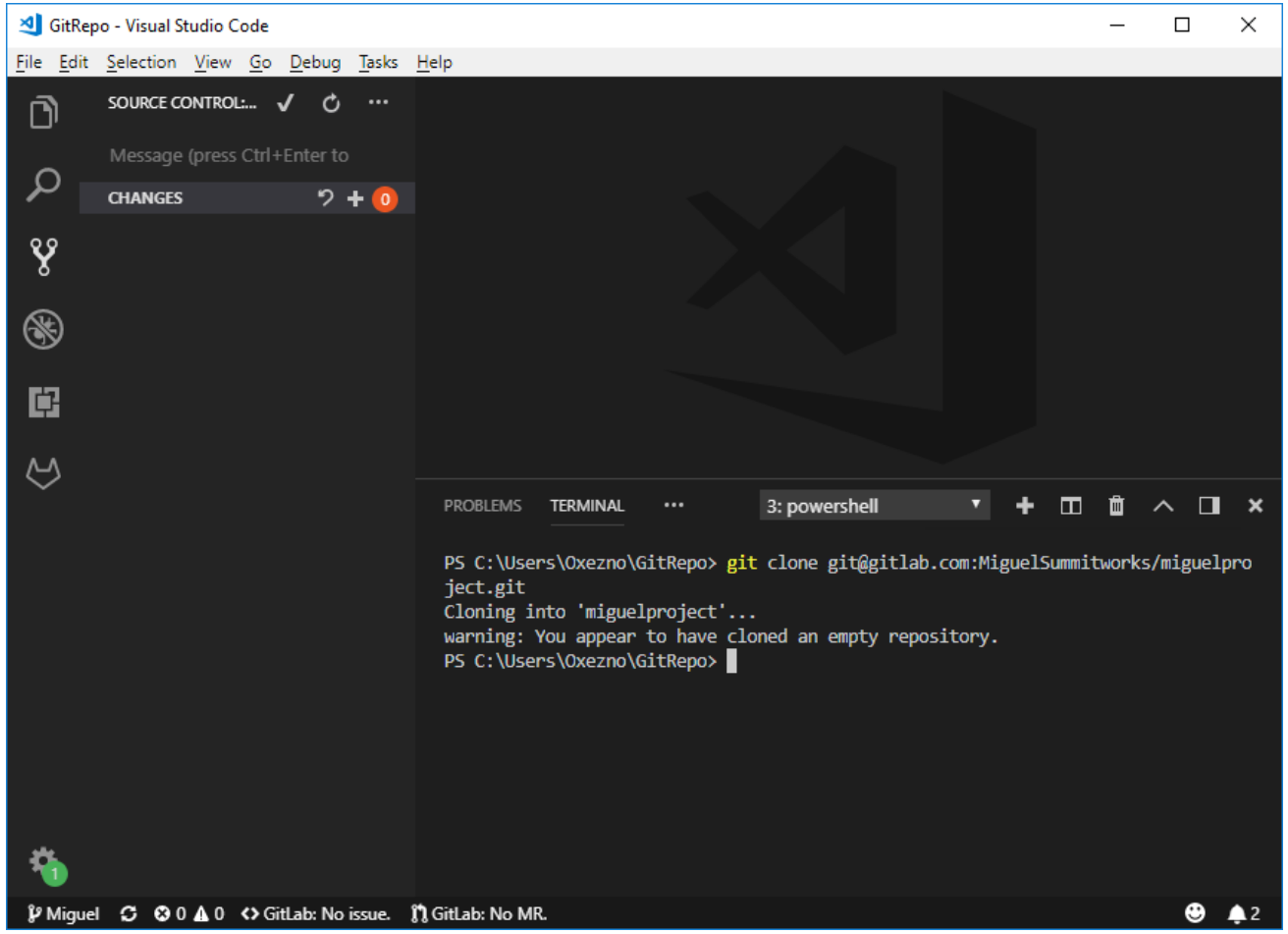
Create a new repository

```
git clone git@gitlab.com:MiguelSummitworks/miguelproject.git
cd miguelproject
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Clone a Git Repository

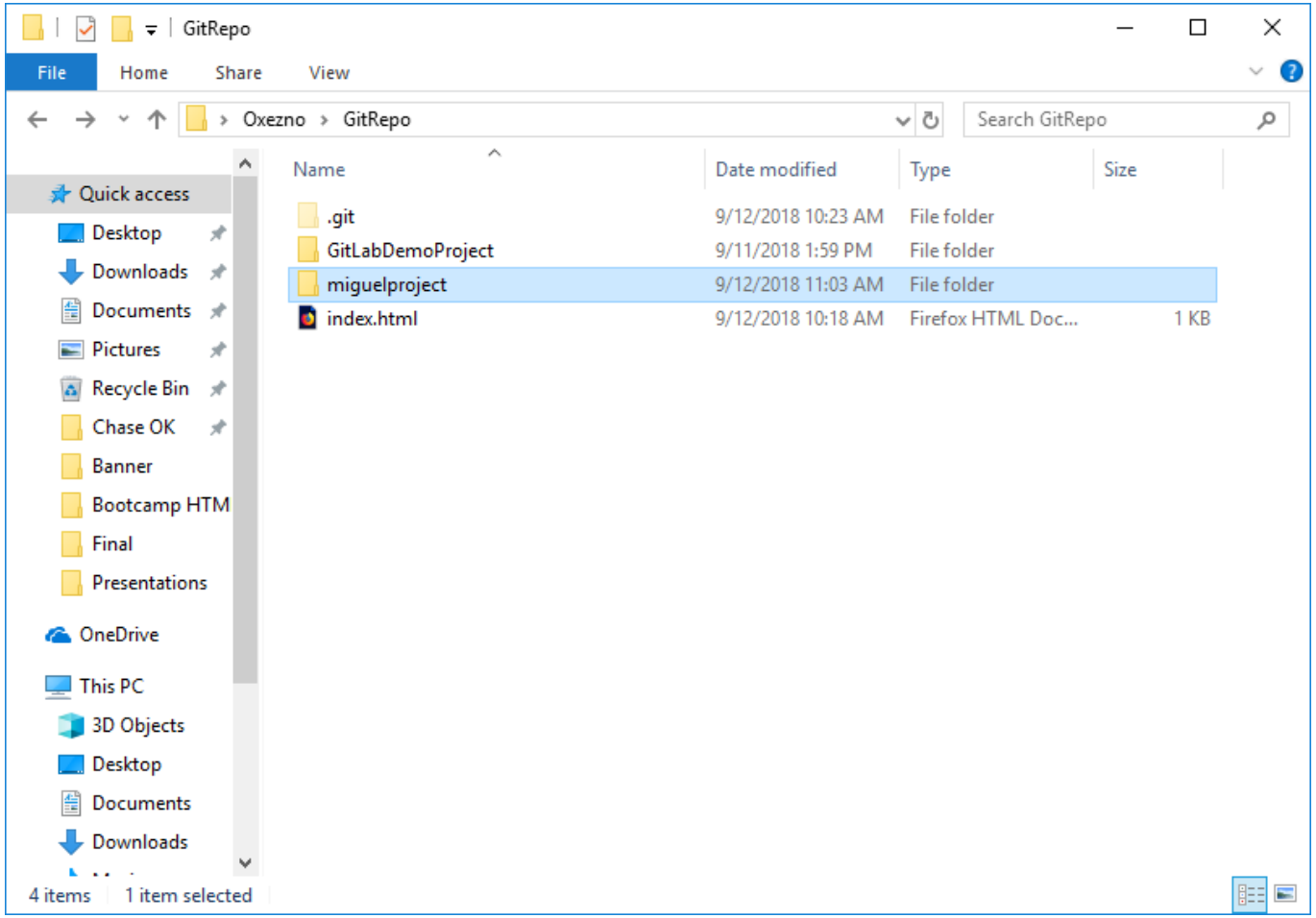
Open up Command Palette by pressing Cmd + Shift + `

Paste the URL from the repo host.



Clone a Git Repository

- Navigate to your local GitLab repository and check if the project was cloned.

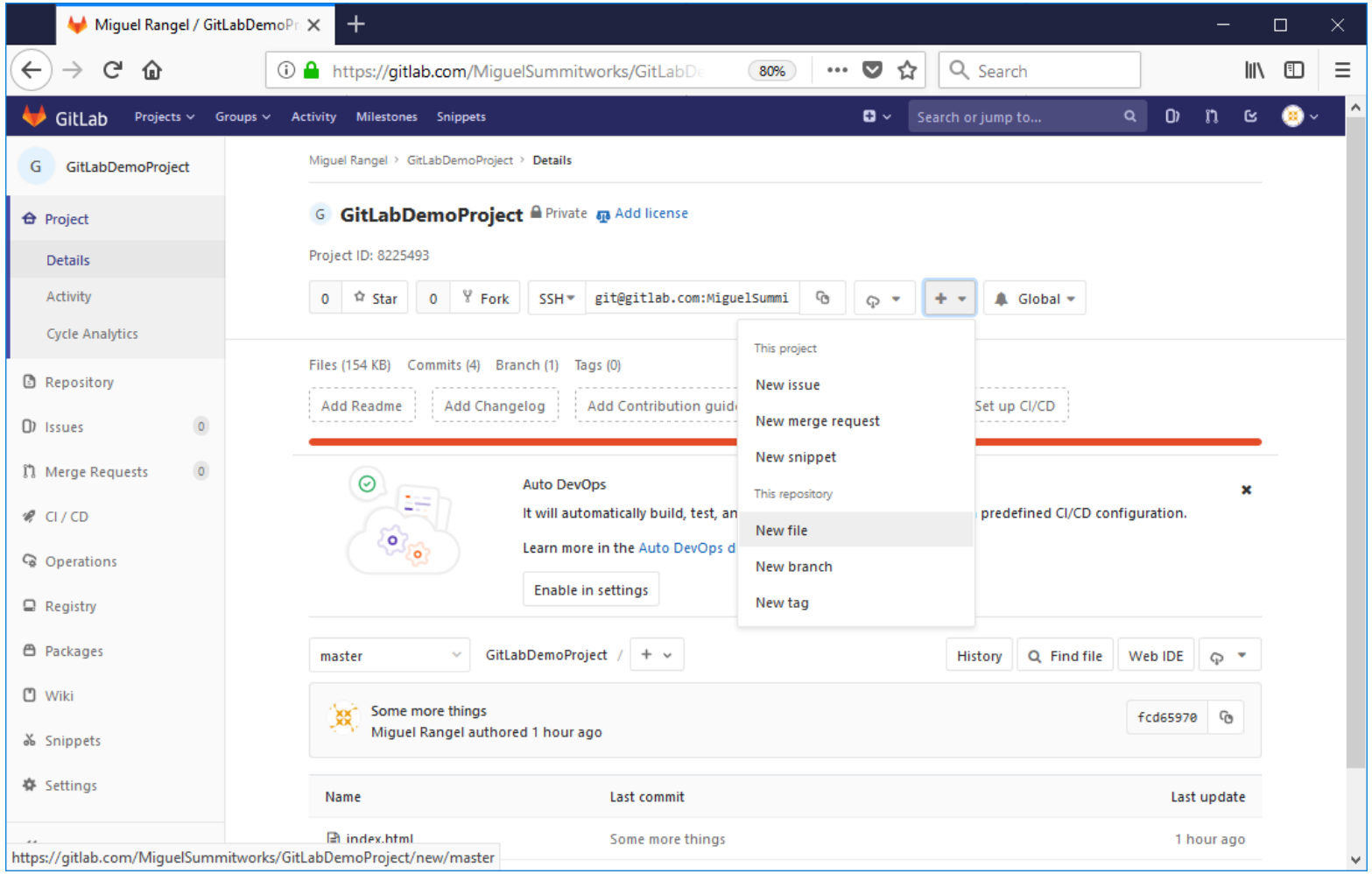


Create a Pull Request (PR)

Create a Pull Request (PR)

Pull from GitLab

- Navigate to your GitLab account
- Click on “Create New” button.
- Select “New File”.

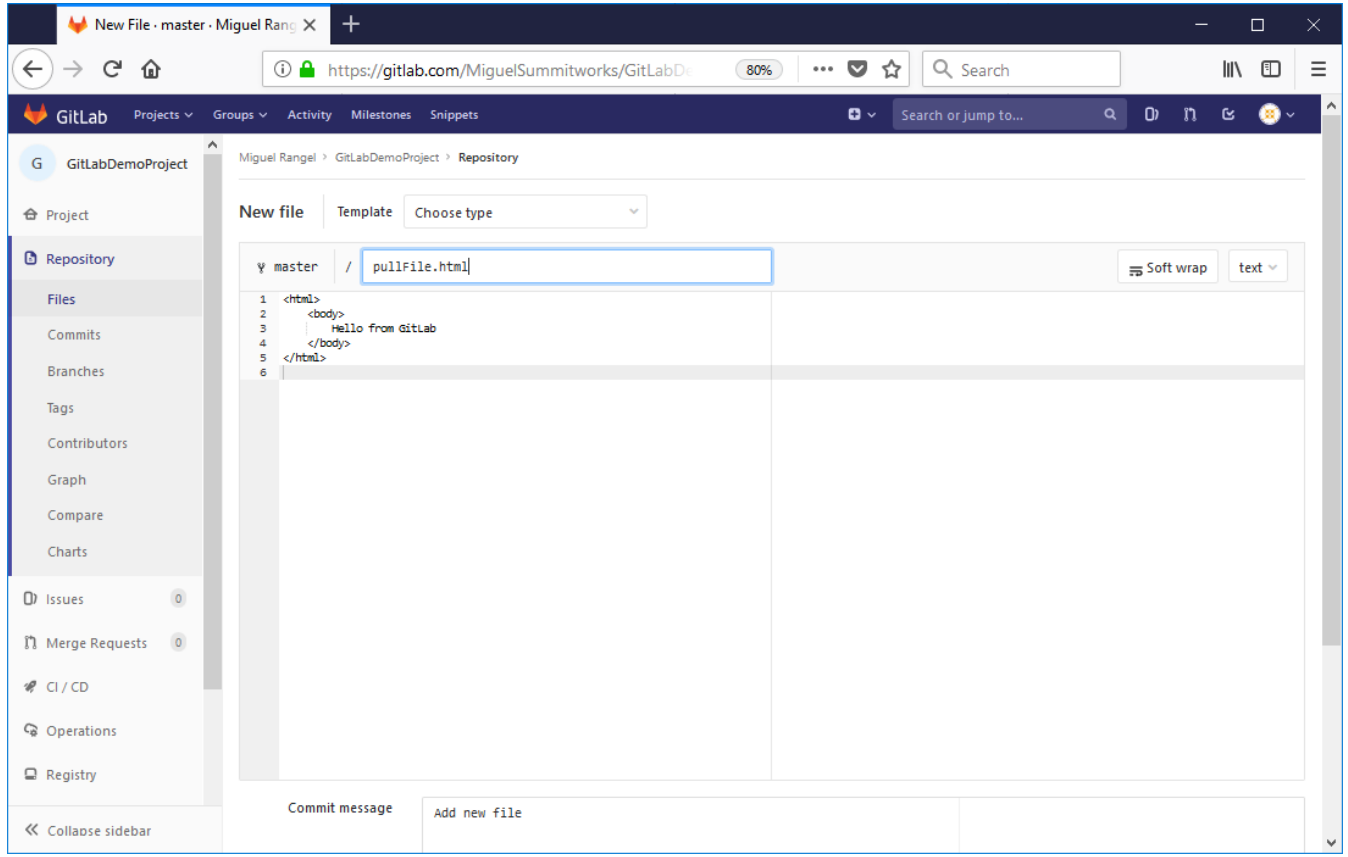


Create a Pull Request (PR)

Pull from GitLab

- Create a new file with the name pullFile.html.

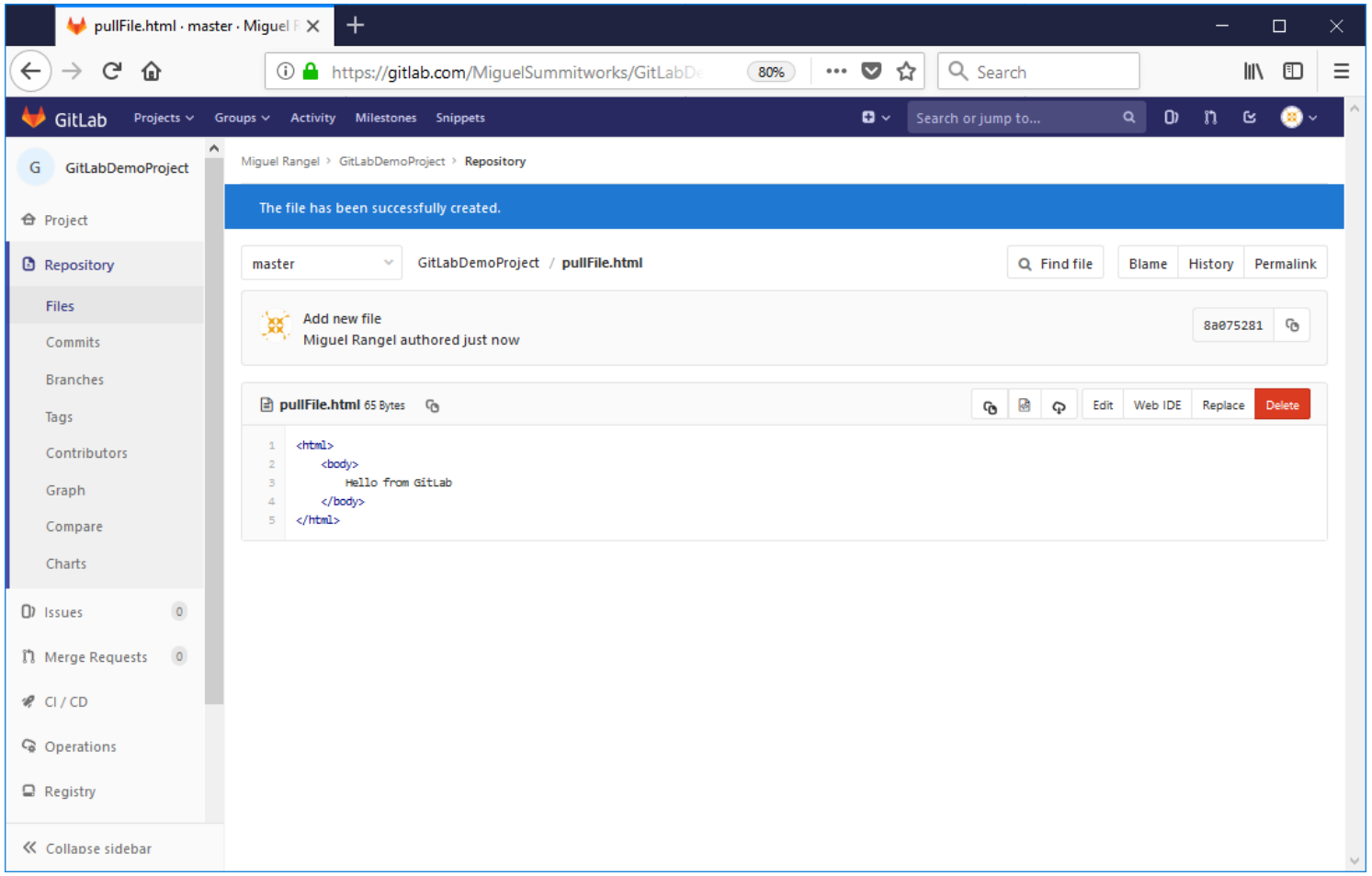
```
<html>
    <body>
        Hello from GitLab!
    </body>
</html>
```



Create a Pull Request (PR)

Pull from GitLab

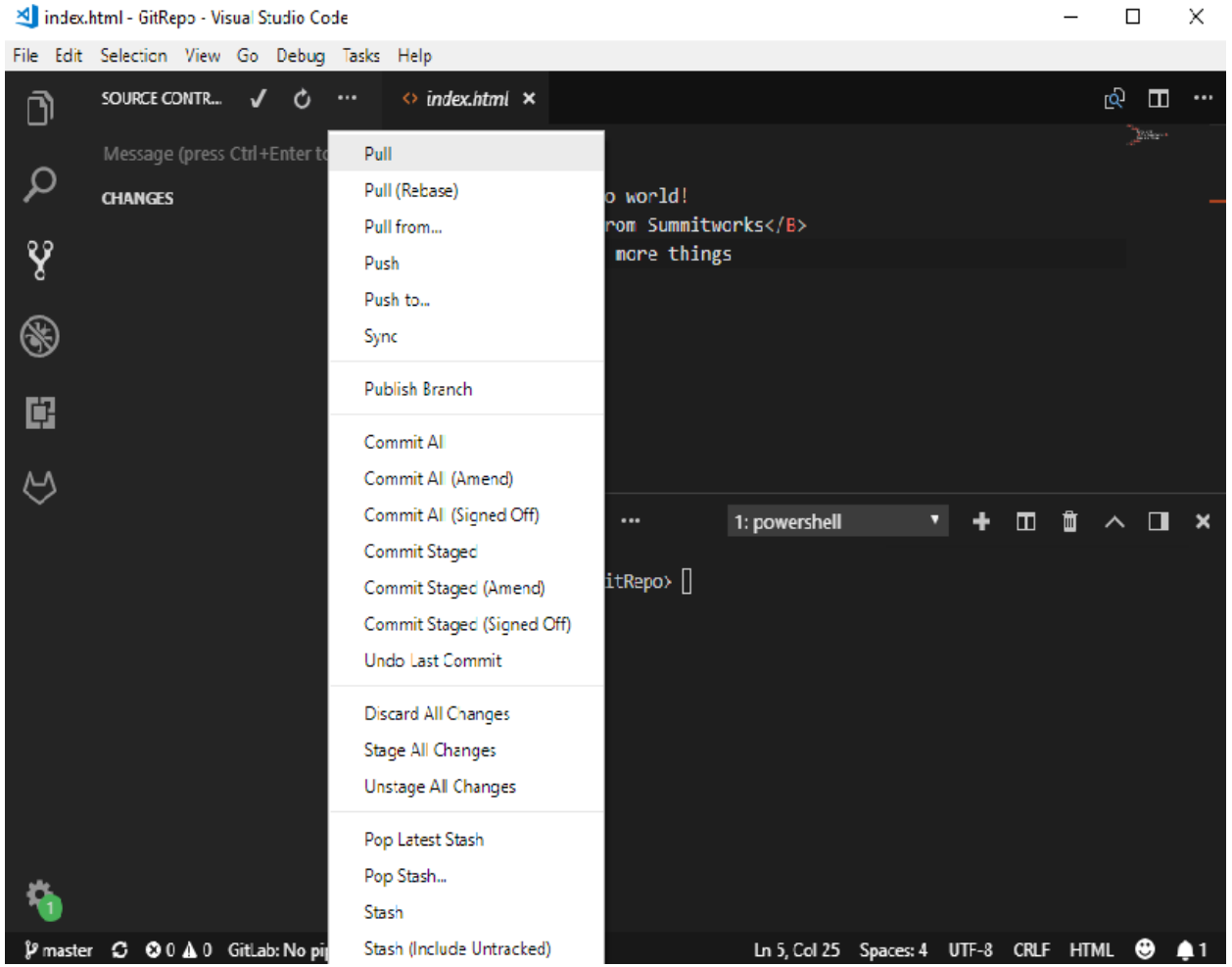
- Once you are done, you will see the file details.



Create a Pull Request (PR)

Pull from GitLab

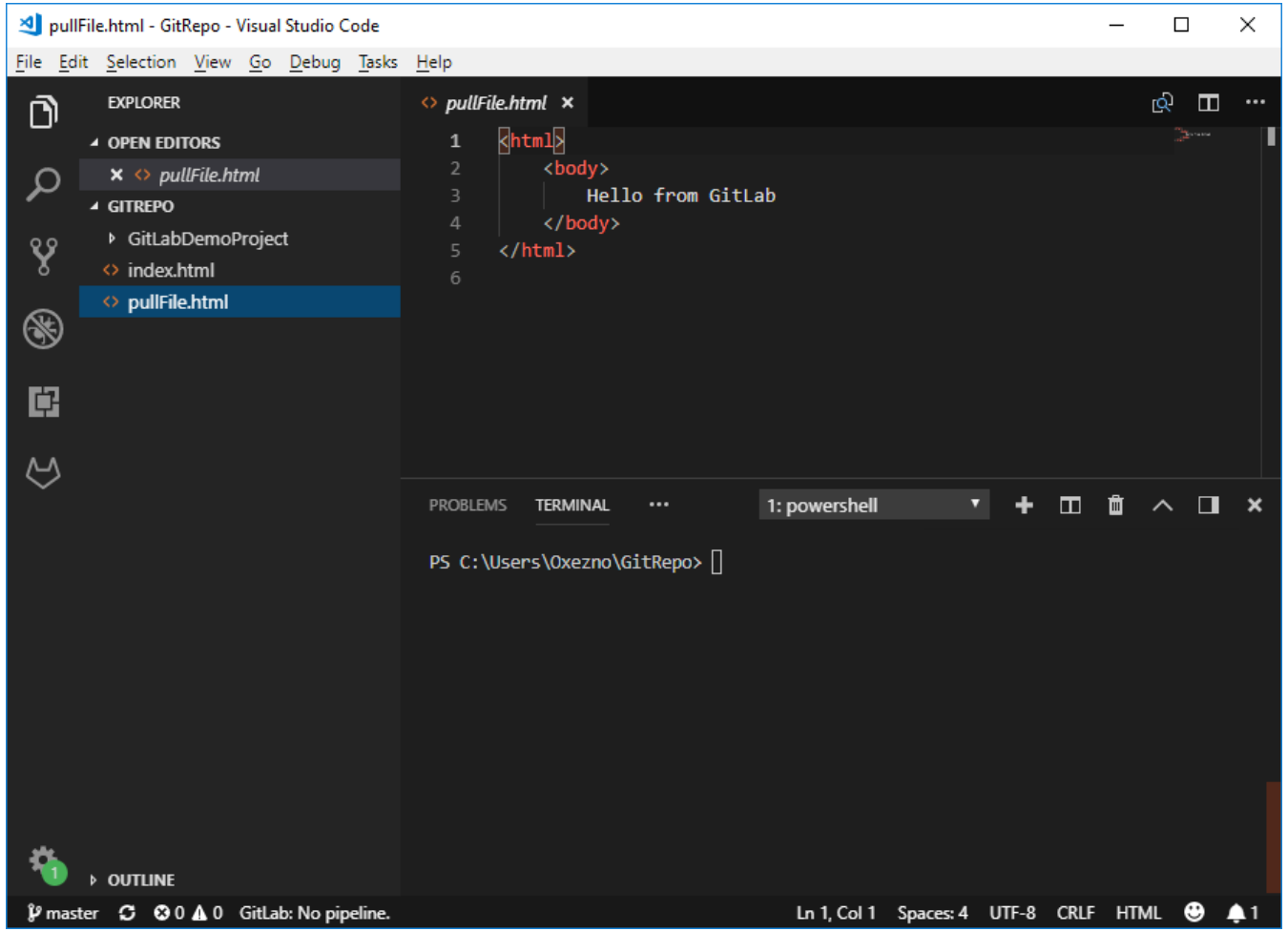
- Go back to VS Code.
- Click on Source Control (Ctrl + Shift + G).
- Click on “More Actions” icon.
- Click on “Pull”.



Create a Pull Request (PR)

Pull from GitLab

- Click on Explorer (Ctrl + Shift + E).
- You can find your “pullFile.html” on your local repository.



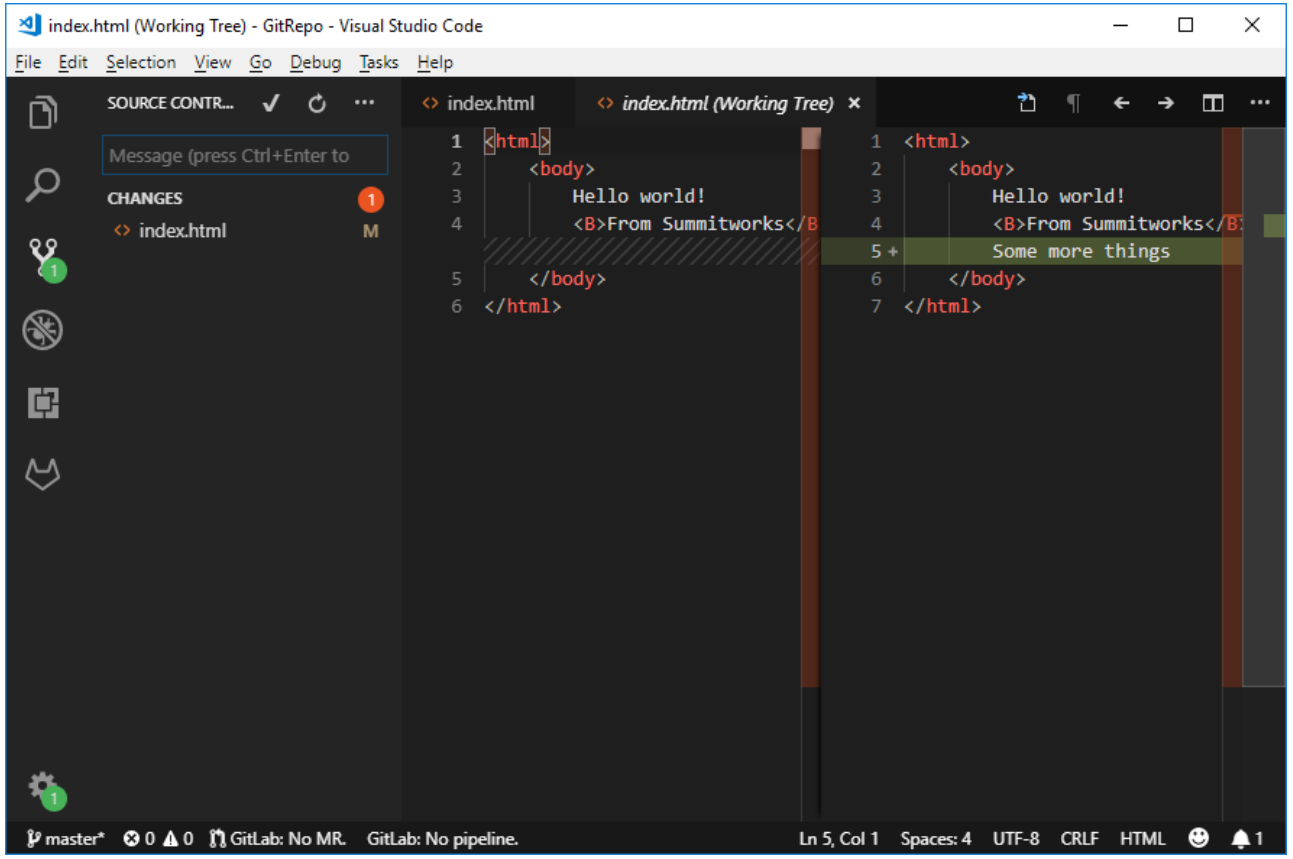
Merge a Pull Request

Merge a Pull Request

Merge conflicts

- Click on Source Control (Ctrl + Shift + G).
- Double click on your index.html file.

Merge conflicts are recognized by VS Code. Differences are highlighted and there are inline actions to accept either or both changes. Once the conflicts are resolved, stage the conflicting file so you can commit those changes.



Exercise

1. Set GitLab Personal Access Token.
2. Create a new Branch (UserBranch)
3. Create a HTML file (User.html)

```
<h1>
    UserName
<h1>
```

4. Push your html file to GitLab repo.
5. Pull from GitLab repo.