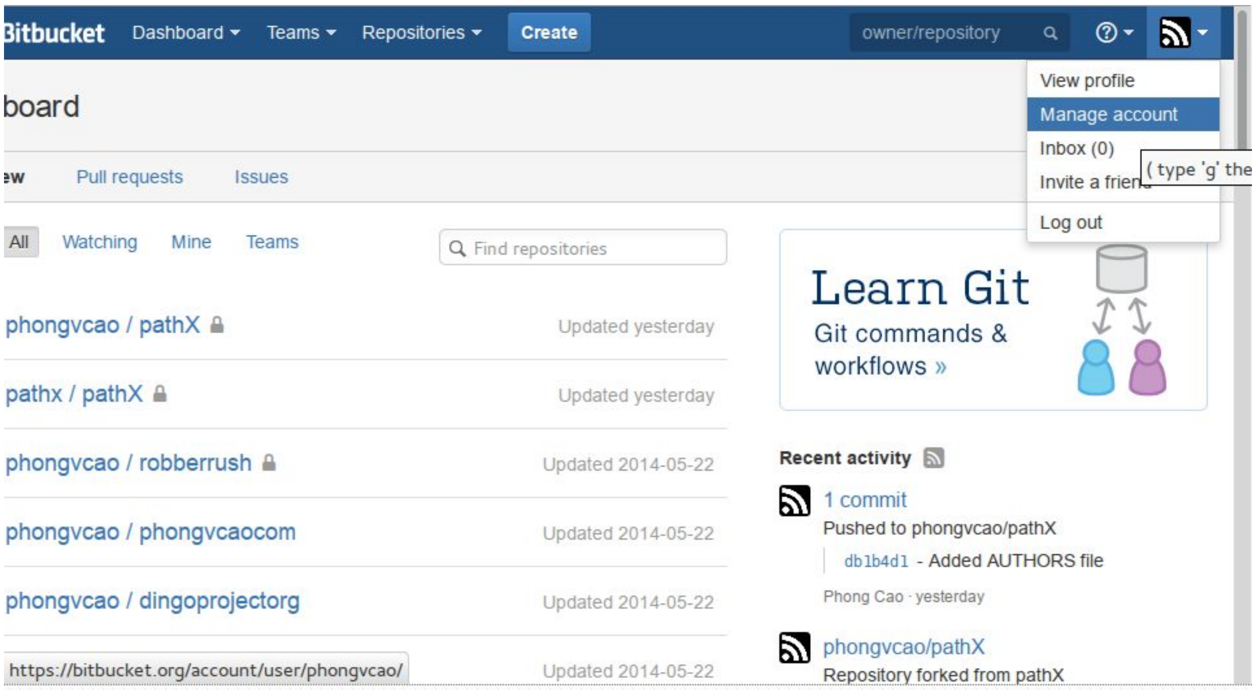
How to collaborate on Bitbucket.org using Git

Contents at a Glance

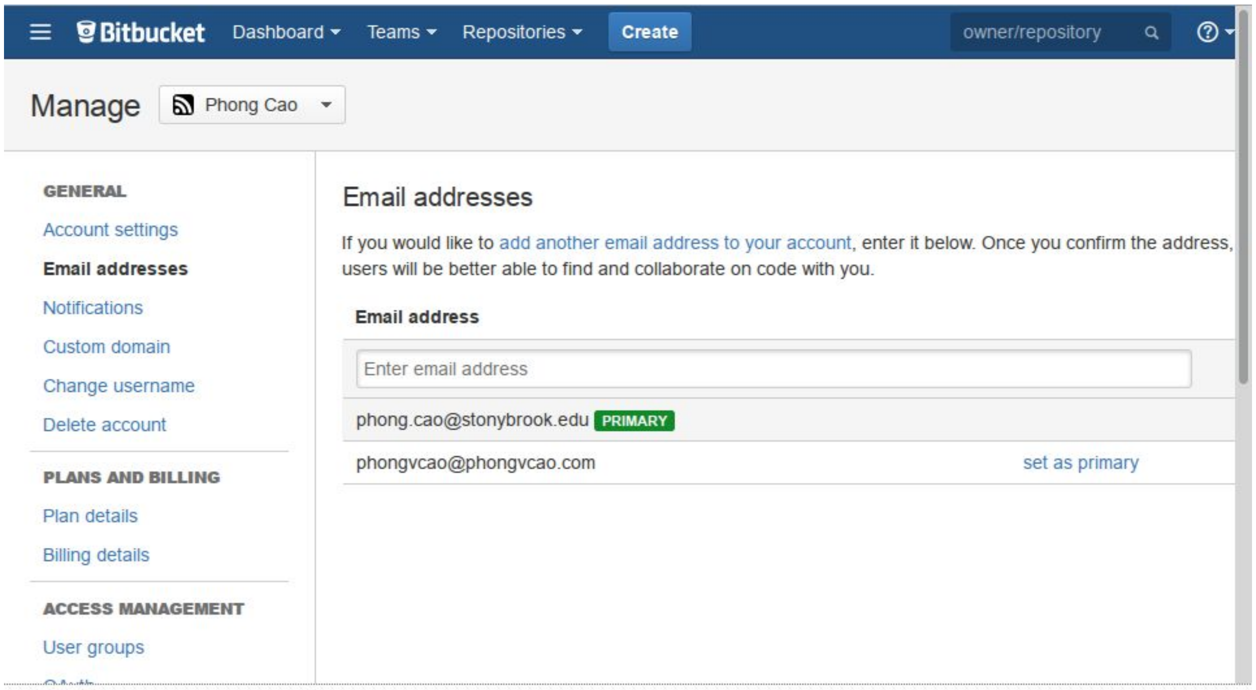
Step 1: Create a Bitbucket “Academic” Account:

1. If you haven’t had a Bitbucket account yet please go to https://bitbucket.org/ and register for one with your Stony Brook Email address (with the *.edu suffix)

2. If you have already had a Bitbucket account, please Log-in into Bitbucket at https://bitbucket.org/, then go to Manage Account -> Email addresses and check if the primary email address of your account is set to your Stony Brook email.



If not, set it as primary to get a *free* student account which allows you to bypass the "5-people per private repository" policy of a typical Bitbucket free account.



Email addresses

If you would like to add another email address to your account, enter it below. Once you confirm the address, users will be better able to find and collaborate on code with you.

Email address

phong.cao@stonybrook.edu PRIMARY

phongvcao@phongvcao.com

Step 2: Download and Install Git for Windows and/or Mac OSX:

1. If you decide to use an IDE to develop this project then you can skip this step, since Git is built-in in most IDE.

2. However, if you are not programming in Java or C# or any language that is not heavily dependent on built-in libraries (such as JavaScript, HTML, CSS, etc.) then I strongly recommend against using an IDE because it offers no better gain (sometimes worse) in productivity, since all the IDE is gonna do in this case is code-highlighting, which you can get with any text editor.

Furthermore, IDE's text editing capability (smart indentation, code highlighting, screen division, etc.) is sub-par and usually worse than professional text editors like Vim or Emacs (these ones are hard to use but once you get a hold of them your speed is god-like), Sublime Text (recommended for non-Vim & non-Emacs), Notepad++ (for Windows-lovers), TextMate or TextWrangler (poor Mac users), BBEdit (rich Mac users), etc., all of which are terrific in features and truly improve your productivity with non-heavily-libraries-dependent languages.

3. There are actually many GUI versions of Git available online, notably Atlassian SourceTree (available on Mac and Windows only) which I also recommend against using because it hides lots of Git behaviors and not so good for learning purpose (which is exactly what we are after). If you have to use non-commandline Git (for example you're coding in Java and you have to use an IDE) then it is acceptable not to use commandline Git. :D

4. Finally (sorry for my verboseness but I just want to make things clear), here is the download link for Git command-line (both Windows and Mac): http://git-scm.com/downloads
For Linux, I believe that you can just issue an install command to get it from the repo, something like `sudo yum install git` (Fedora, YUM, rpm package) or `sudo apt-get install git` (Ubuntu, Aptitude, deb package).

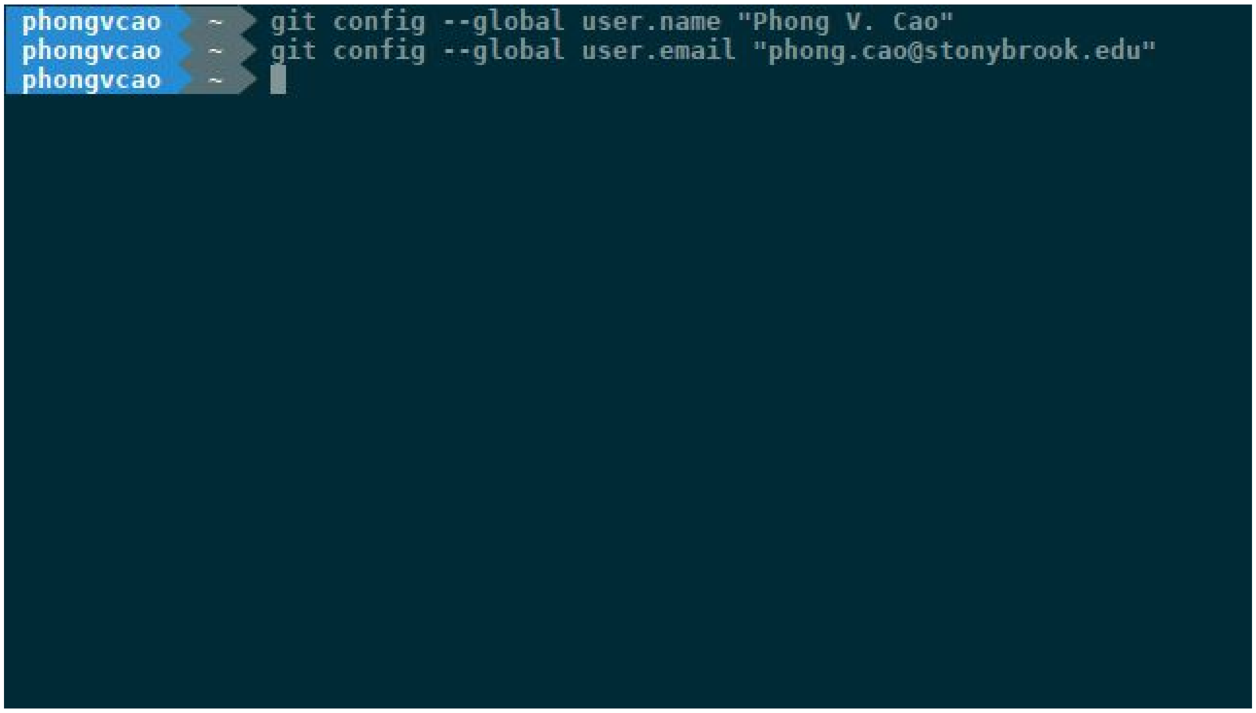
5. This tutorial is based on the official online Git book ("Pro Git") which can be viewed here: http://git-scm.com/book.

6. The tutorial will mostly present examples and screenshots of Git commandline, but the steps should apply similarly in IDE-embedded Git. Thus, if you don't have to use an IDE (see the reasons above), please please please use Git command-line to make this tutorial easier to understand and learn interesting Git stuffs.

Step 3: First-time set-up for your Git:

1. Open your command-line Git and set your Git public identity for easier collaboration and identification as follow:

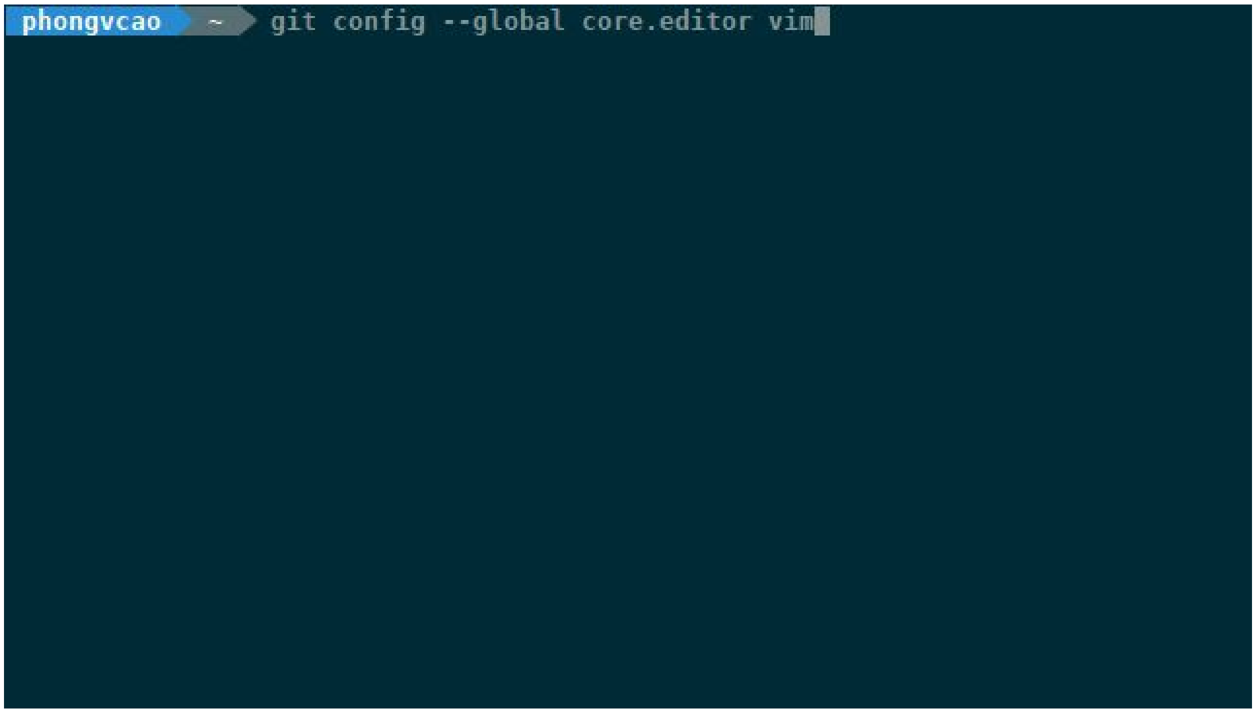
```
$ git config --global user.name "<First Name> <Last Name>"
$ git config --global user.email "<Your Stony Brook Email Address>"
```



```
phongvcao  ~  git config --global user.name "Phong V. Cao"
phongvcao  ~  git config --global user.email "phong.cao@stonybrook.edu"
phongvcao  ~
```

2. By default, Git lets you type in your commit message using the system's command-line. However, you can set Git to use a different editor for your commit message:

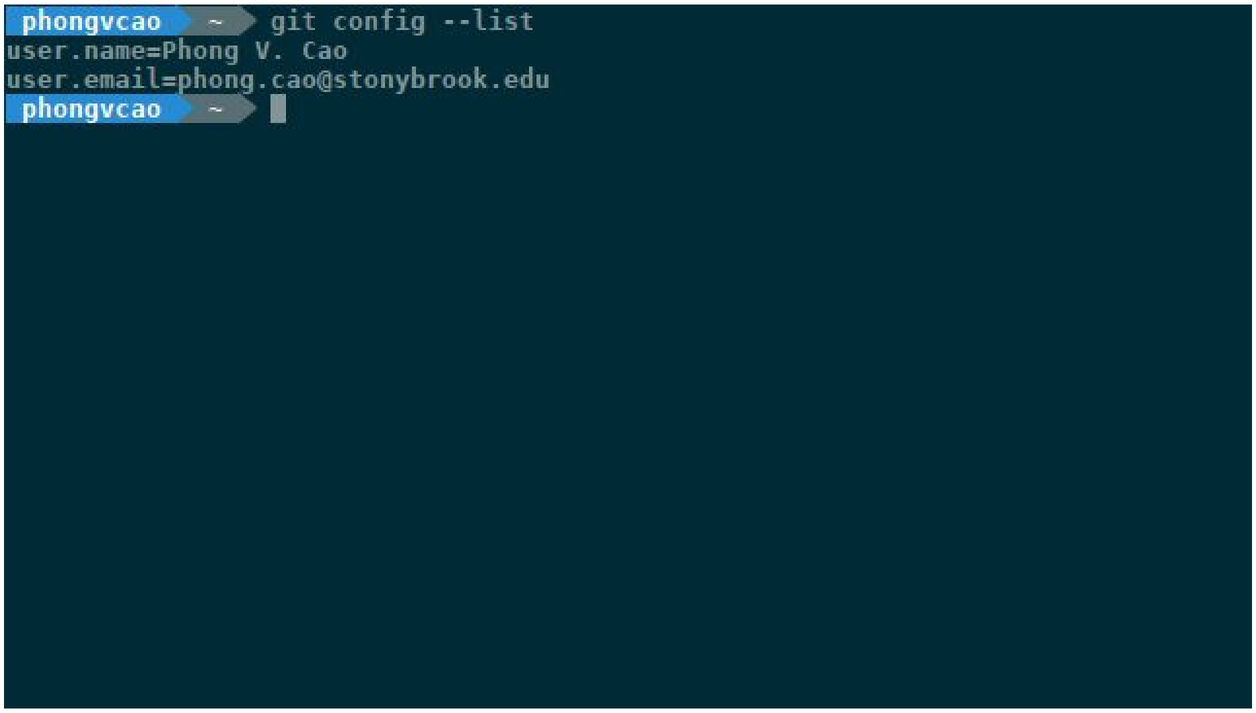
```
$ git config --global core.editor <Your Text Editor of Choice>
```



```
phongvcao  ~  git config --global core.editor vim
```

3. If you want to check all of your Git settings, issue the following command:

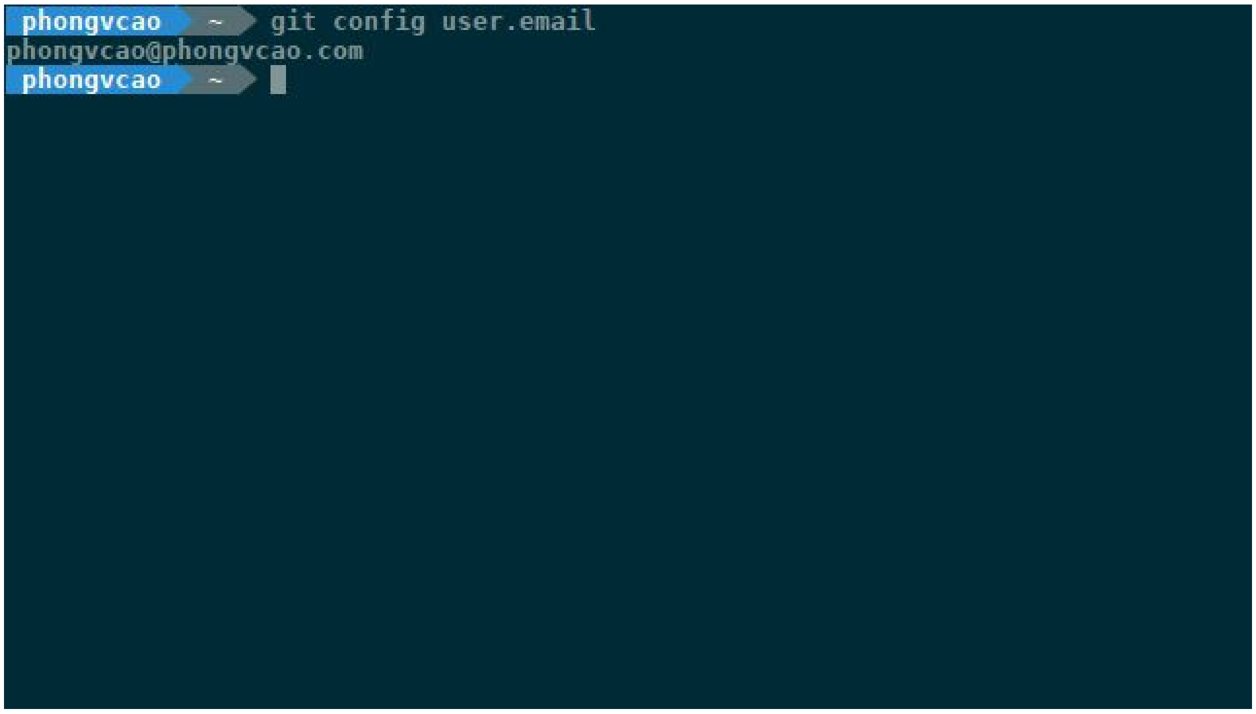
```
$ git config --list
```



```
phongvcao  ~  git config --list
user.name=Phong V. Cao
user.email=phong.cao@stonybrook.edu
phongvcao  ~  
```

Or the following command if you just want to check for a particular setting:

```
$ git config <Setting’s Name>
```



```
phongvcao  ~  git config user.email
phongvcao@phongvcao.com
phongvcao  ~  
```

Step 4: Generate SSH Key:

So what is an SSH Key? Well simply put, SSH Key is a simple way that helps Bitbucket or Github or any Git server knowing who has the rights to modify and/or push/pull to/from an online repository.

It is similar to regular website log-in: Say you save some of your files on MegaUpload and want to modify their content. What are you gonna do? Of course, you will have to download and modify them or edit the already-existed version of the files on your local hard drive. Then you will have to upload the files into your account. In this upload step, MegaUpload will first require you to log-in into your account before you can upload your files there (if not, everyone can upload and delete and do whatever they want with your files and that is worse than horrible!)

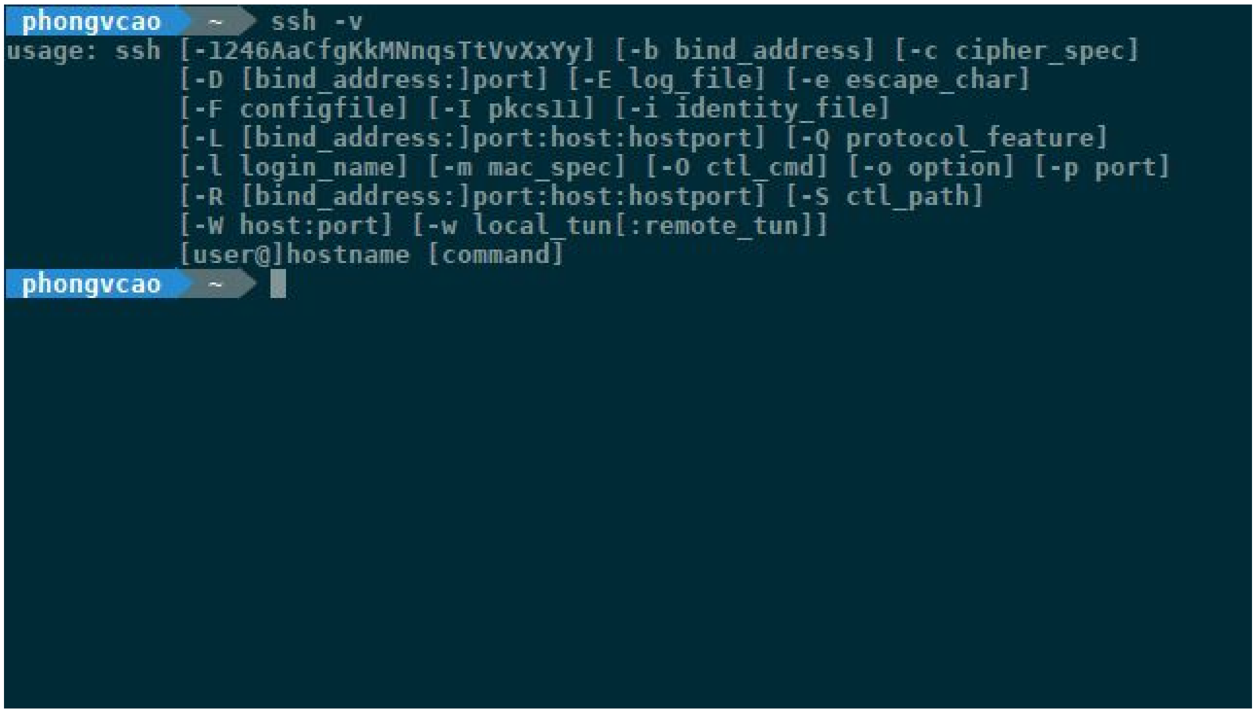
The way Git and Github and Bitbucket work is similar, except that it uses SSH Key instead of the log-in dialog to let you log-in. But the difference between SSH Key and normal log-in is that SSH Key allow multiple username and password to modify a repository, while normal log-in usually only allows one user logging in at a time. SSH Key is the way of log-in and authentication for software development project where there are lots of contributors, not just the author.

In order to achieve all of these fancy stuffs, you have to first have an SSH Key. To get one, follow the steps below:

1. Check if you have existing default Identity:
a. Open Git (or Git Bash on Windows)
b. Enter the following command to verify the SSH client is available:

```
$ ssh -v
```

If you are using Git Bash (Windows) or have ssh installed (Mac OSX), you should see the following output:

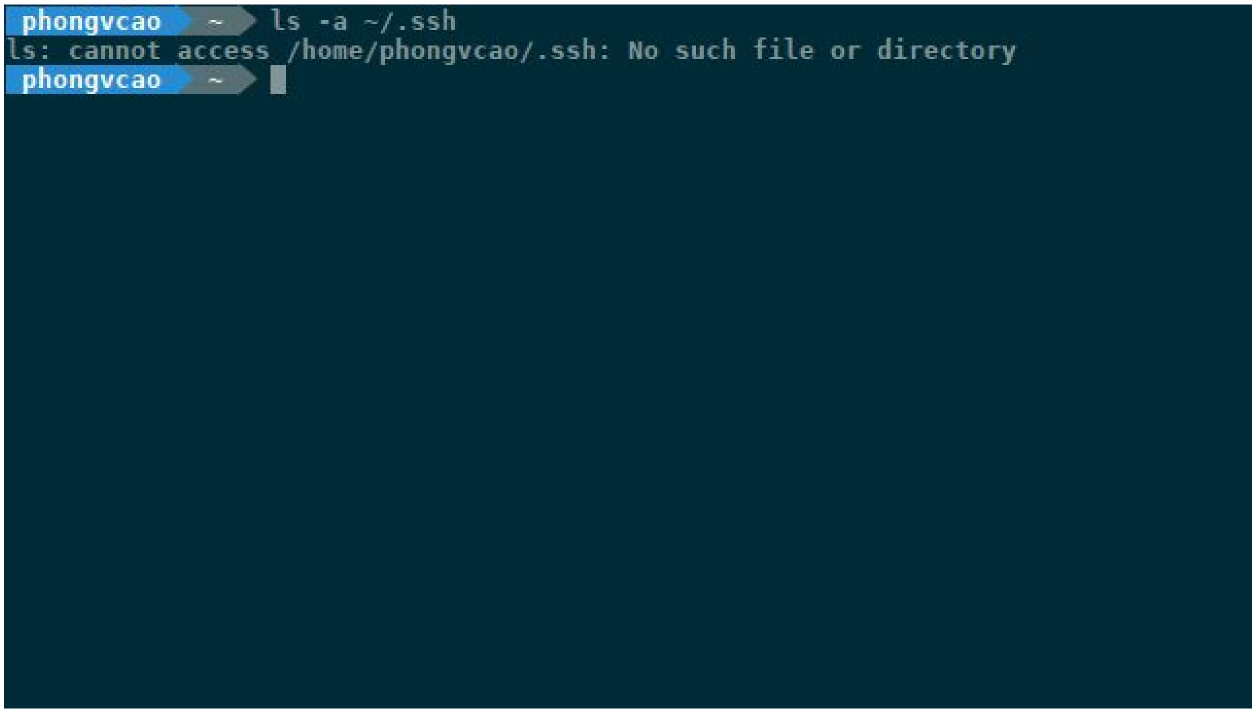


```
phongvcao  ~  ssh -v
usage: ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport] [-Q protocol_feature]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-R [bind_address:]port:host:hostport] [-S ctl_path]
           [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
phongvcao  ~
```

c. Now that you have ssh installed, enter the following commands to check if you already have an SSH key:

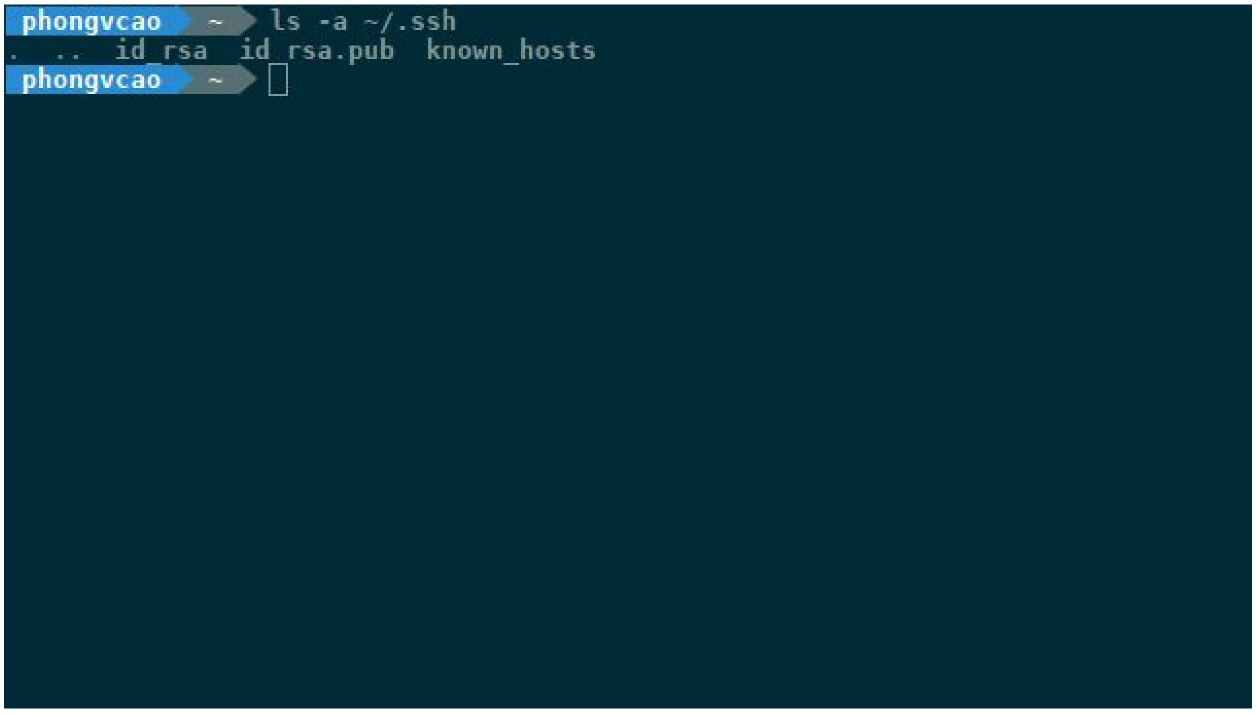
```
$ ls -a ~/.ssh
```

If you have not used SSH on Git Bash or Terminal you will see something like this:



```
phongvcao  ~  ls -a ~/.ssh
ls: cannot access /home/phongvcao/.ssh: No such file or directory
phongvcao  ~ 
```

If you have already had a default identity, you will see two id_* files (Skip immediately to Step 5):



```
phongvcao  ~  ls -a ~/.ssh
.  ..  id_rsa  id_rsa.pub  known_hosts
phongvcao  ~ 
```

The content of the id_rsa.pub file is what we are after.

d. If you haven't yet had the SSH Key files, issue the following command to create them:

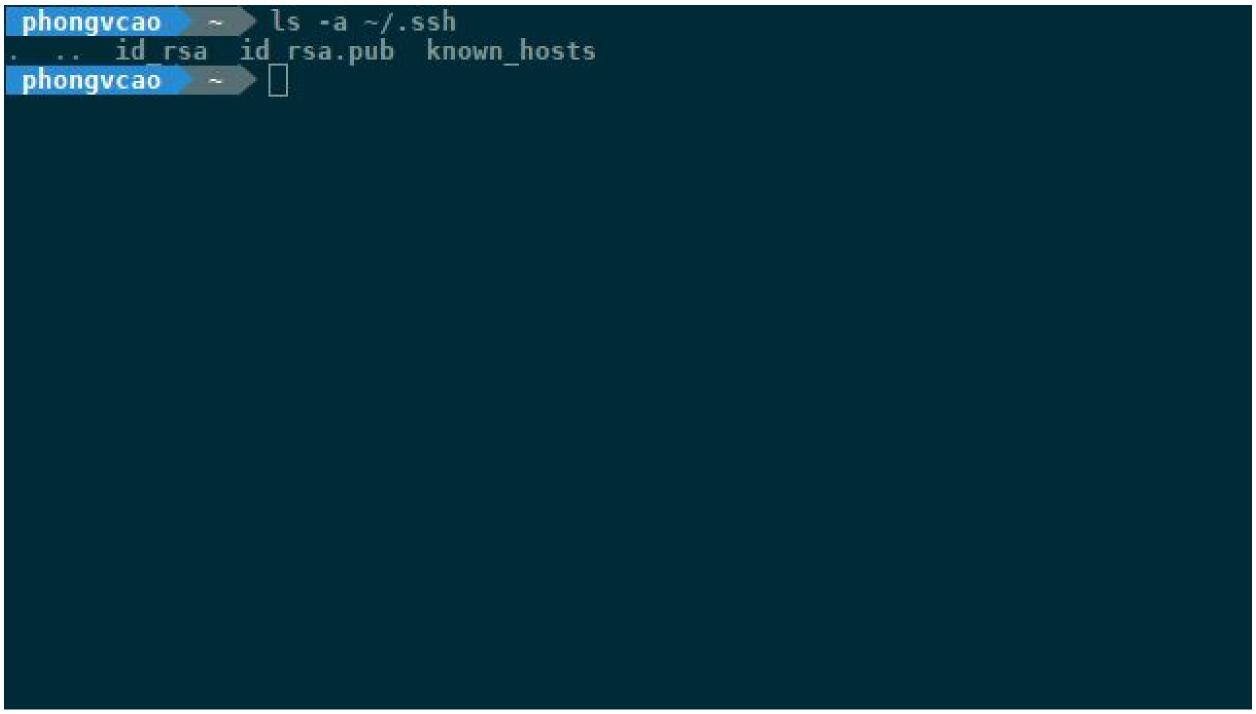
```
$ ssh-keygen
```

If it asks "Enter file in which to save the key: " and "Enter passphrase: " and "Enter same passphrase again: " just leave them blank and press <Enter>. The final output should look like this:

After this, issue the command in Part (c) above to make sure that the SSH Key has been generated:

```
$ ls -a ~/.ssh
```

which should show the following output:



```
phongvcao  ~  ls -a ~/.ssh
.  ..  id_rsa  id_rsa.pub  known_hosts
phongvcao  ~
```

Step 5: Make Git Bash remember your SSH Key:

As stated above, the SSH Key provides a mean of authentication to collaborate on software projects. Now that you got your SSH Key (which contains your "username" and "password"), you must "register" your SSH Key with Bitbucket. In order to do that, follow the following steps:

1. Use your favorite text editor, edit an existing (or create a new) file called "`config`" under `~/.ssh` folder (or in other word, edit or create a new `~/.ssh/config` file) with the following content:

```
Host bitbucket.org
 IdentityFile ~/.ssh/id_rsa
```

This config file is similar to the option “Remember me” or “Remember my Username and Password” when you log-in into Amazon or Google. It helps you not having to enter the whole SSH Key anytime you push your stuffs to Bitbucket.org (it’s a pain in the ass bra!)

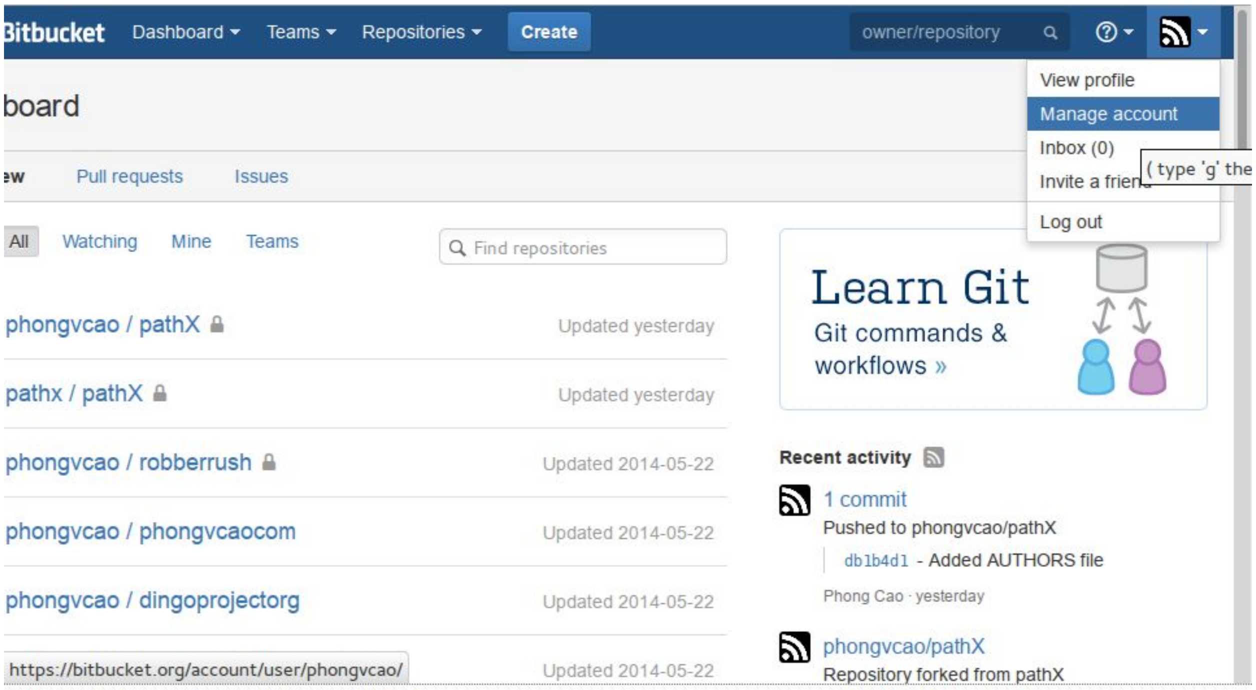
Note that the single space in the 2nd line is VERY important! You have to indent exactly 1 space or Git Bash won’t be able to remember your SSH Key.

2. Save and close the file.

3. Restart your Git Bash terminal

Step 6: Add your SSH Key to Bitbucket:

1. From the top right corner of Bitbucket, go to Manage Account which display the Account Settings page.

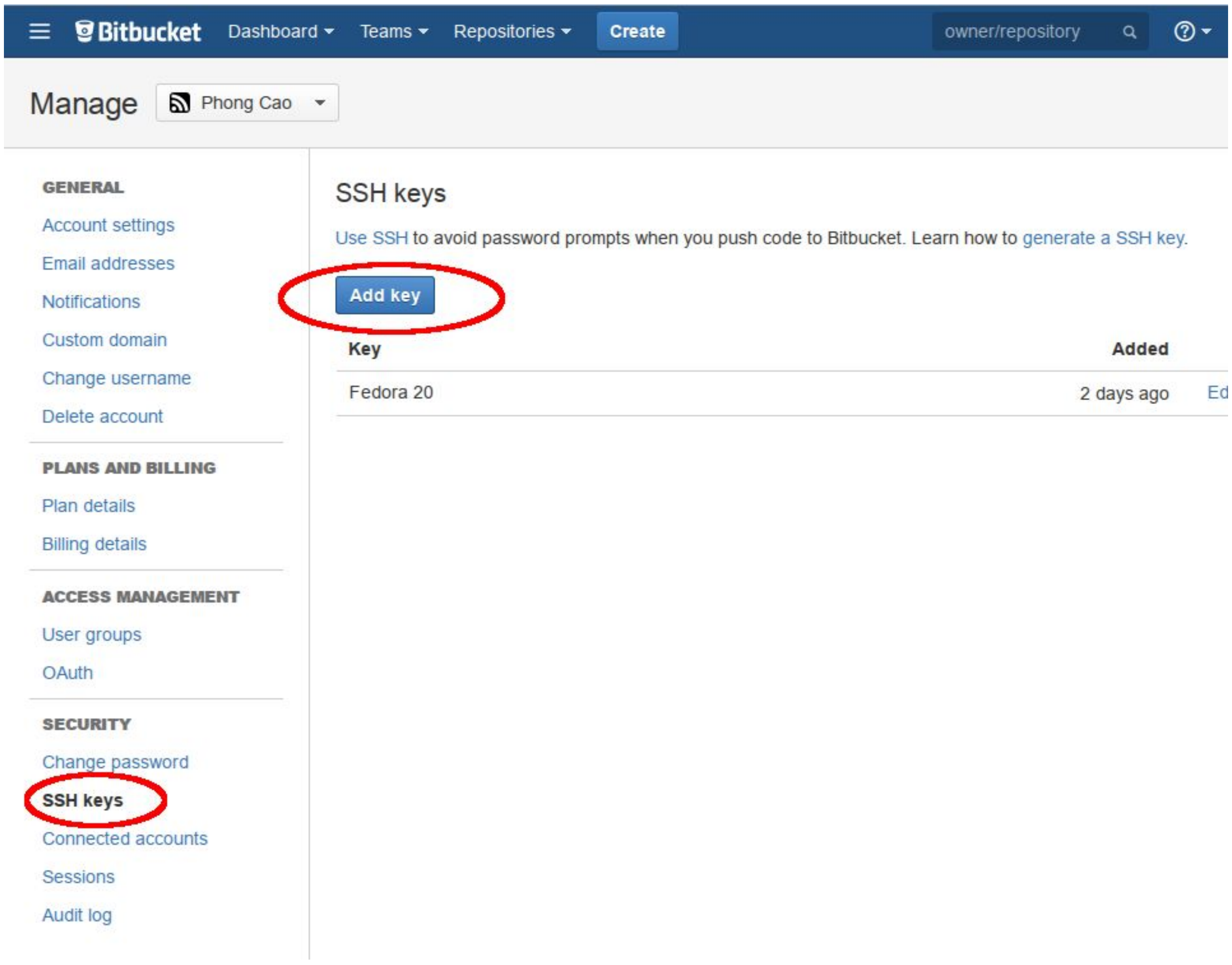


phongvcao / pathX Updated yesterday
pathx / pathX Updated yesterday
phongvcao / robberrush Updated 2014-05-22
phongvcao / phongvcaocom Updated 2014-05-22
phongvcao / dingoprojectorg Updated 2014-05-22
https://bitbucket.org/account/user/phongvcao/ Updated 2014-05-22

1 commit
Pushed to phongvcao/pathX
db1b4d1 - Added AUTHORS file
Phong Cao · yesterday

phongvcao/pathX
Repository forked from pathX

2. In the Account Settings page click on “SSH keys” on the left sidebar, which takes you to the SSH Keys page. Now click on “Add Key” to add your SSH Key to Bitbucket:

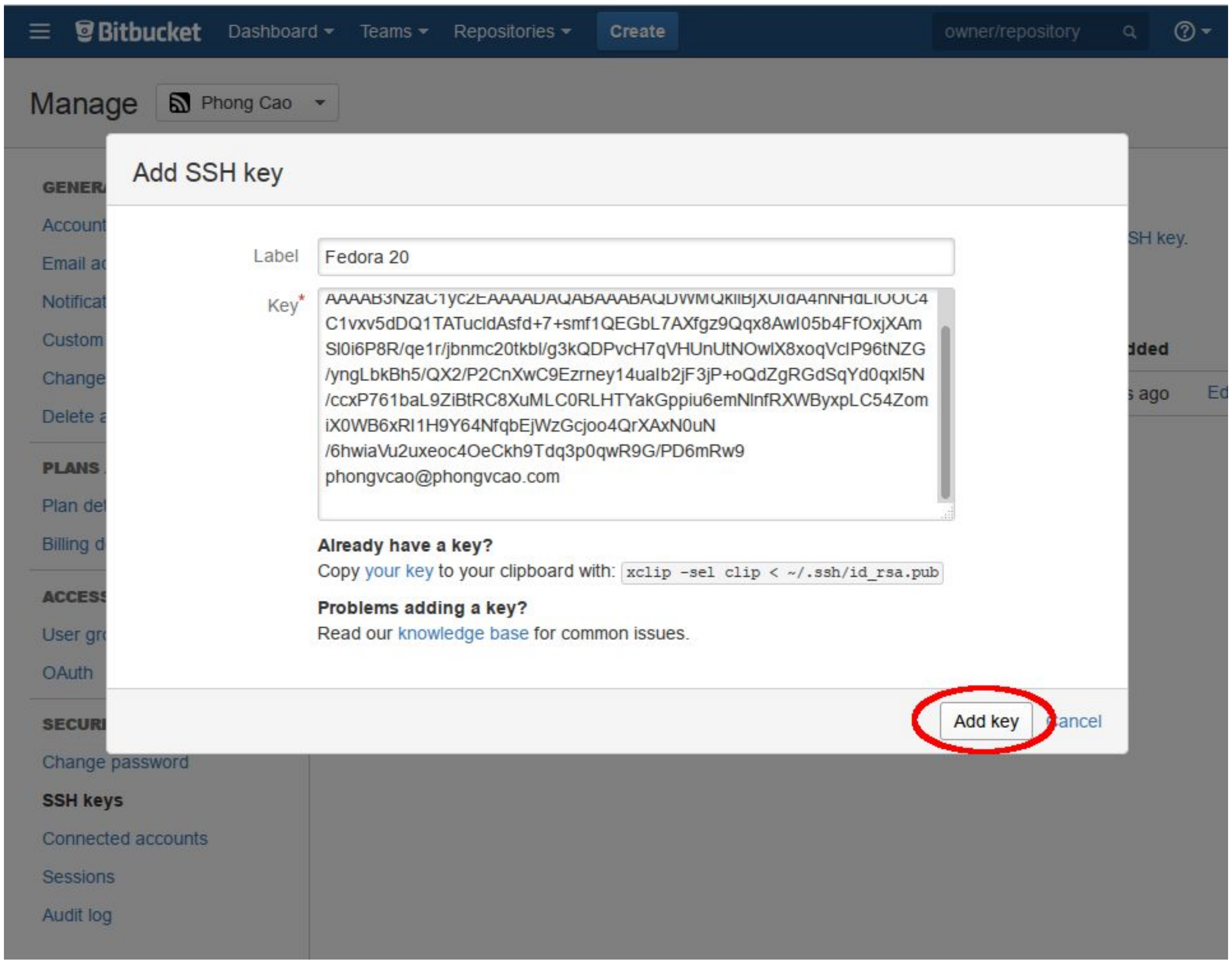


SSH keys

Use SSH to avoid password prompts when you push code to Bitbucket. Learn how to generate a SSH key.

Key	Added	
Fedora 20	2 days ago	Ed

3. A dialog will appear letting you add the key. Simply enter a name for the key in the "Label" textbox and paste the whole content of `~/.ssh/id_rsa.pub` file into the "Key" textarea (the extension `*.pub` indicates that this file contains SSH Key for registering on public sites & repos like Bitbucket. The `id_rsa` file on the other hand only responsible for logging-in when you do a Git push):



Add SSH key

Label: Fedora 20

Key*:

```
AAAAB3NzaC1yc2EAAAADAQABAAABAQDWMQkIIBjXUIdA4nNHdLIOOC4
C1vxv5dDQ1TATucldAsfd+7+smf1QEGbL7AXfgz9Qqx8AwI05b4FfOxjXAm
Sl0i6P8R/qe1r/jbnmc20tkbl/g3kQDPvcH7qVHUnUtNOwlX8xoqVclP96tNZG
/yngLbkBh5/QX2/P2CnXwC9Ezrney14ualb2jF3jP+oQdZgRGdSqYd0qxl5N
/ccxP761baL9ZiBtRC8XuMLC0RLHTYakGppiu6emNlnfRXWByxpLC54Zom
iX0WB6xRI1H9Y64NfqbEjWzGcjoo4QrXAxN0uN
/6hwiaVu2uxeoc4OeCkh9Tdq3p0qwR9G/PD6mRw9
phongvcao@phongvcao.com
```

Already have a key?
Copy your key to your clipboard with: `xclip -sel clip < ~/.ssh/id_rsa.pub`

Problems adding a key?
Read our knowledge base for common issues.

4. Finally, click "Add" to add the SSH Key into your Bitbucket account.

Step 7: Fork PathX to your Bitbucket account:

Okay so before going deeper, you need to understand what a "fork" is. A fork is simply a copy of the whole project (from the original repository) on your own Bitbucket account. When you "fork" a project on Bitbucket or Github or any other service, what it does is basically copying all the files & folders (including the `.git` folder) from the original repo to your account.

So why do we need to fork? Well it helps leader of projects that have more than 1 contributors to manage workflow better and avoid conflicts. Say you want to make a change to the source code or do some big modification that may affect (or even break!) the software.

Typically, if this is your own personal project where you're the only contributor, you would just clone the whole repository down to your computer and modify and then push it back later. However, in large software projects with dozens, hundreds and thousands contributors, if everyone clone the main repo down and modify the way they want to and then being able to push the repo back to Bitbucket, then there will always be code conflicts in the source code that break the software lots of time, causing anger for other contributors (especially those who debug) and slow down the development process.

Thus, “forking” was invented to solve this problem. Here is the whole process of “forking”:

- Create a fork on Bitbucket.
- Clone the forked repository your local system.
- Modify the local repository.
- Commit your changes.
- Push changes back to the remote fork on Bitbucket.
- Create a pull request from the forked repository (source) back to the original (destination).

This makes the whole process of collaborating easier because modifying your own “fork” will not affect or break the original repository. When you’re done with your changes and new-feature implementation (and you’re sure that it is working correcly through rigorous testing!), you can do a Git commit and push the changes back to your fork.

After that, you email or message the *original* repository administrators (in the case of PathX it is me, Prof. Richard McKenna and Danny Gibson) to test the changes and create a “Git merge” & “Git commit” plan.

If the changes work well (meaning they cause no conflicts with other codes), the repository administrators will “Create a Pull Request” to your fork and merge your changes with the original Bitbucket repository. And bingo: your changes now become official parts of the original Bitbucket repository!

In reality (and also with our project), even though we as administrators have the right to push/pull/modify/delete the *original* pathX Bitbucket repository and can do whatever we want with the official, *original* codes, in order to respect the changes made by other programmers and avoid conflicts with codes created by other team members, we (me, Prof. Richard McKenna and Danny Gibson) still have to create our own forks of the project in our own Bitbucket accounts and Create a Pull Request for ourselves normally to merge our codes with the main *original* pathX repository like any other team members.

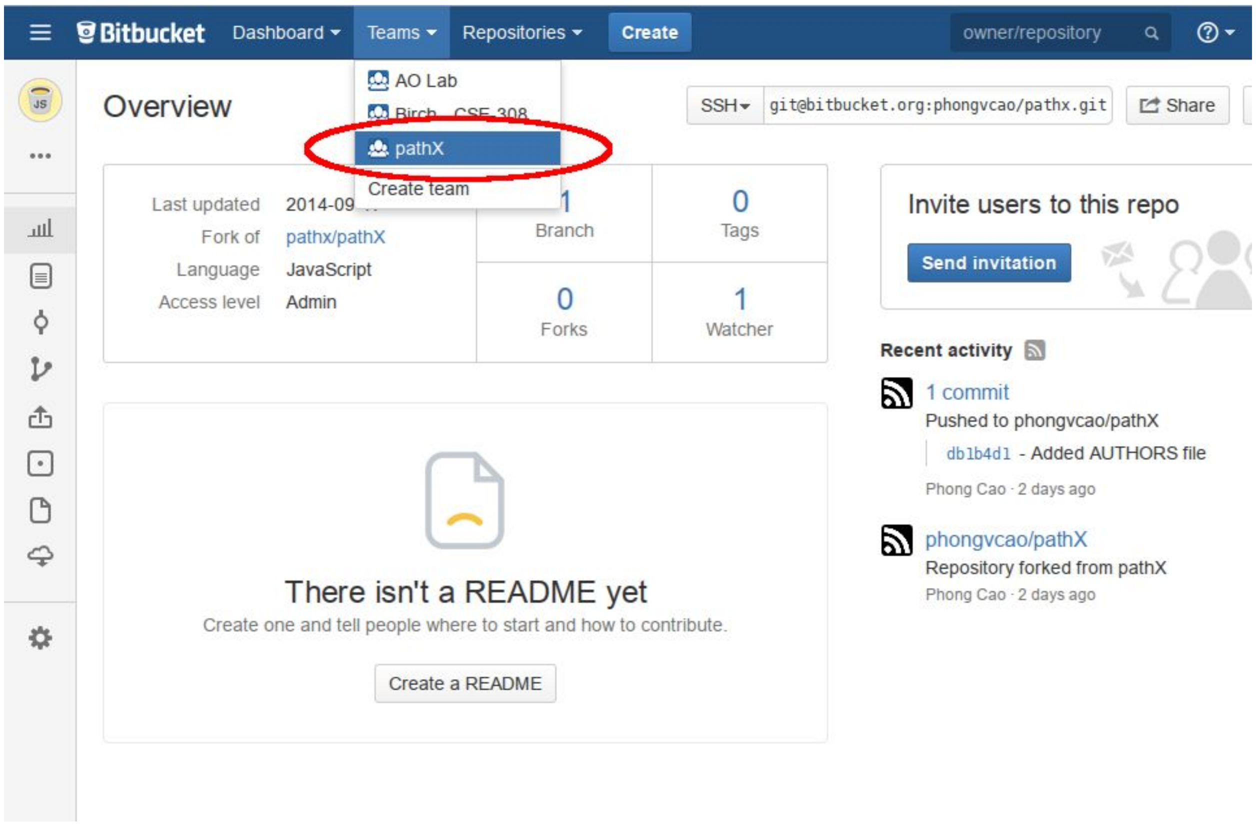
The word “Administrators” of the repository are for show only (we should be called “Documentors” or “Secretaries” instead). The people that are administrators are not actually the “bosses” or “owners” of the repository themselves, but rather a group of people that know the most about how Git and Bitbucket work so that we would be able to resolve “Git merge” code conflicts (more on this later) for team members and also take care of repository maintenance (changing folders, changing names, changing Wiki documentation and other stuffs). If we make everyone in our team “Administrators” of the official, *original* pathX repository, which means that everyone has the right to push/pull/modify/delete/do_whatever_they_want_to_do with the repository, then those team members that know little or nothing about Git may mess up the whole project by issuing a wrong merge, a reckless commit, or push their untested code to the official repo, which negatively affects the workflow and development speed of the whole team.

Thus, if you are not made the “Administrators” of the official pathX project repository on Bitbucket then please don’t get offended or feel like we don’t trust your coding/technical compentence. We just have to do this for safety reasons since if you’re reading this guide it means that you’re quite inexperienced with Git & Bitbucket team collaboration & workflow. We as “Administrators” are not the ones who are in charge or own the project (the whole team is in charge and own the project, not only us). We are just “Secretaries” or “Document Maintainers” who know the most about Git to ensure that our team members can collaborate through Bitbucket smoothly and avoid reckless, untested push/pull from team members.

After the project has going on for a while (let’s say by the end of October, when you all know your ways around Git and Bitbucket), we will make the whole team Administrators. At the moment for Git-safety and collaboration-safety reasons we cannot make everyone repository Administrators yet.

With everything about fork being said above, here are the steps to fork your project from the original PathX repository to your Bitbucket account:

1. From the top Bitbucket navigation bar, click on Team -> pathX. This will bring you to the Bitbucket page for pathX Team:



Last updated	2014-09
Fork of	pathx/pathX
Language	JavaScript
Access level	Admin

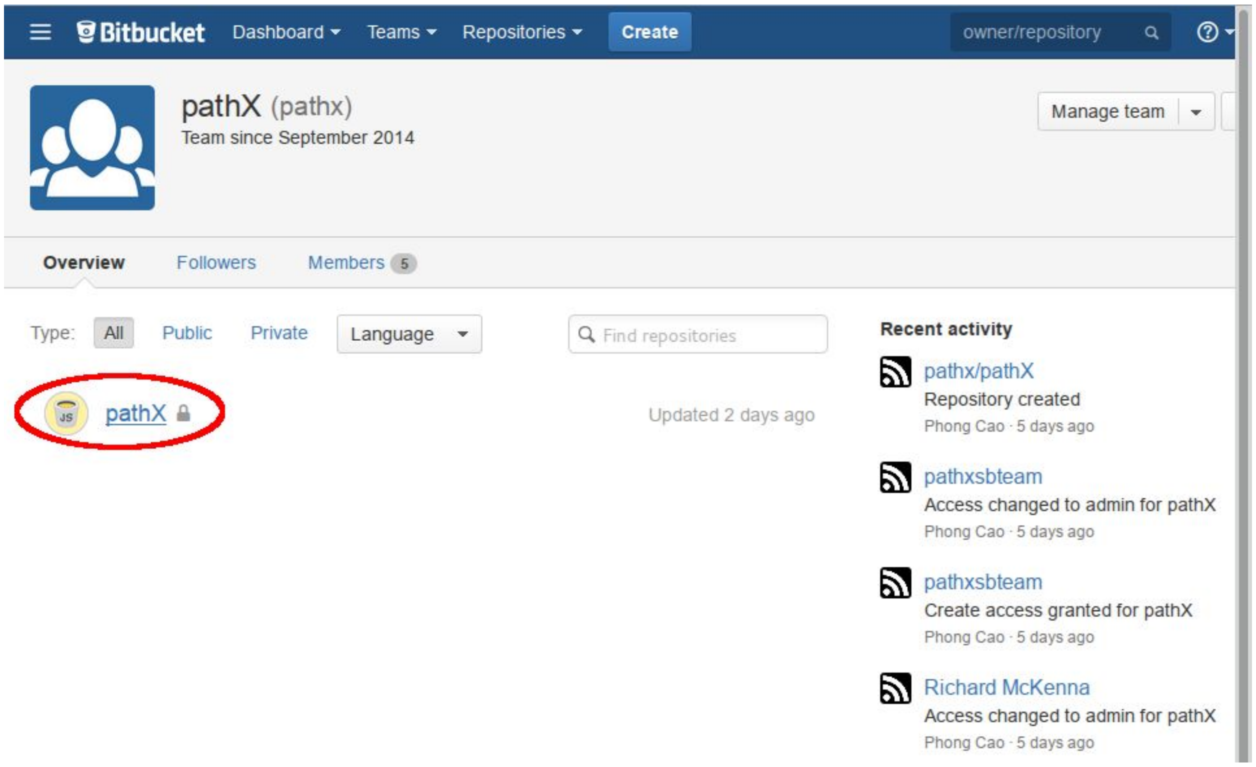
1 Branch · 0 Tags · 0 Forks · 1 Watcher

Recent activity

1 commit
Pushed to phongvcao/pathX
db1b4d1 - Added AUTHORS file
Phong Cao · 2 days ago

phongvcao/pathX
Repository forked from pathX
Phong Cao · 2 days ago

2. Now click on the URL for pathX project, which would lead you to the pathX project main page:



pathX (pathx)
Team since September 2014

pathX — Updated 2 days ago

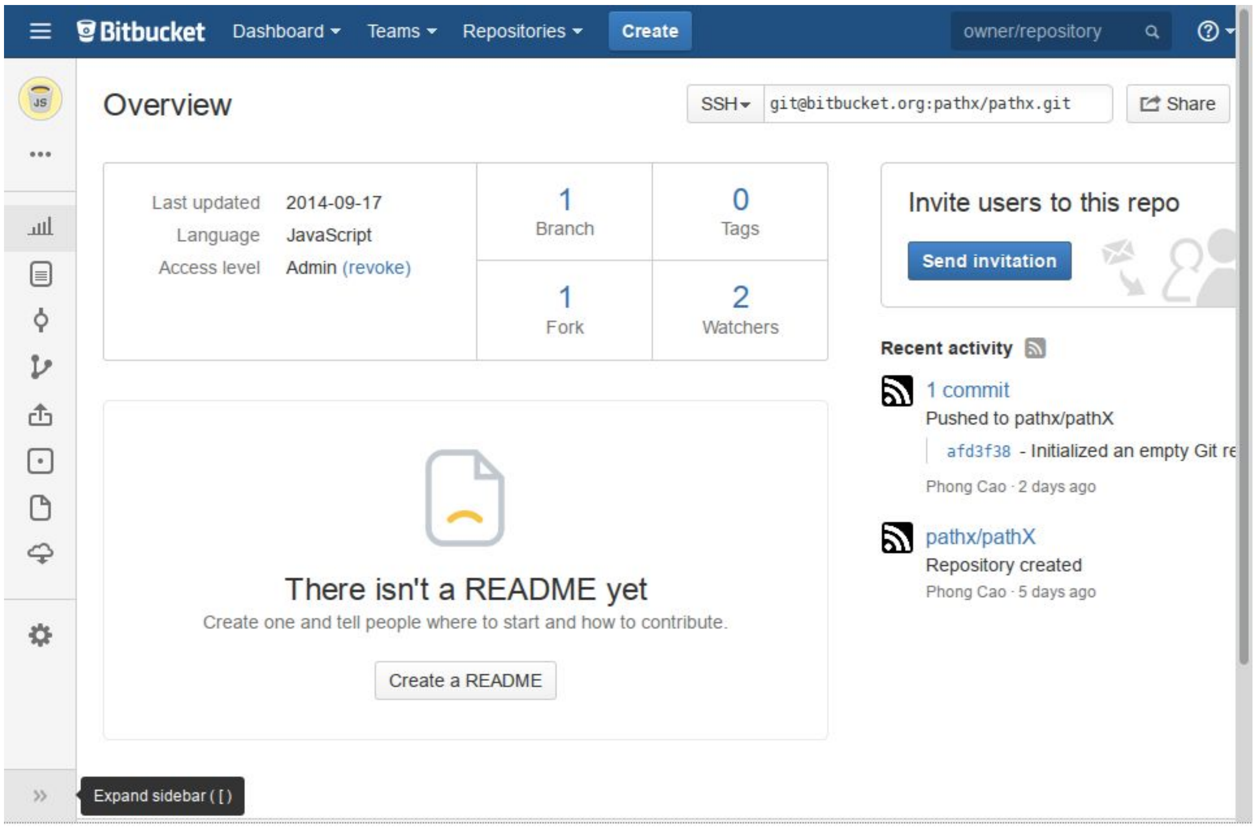
pathx/pathX
Repository created
Phong Cao · 5 days ago

pathxsbteam
Access changed to admin for pathX
Phong Cao · 5 days ago

pathxsbteam
Create access granted for pathX
Phong Cao · 5 days ago

Richard McKenna
Access changed to admin for pathX
Phong Cao · 5 days ago

3. On the pathX project homepage, click on "Expand Sidebar" at the bottom to expand the Bitbucket Sidebar:



git@bitbucket.org:pathx/pathx.git

Last updated	2014-09-17
Language	JavaScript
Access level	Admin (revoke)

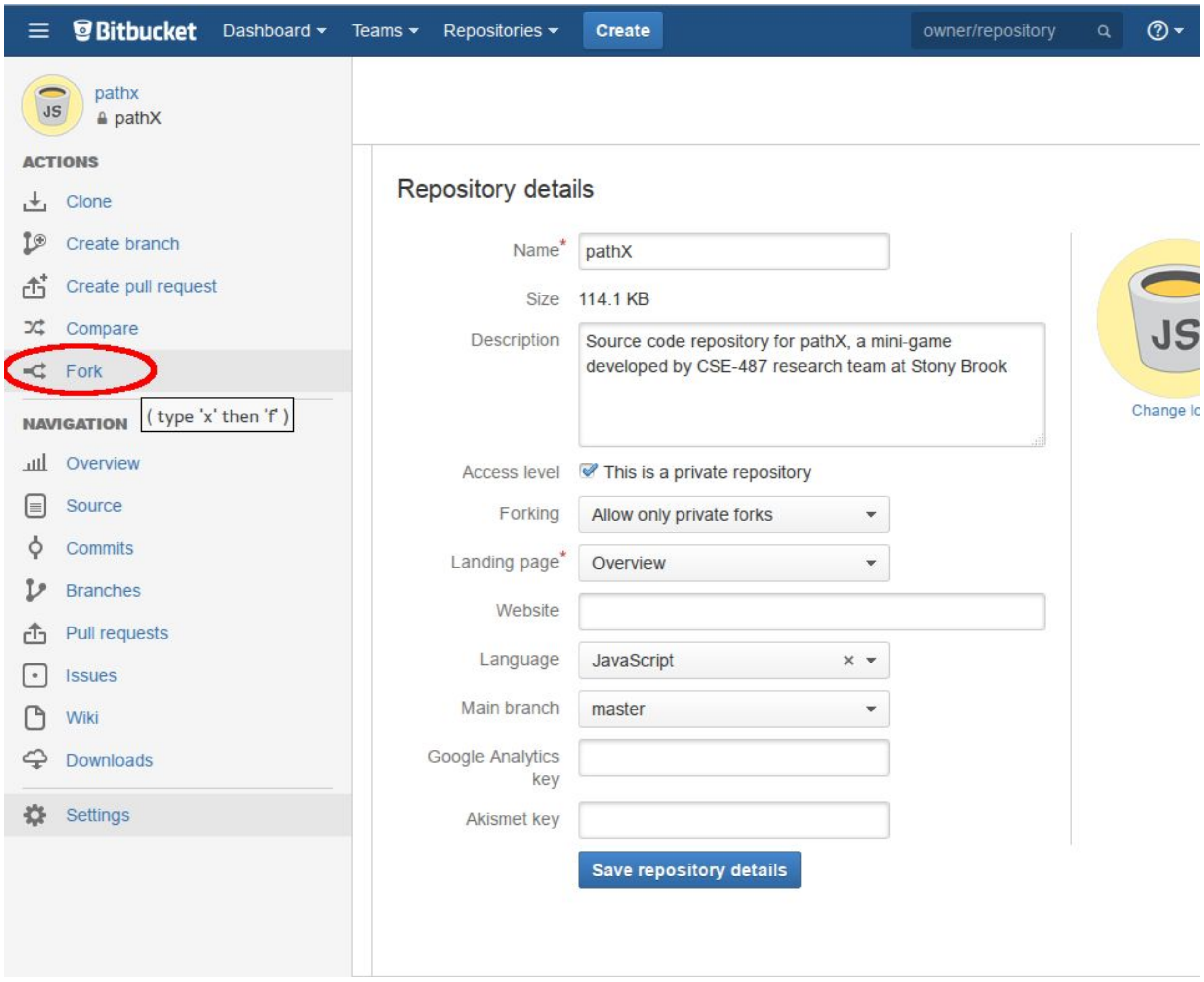
1 Branch · 0 Tags · 1 Fork · 2 Watchers

There isn't a README yet
Create one and tell people where to start and how to contribute.

1 commit
Pushed to pathx/pathX
afd3f38 - Initialized an empty Git re
Phong Cao · 2 days ago

pathx/pathX
Repository created
Phong Cao · 5 days ago

4. On the expanded Bitbucket Sidebar you'll see an option called "Fork". Click on that "Fork" link will lead you to the "Fork" settings page where you can set options for your fork:



Repository details

Name* pathX

Size 114.1 KB

Description Source code repository for pathX, a mini-game developed by CSE-487 research team at Stony Brook

Access level [✓] This is a private repository

Forking Allow only private forks

Landing page* Overview

Website

Language JavaScript

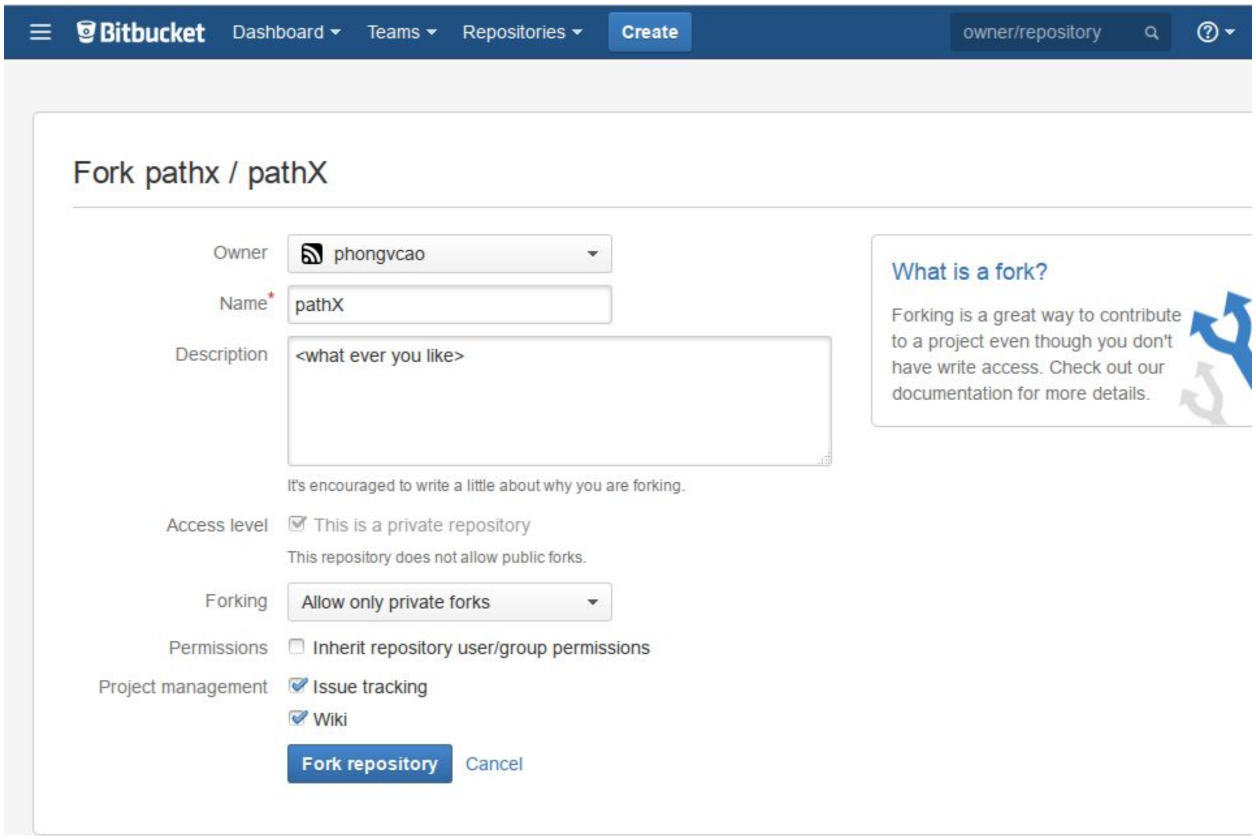
Main branch master

Google Analytics key

Akismet key

Save repository details

5. After the "Fork" settings page is displayed, enter the information as below and click "Fork Repository" to fork the original pathX repo to your Bitbucket account:



Fork pathx / pathX

Owner: phongvcao

Name*: pathX

Description: <what ever you like>

It's encouraged to write a little about why you are forking.

Access level: ✓ This is a private repository

This repository does not allow public forks.

Forking: Allow only private forks

Permissions: ☐ Inherit repository user/group permissions

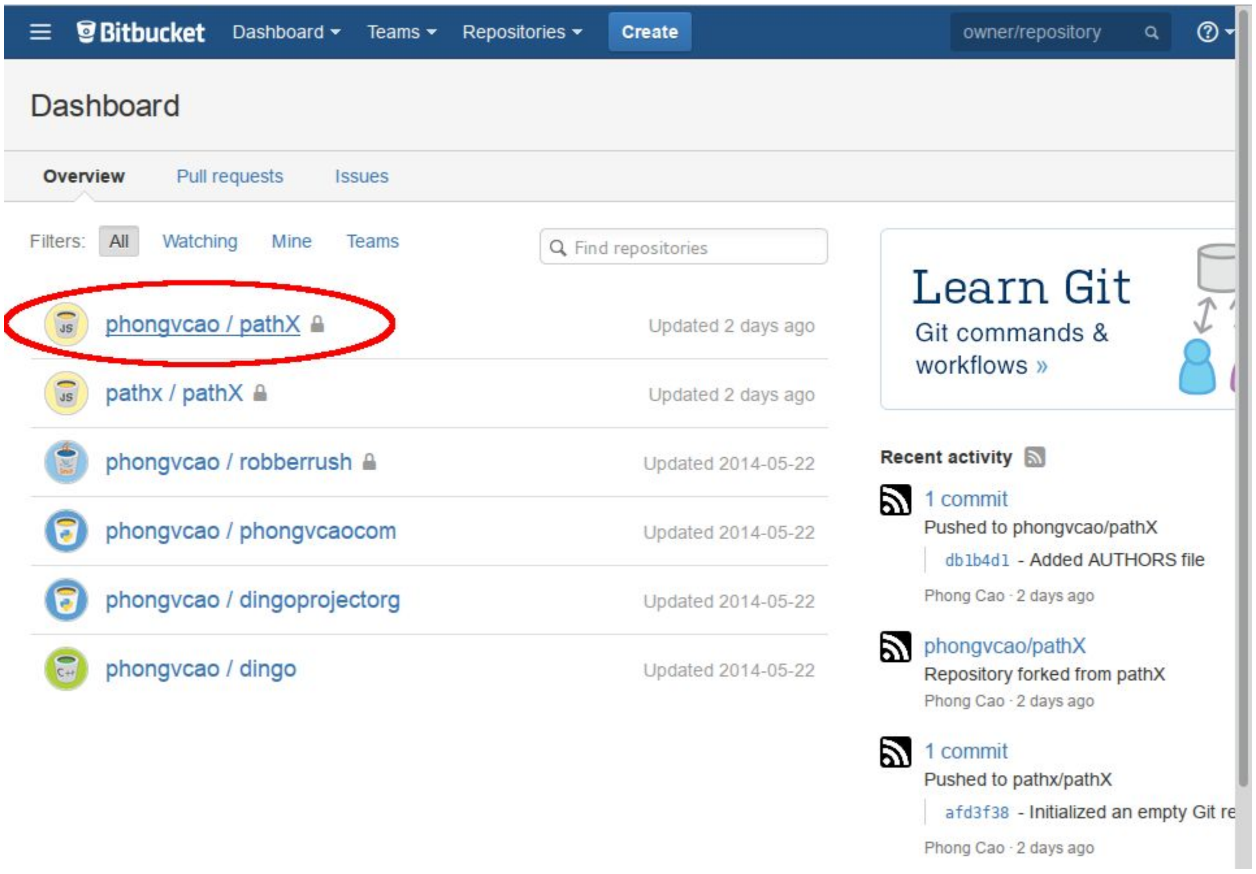
Project management: ✓ Issue tracking

✓ Wiki

What is a fork?

Forking is a great way to contribute to a project even though you don't have write access. Check out our documentation for more details.

6. After this step, go back to your homepage. You should see a project called "<your_Bitbucket_username>/pathX" under your Bitbucket homepage. Click on that project link to get to the Overview page of your pathX fork:



phongvcao / pathX	Updated 2 days ago
pathx / pathX	Updated 2 days ago
phongvcao / robberrush	Updated 2014-05-22
phongvcao / phongvcaocom	Updated 2014-05-22
phongvcao / dingoprojectorg	Updated 2014-05-22
phongvcao / dingo	Updated 2014-05-22

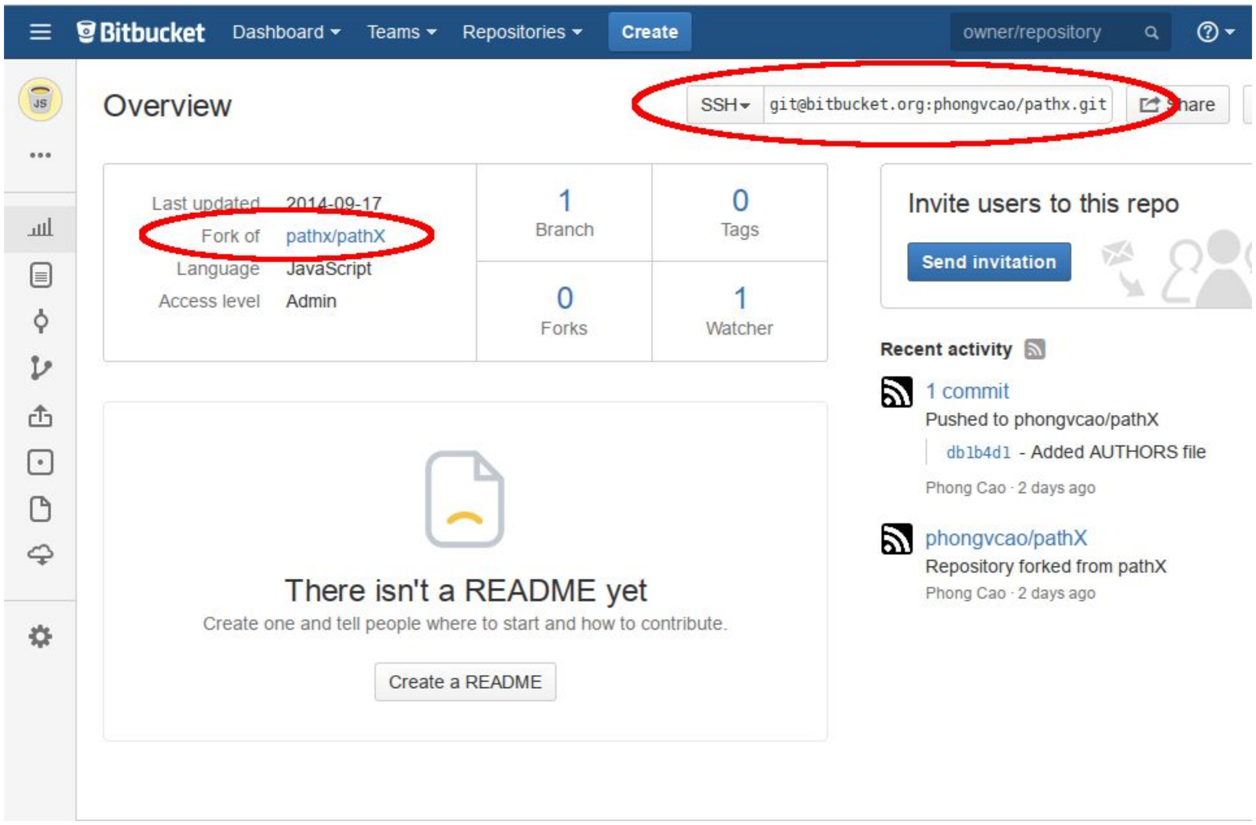
Recent activity

1 commit
Pushed to phongvcao/pathX
db1b4d1 - Added AUTHORS file
Phong Cao · 2 days ago

phongvcao/pathX
Repository forked from pathX
Phong Cao · 2 days ago

1 commit
Pushed to pathx/pathX
afd3f38 - Initialized an empty Git re
Phong Cao · 2 days ago

7. When you're at the Overview page of your pathX fork, you will see a line called "Fork of" which indicates the original repository that this pathX fork belongs to, and on the top-right of the screen you will see an SSH link that you will (later) use to clone the fork to your local computer and update it as changes are made to the original repo:



git@bitbucket.org:phongvcao/pathx.git

Last updated	2014-09-17
Fork of	pathx/pathX
Language	JavaScript
Access level	Admin

1 Branch · 0 Tags · 0 Forks · 1 Watcher

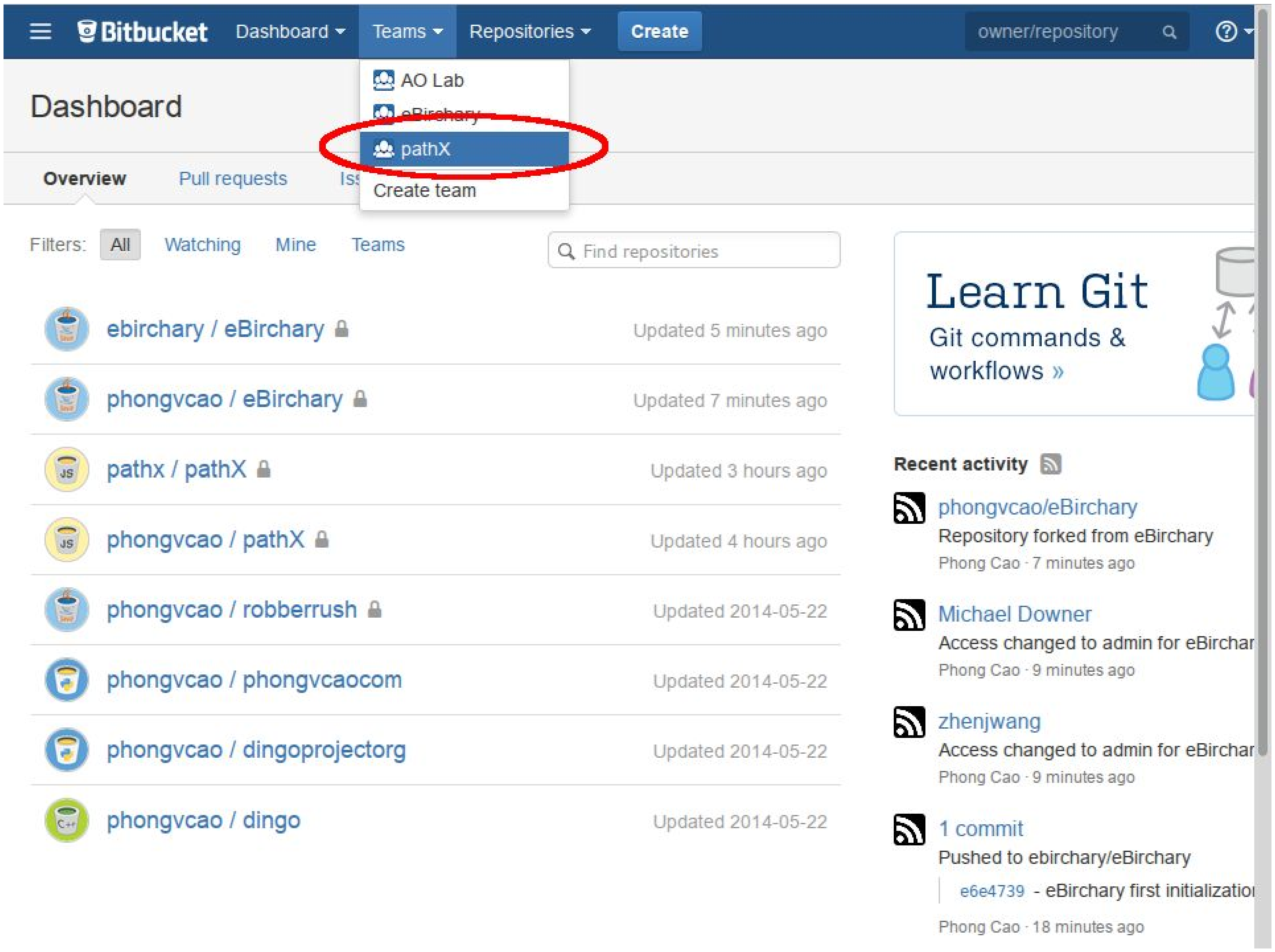
1 commit
Pushed to phongvcao/pathX
db1b4d1 - Added AUTHORS file
Phong Cao · 2 days ago

phongvcao/pathX
Repository forked from pathX
Phong Cao · 2 days ago

8. Now as you're done with the forking of pathX to your Bitbucket account, copy the SSH link (shown in the above screenshot) and move on to the next step!

Step 8: Subscribe to a Bitbucket repository:

1. From the top Bitbucket navigation bar, go to the Bitbucket page for pathX Team and then pathX project repository:



Dashboard

ebirchary / eBirchary	Updated 5 minutes ago
phongvcao / eBirchary	Updated 7 minutes ago
pathx / pathX	Updated 3 hours ago
phongvcao / pathX	Updated 4 hours ago
phongvcao / robberrush	Updated 2014-05-22
phongvcao / phongvcaocom	Updated 2014-05-22
phongvcao / dingoprojectorg	Updated 2014-05-22
phongvcao / dingo	Updated 2014-05-22

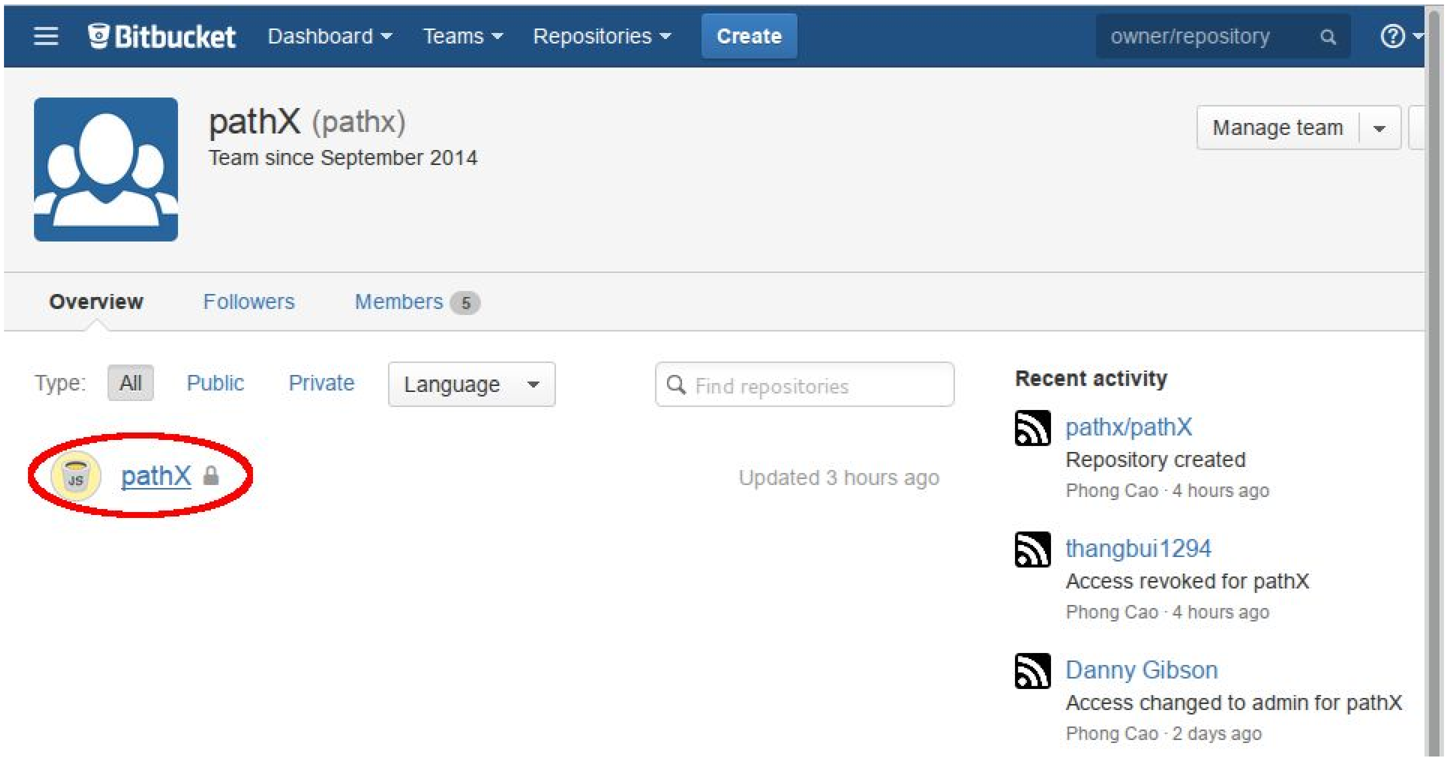
Recent activity

phongvcao/eBirchary
Repository forked from eBirchary
Phong Cao · 7 minutes ago

Michael Downer
Access changed to admin for eBirchar
Phong Cao · 9 minutes ago

zhenjwang
Access changed to admin for eBirchar
Phong Cao · 9 minutes ago

1 commit
Pushed to ebirchary/eBirchary
e6e4739 - eBirchary first initializatio
Phong Cao · 18 minutes ago



pathX (pathx)
Team since September 2014

pathX	Updated 3 hours ago

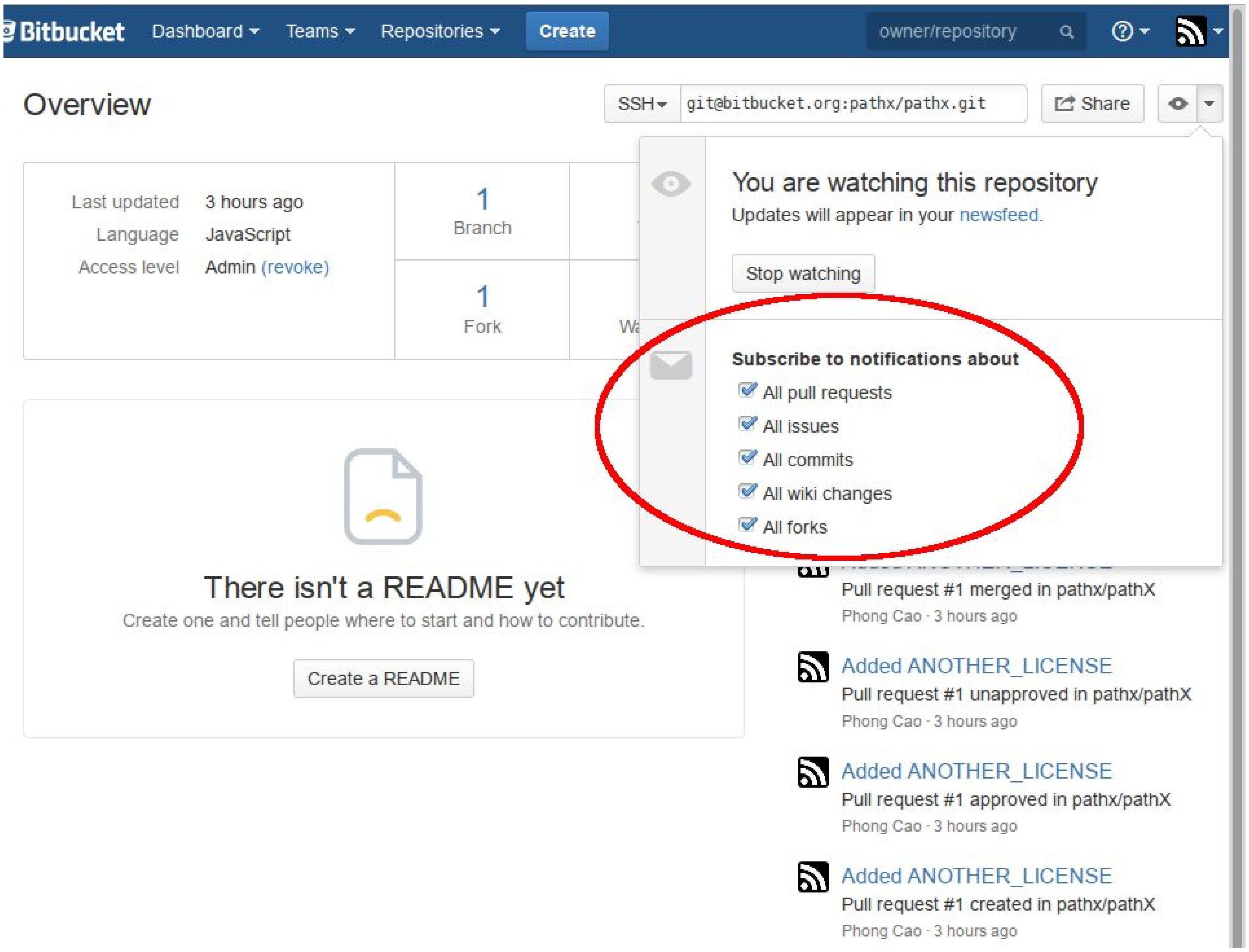
Recent activity

pathx/pathX
Repository created
Phong Cao · 4 hours ago

thangbui1294
Access revoked for pathX
Phong Cao · 4 hours ago

Danny Gibson
Access changed to admin for pathX
Phong Cao · 2 days ago

2. When you're at the Bitbucket page of the pathX repository, click on the "Down" arrow next to the "Watch" button on the top right of the project Bitbucket page. Check all the checkboxes so that you can watch for all the changes (Wiki, Push, Pull, etc.) of the project:



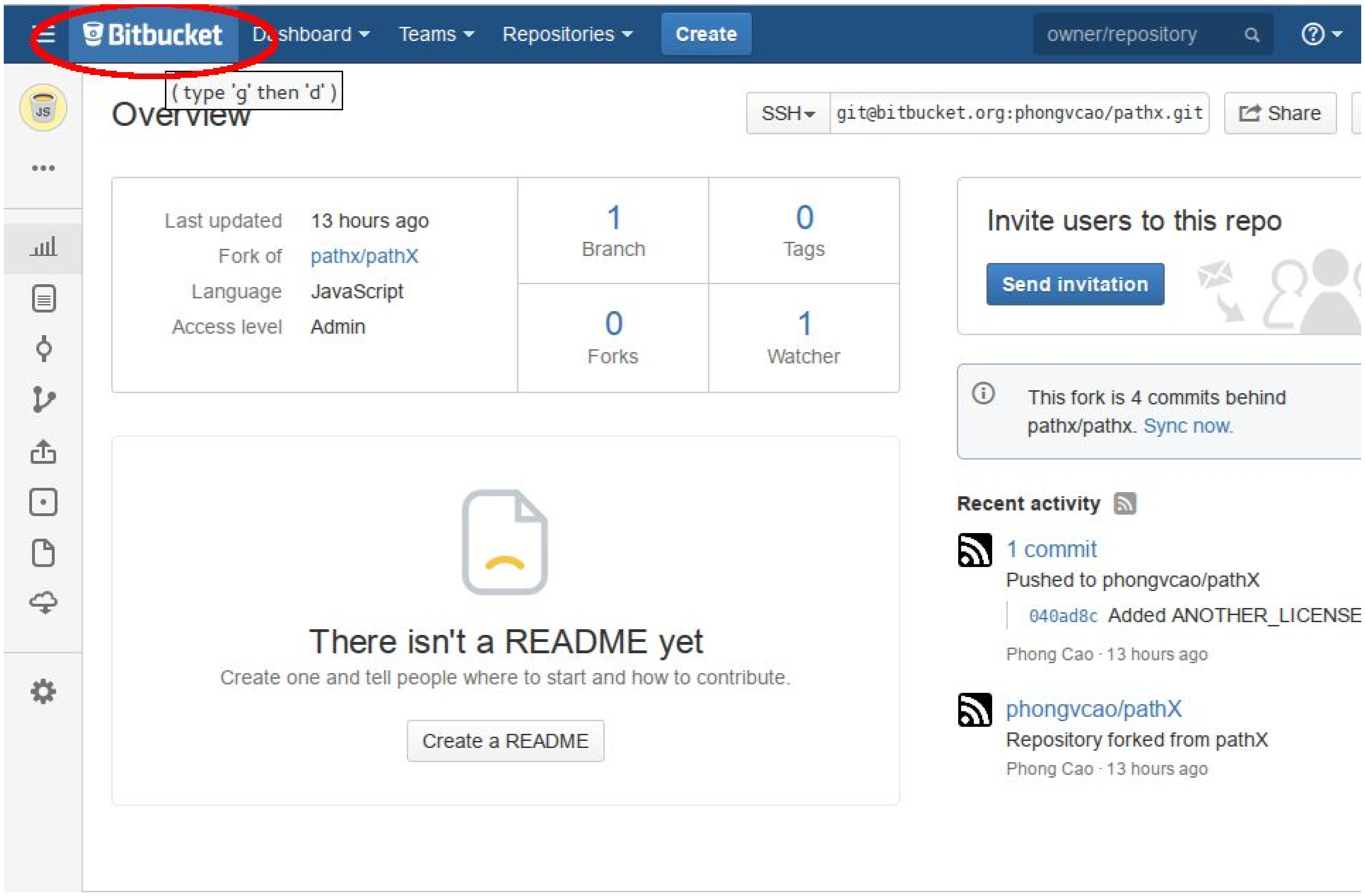
3. Now go to your (fork) pathX repository on Bitbucket and repeat the 2 steps above. This is very important because it lets you know in case there is any team member that forks your pathX fork repository.

Step 9: Grant fork repository access to team members

Now that you have created your private pathX fork repository on Bitbucket, you need to grant the pathX team members access to this repo so that they can view, fork and merge your codes (more on this later) to their codes or the official pathX codes on the **original** pathX repository.

Here are the steps you need to do to grant other team members access to your codes:

1. From the Bitbucket navigation bar, go to your Bitbucket account homepage and then your pathX fork repository's Overview page:



Overview

git@bitbucket.org:phongvcao/pathx.git

Last updated	13 hours ago
Fork of	pathx/pathX
Language	JavaScript
Access level	Admin

1 Branch

0 Tags

0 Forks

1 Watcher

Invite users to this repo

This fork is 4 commits behind pathx/pathx. Sync now.

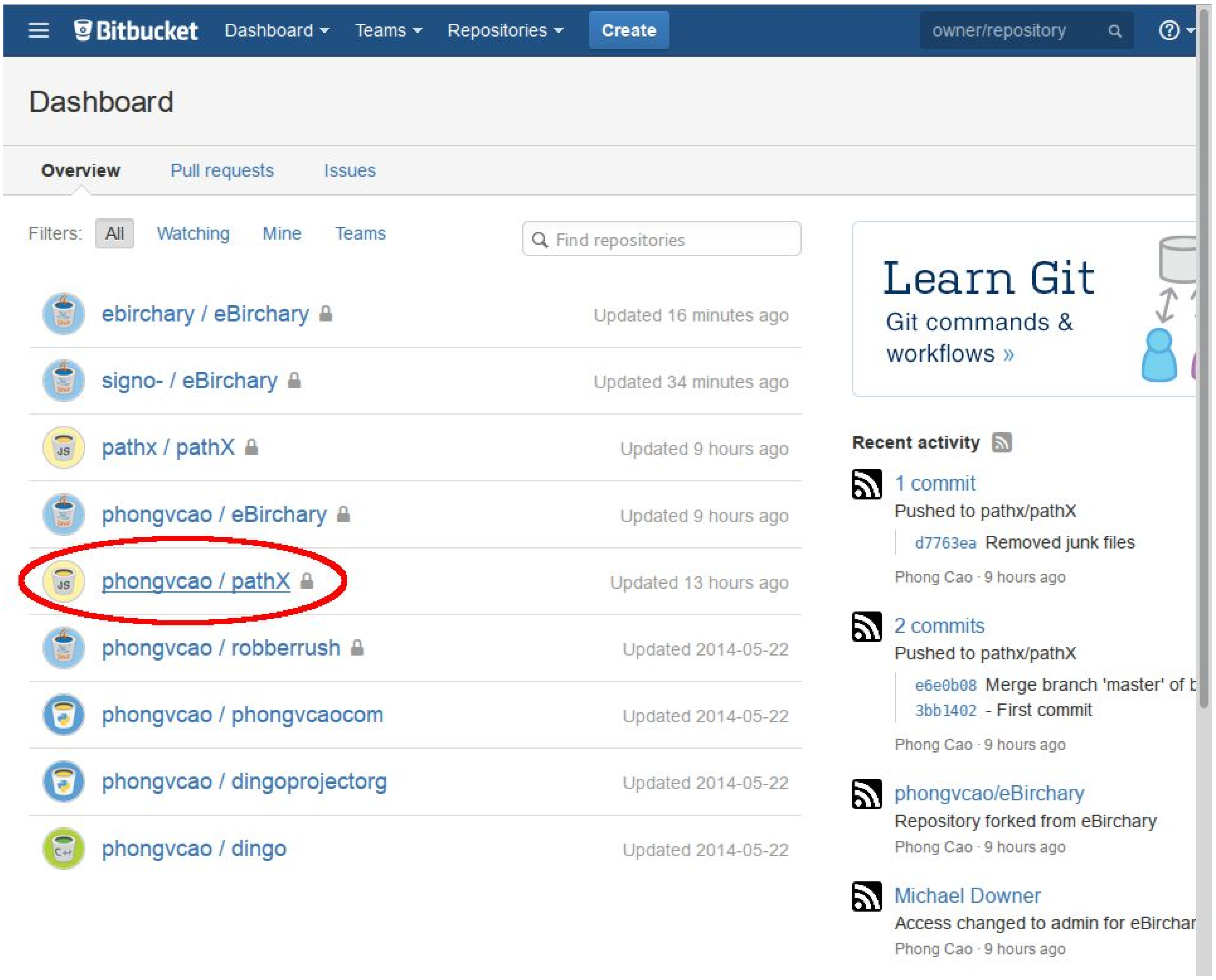
Recent activity

1 commit
Pushed to phongvcao/pathX
040ad8c Added ANOTHER_LICENSE
Phong Cao · 13 hours ago

phongvcao/pathX
Repository forked from pathX
Phong Cao · 13 hours ago

There isn't a README yet

Create one and tell people where to start and how to contribute.



ebirchary / eBirchary Updated 16 minutes ago
signo- / eBirchary Updated 34 minutes ago
pathx / pathX Updated 9 hours ago
phongvcao / eBirchary Updated 9 hours ago
phongvcao / pathX Updated 13 hours ago
phongvcao / robberrush Updated 2014-05-22
phongvcao / phongvcaocom Updated 2014-05-22
phongvcao / dingoprojectorg Updated 2014-05-22
phongvcao / dingo Updated 2014-05-22

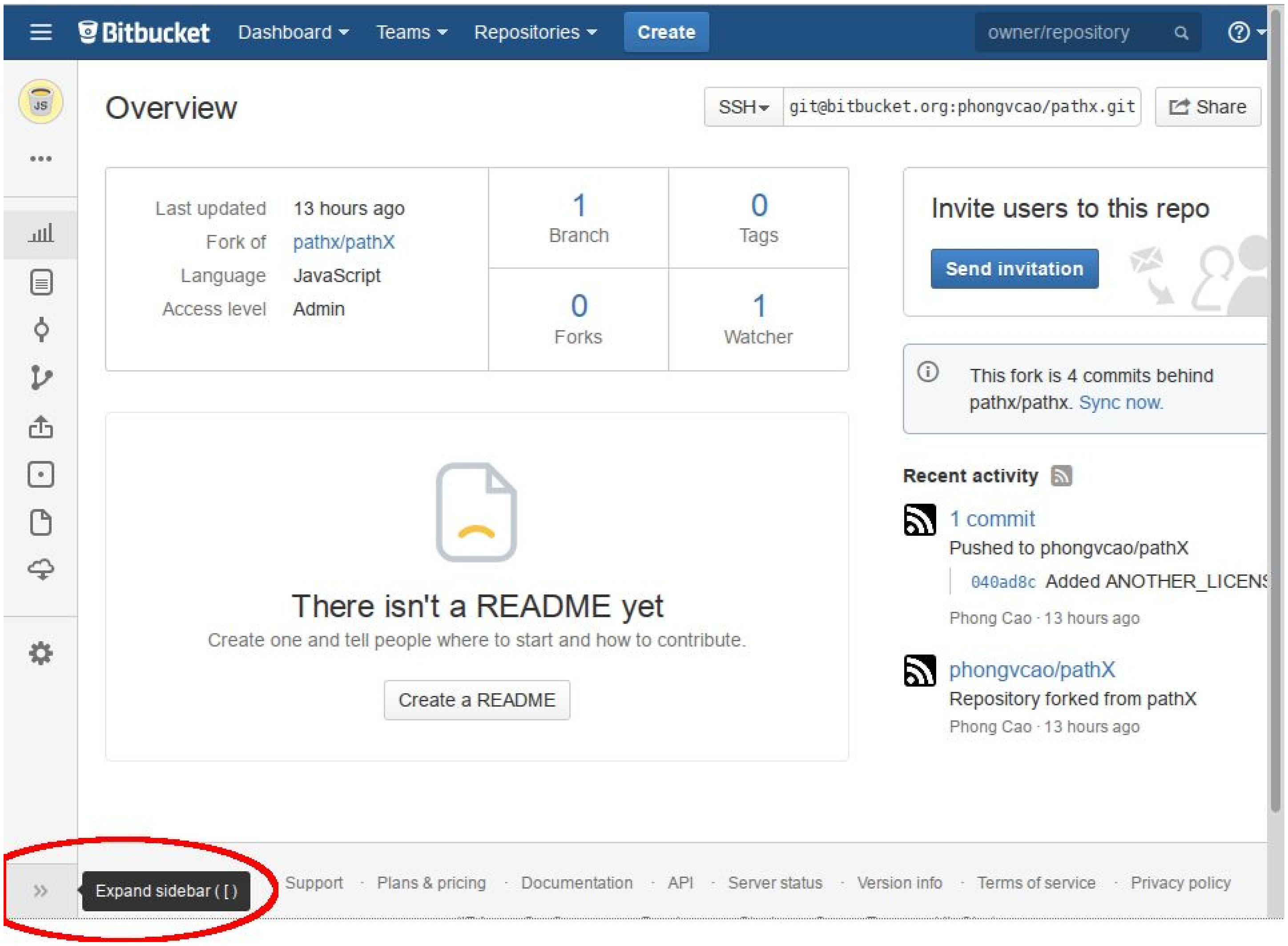
1 commit
Pushed to pathx/pathX
d7763ea Removed junk files
Phong Cao · 9 hours ago

2 commits
Pushed to pathx/pathX
e6e0b08 Merge branch 'master' of b
3bb1402 - First commit
Phong Cao · 9 hours ago

phongvcao/eBirchary
Repository forked from eBirchary
Phong Cao · 9 hours ago

Michael Downer
Access changed to admin for eBirchar
Phong Cao · 9 hours ago

2. On your pathX fork repository Overview page, expand the left sidebar. Click on the item called “Settings” to access the settings of your pathX fork repository:


Bitbucket
Dashboard
Teams
Repositories
Create
owner/repository
Overview
SSH
git@bitbucket.org:phongvcao/pathx.git
Share
Invite users to this repo
Send invitation
There isn't a README yet
Create one and tell people where to start and how to contribute.
Create a README
Expand sidebar ([ )
Support · Plans & pricing · Documentation · API · Server status · Version info · Terms of service · Privacy policy

Last updated	13 hours ago
Fork of	pathx/pathX
Language	JavaScript
Access level	Admin

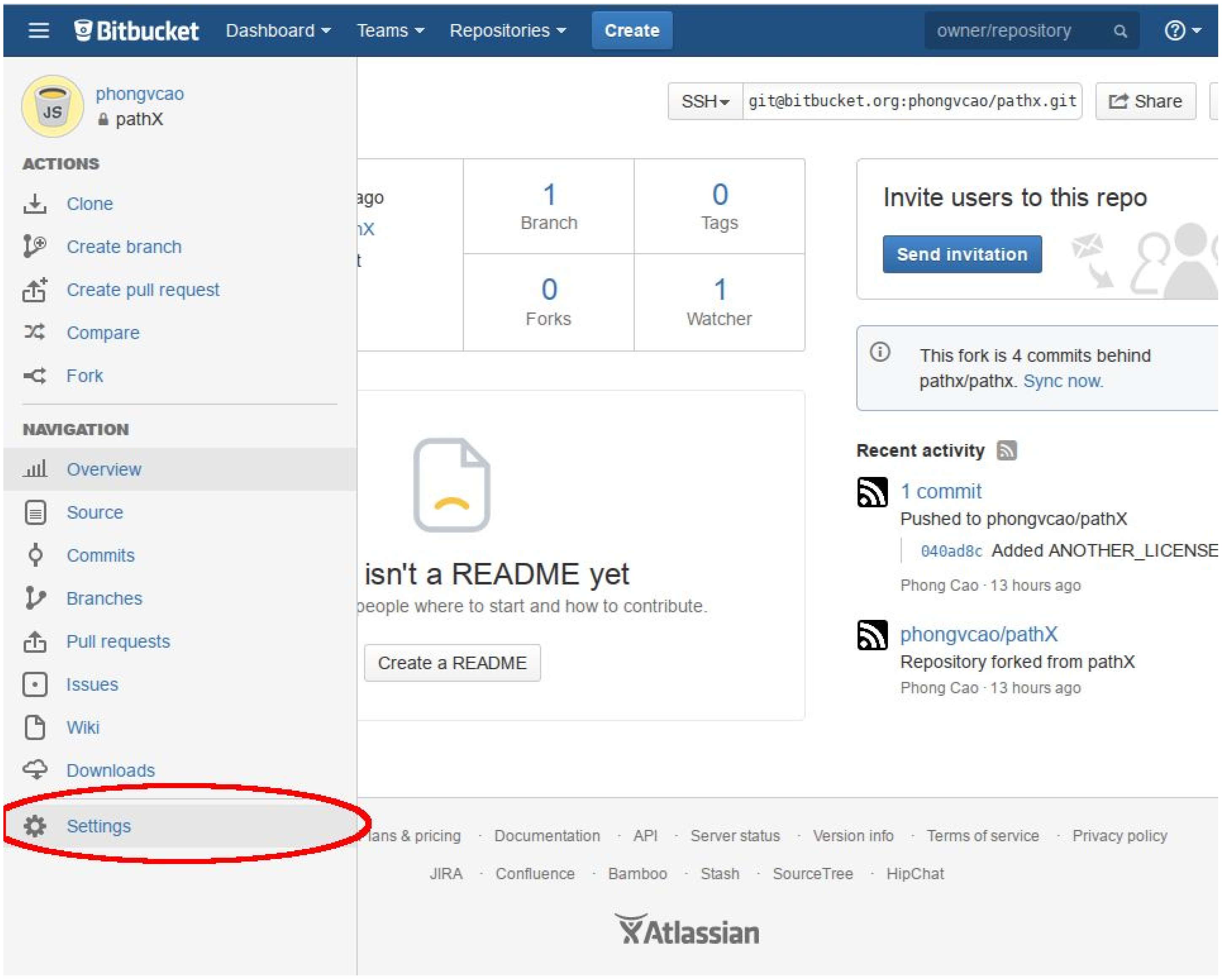
1 Branch	0 Tags
0 Forks	1 Watcher

This fork is 4 commits behind pathx/pathx. Sync now.

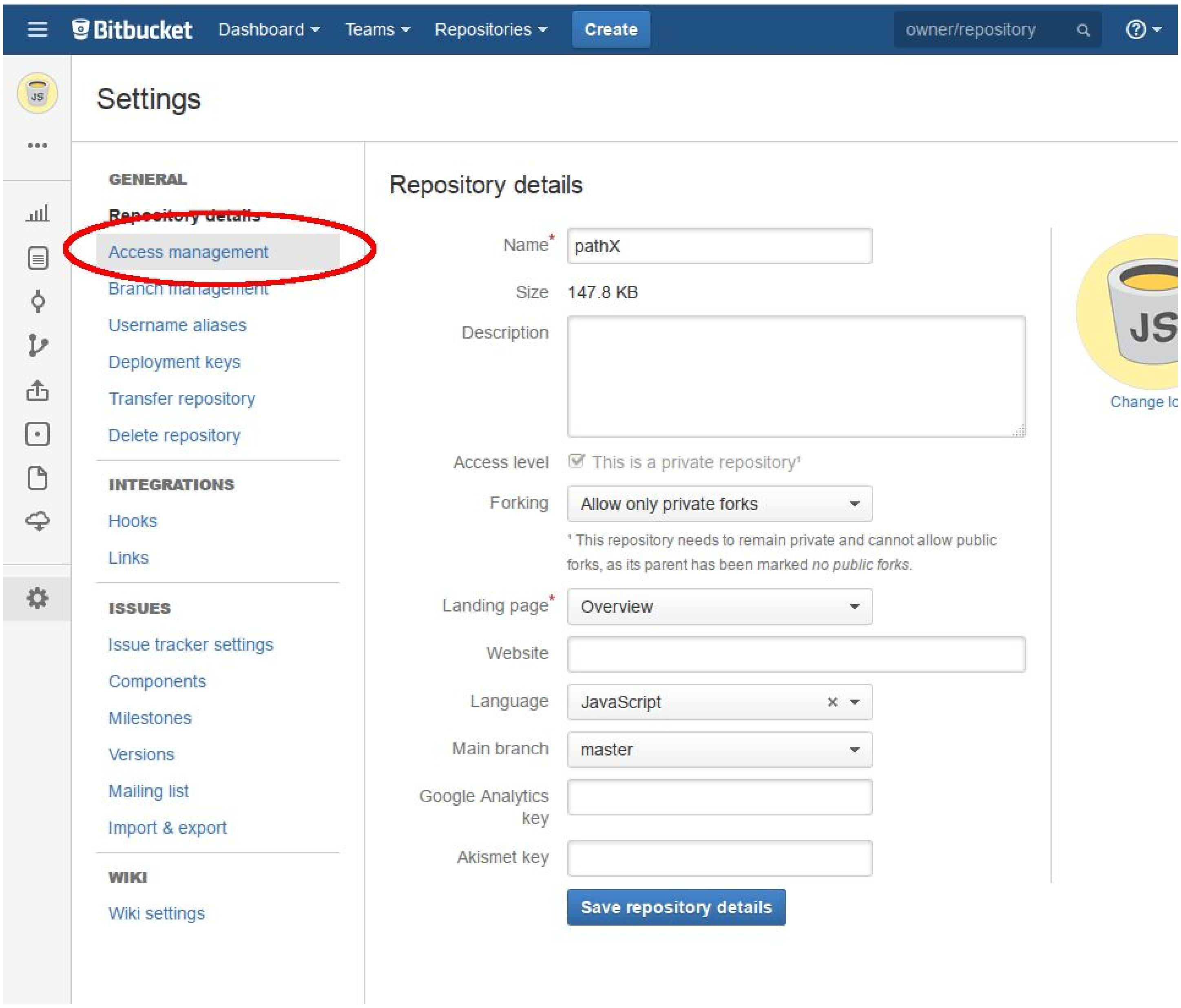
Recent activity

1 commit
Pushed to phongvcao/pathX
040ad8c Added ANOTHER_LICENS
Phong Cao · 13 hours ago

phongvcao/pathX
Repository forked from pathX
Phong Cao · 13 hours ago


Bitbucket
Dashboard
Teams
Repositories
Create
owner/repository
phongvcao
pathX
ACTIONS
Clone
Create branch
Create pull request
Compare
Fork
NAVIGATION
Overview
Source
Commits
Branches
Pull requests
Issues
Wiki
Downloads
Settings
SSH
git@bitbucket.org:phongvcao/pathx.git
Share
1 Branch
0 Tags
0 Forks
1 Watcher
Invite users to this repo
Send invitation
This fork is 4 commits behind pathx/pathx. Sync now.
isn't a README yet
people where to start and how to contribute.
Create a README
Recent activity
1 commit
Pushed to phongvcao/pathX
040ad8c Added ANOTHER_LICENSE
Phong Cao · 13 hours ago
phongvcao/pathX
Repository forked from pathX
Phong Cao · 13 hours ago
lans & pricing · Documentation · API · Server status · Version info · Terms of service · Privacy policy
JIRA · Confluence · Bamboo · Stash · SourceTree · HipChat
Atlassian

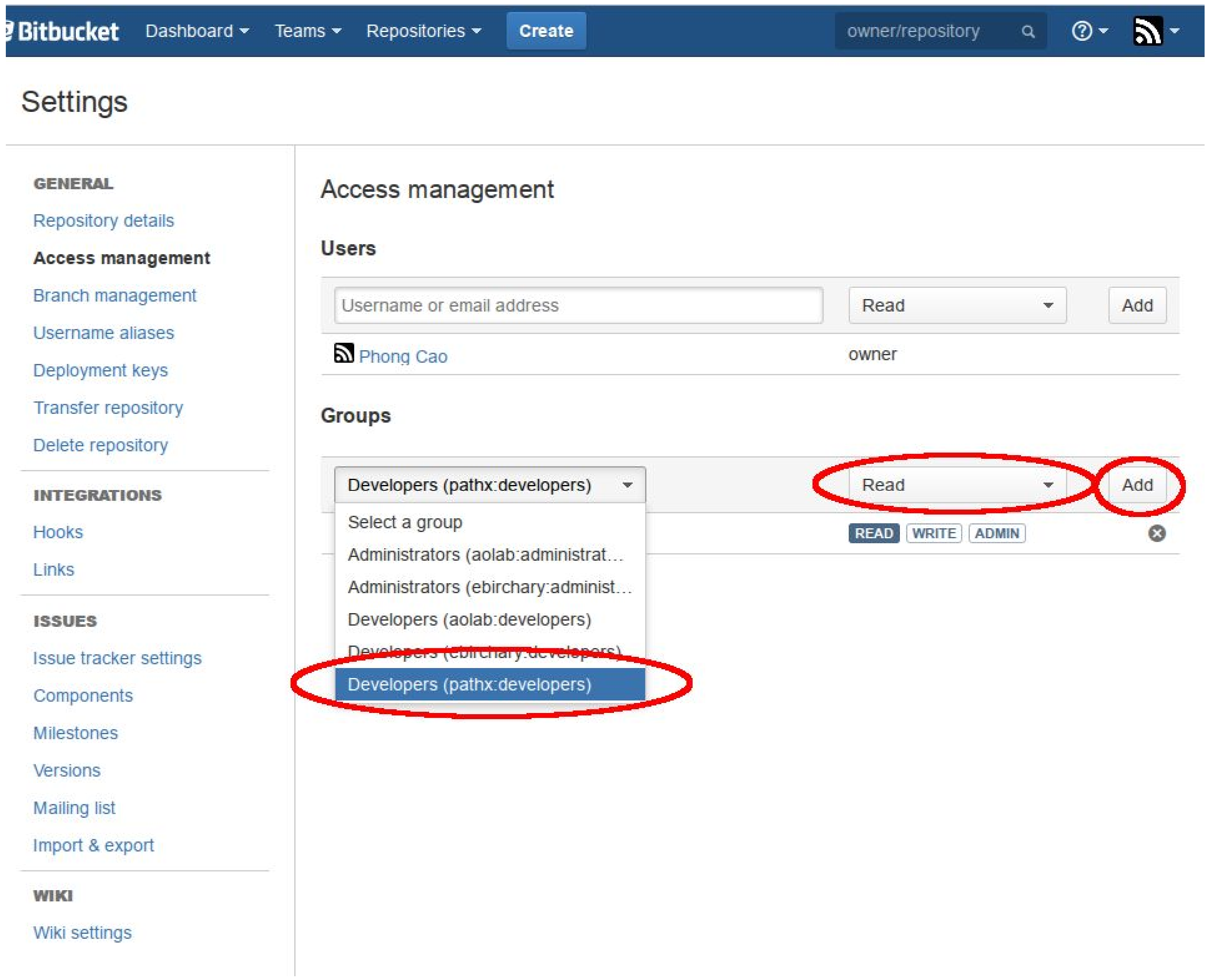
3. On your pathX fork repository settings page, click on “Access Management” to access the Access Management page:



Repository details

Name*	pathX
Size	147.8 KB
Description	
Access level	✓ This is a private repository¹
Forking	Allow only private forks
	¹ This repository needs to remain private and cannot allow public forks, as its parent has been marked *no public forks*.
Landing page*	Overview
Website	
Language	JavaScript
Main branch	master
Google Analytics key	
Akismet key	

4. When you're on the Access Management page, click on the drop-down menu in the "Groups" section and choose "Administrators (pathx:administrators)" then click "Add". Do the same for "Developers (pathx:developers)" group in the drop-down menu. The final result is in the second screenshot following:



Settings

GENERAL

Repository details

Access management

Branch management

Username aliases

Deployment keys

Transfer repository

Delete repository

INTEGRATIONS

Hooks

Links

ISSUES

Issue tracker settings

Components

Milestones

Versions

Mailing list

Import & export

WIKI

Wiki settings

Access management

Users

Username or email address | Read | Add

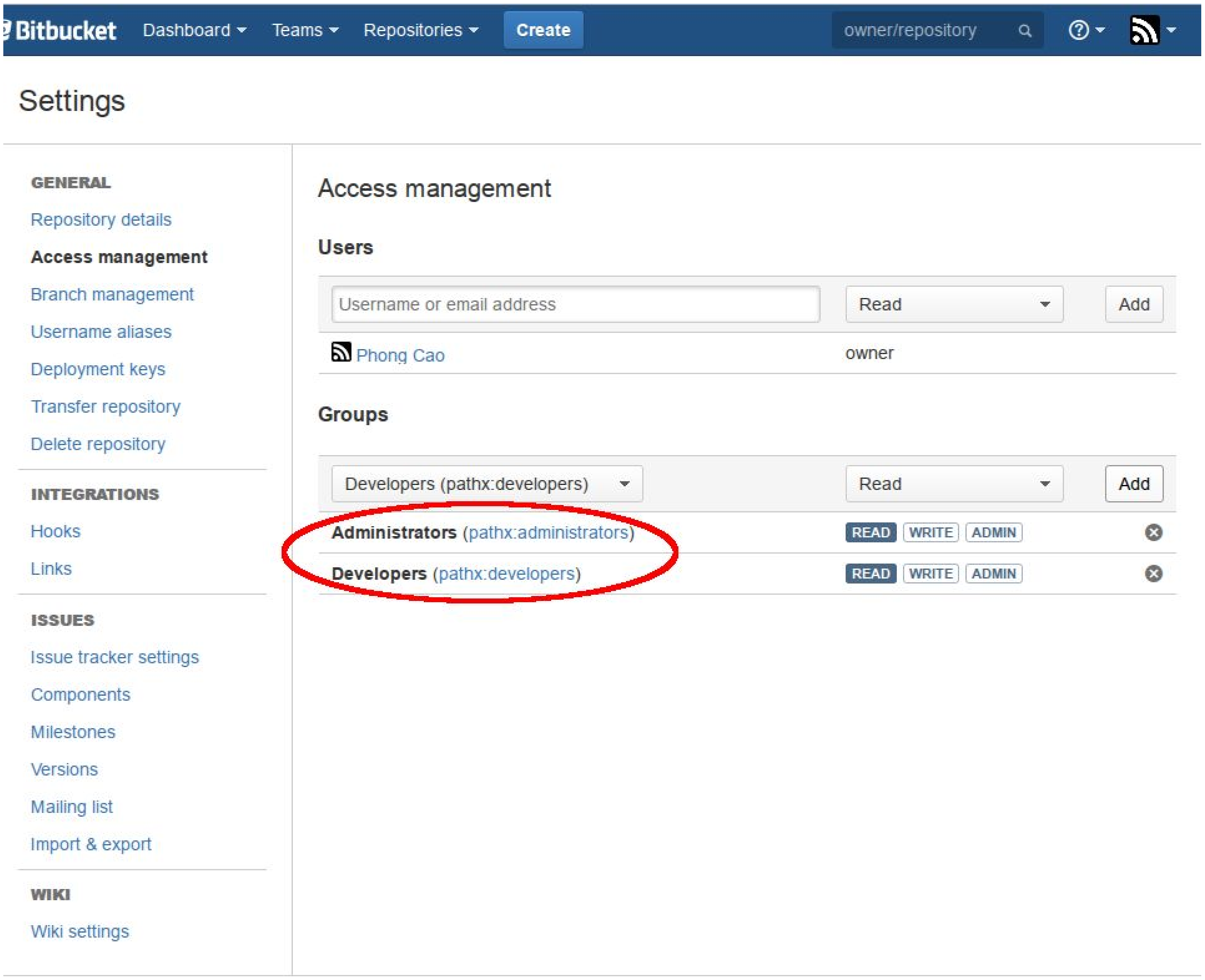
Phong Cao | owner

Groups

Developers (pathx:developers) | Read | Add

READ WRITE ADMIN

Select a group
Administrators (aolab:administrat…
Administrators (ebirchary:administ…
Developers (aolab:developers)
Developers (ebirchary:developers)
Developers (pathx:developers)



Settings

GENERAL
Repository details
Access management
Branch management
Username aliases
Deployment keys
Transfer repository
Delete repository

INTEGRATIONS
Hooks
Links

ISSUES
Issue tracker settings
Components
Milestones
Versions
Mailing list
Import & export

WIKI
Wiki settings

Access management

Users

Username or email address	Read	Add
Phong Cao	owner	

Groups

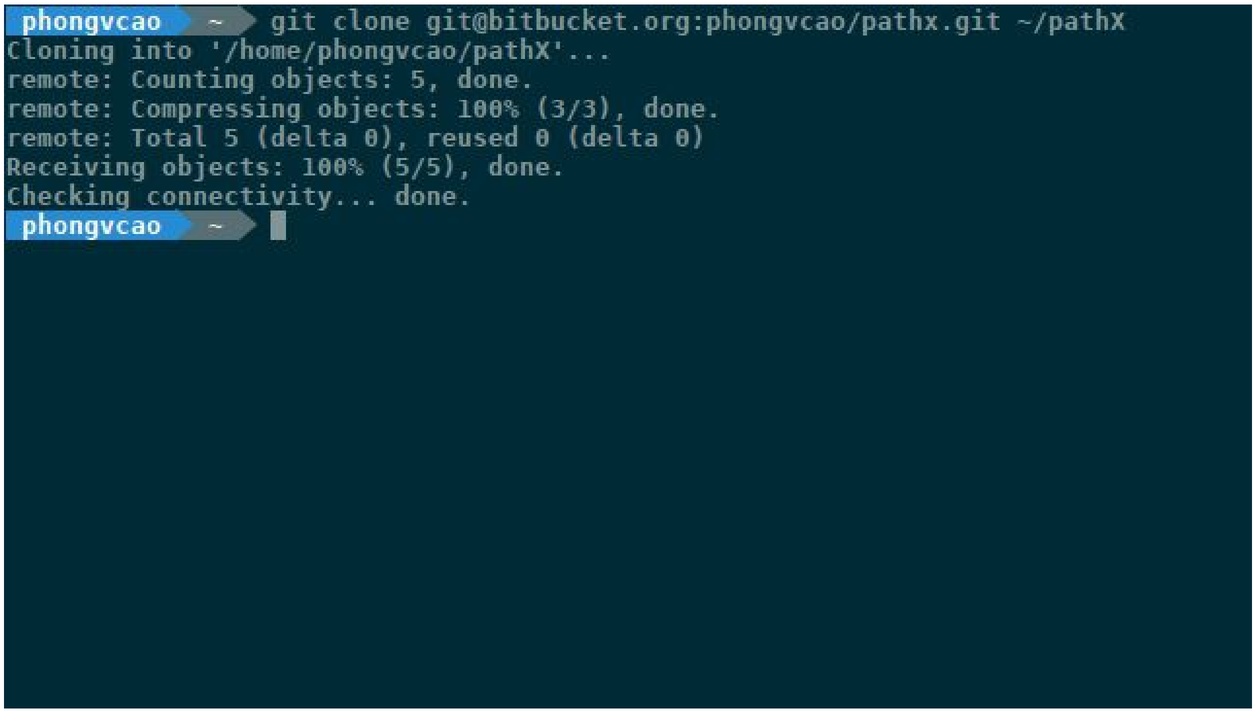
Developers (pathx:developers)	Read	Add
Administrators (pathx:administrators)	READ WRITE ADMIN	
Developers (pathx:developers)	READ WRITE ADMIN	

Step 10: Clone PathX from your fork to your computer

1. Now that you have copied the SSH link, open Git Bash and issue the following command:

```
$ git clone git@bitbucket.org:<your_Bitbucket_username>/pathx.git ~/pathX
```

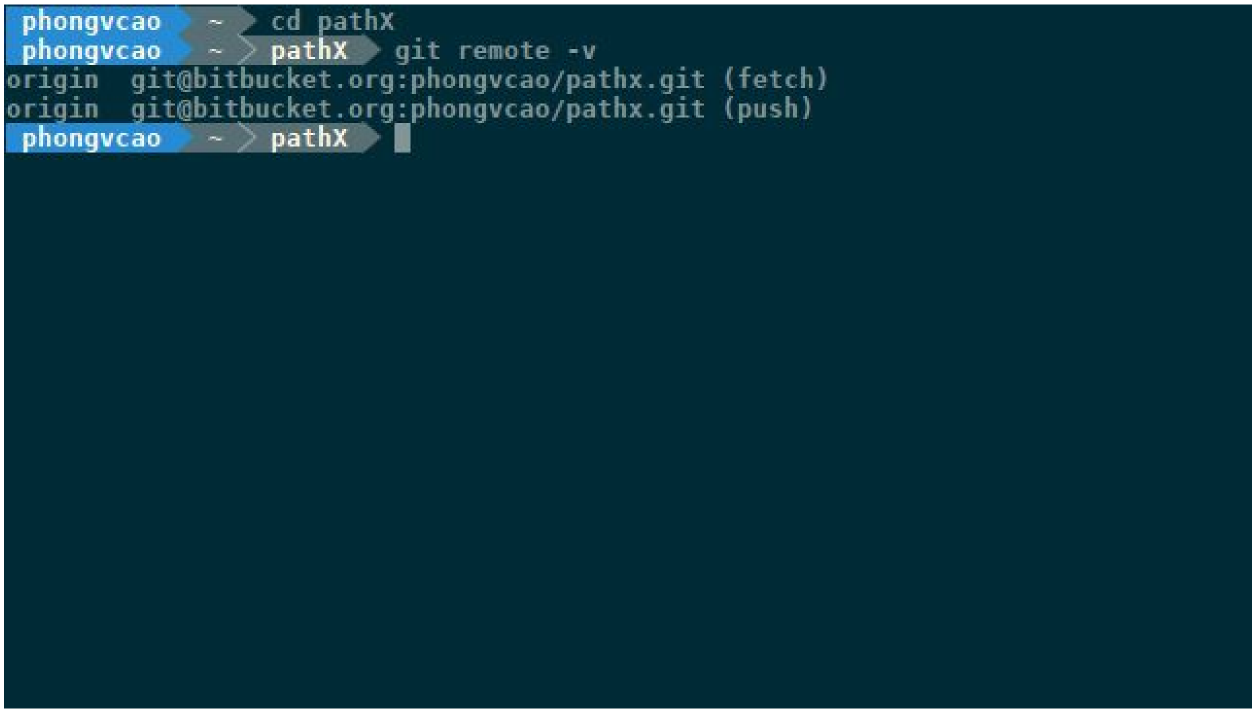
This command will clone (or download, in other words) the pathX fork from your account to a folder called "pathX" under your home folder.
+ If you're using Windows, the path to folder is `C:\Users\<your_Windows_username>\pathX`.
+ If you're using Mac OSX or Linux, the path to folder is `/home/<your_username>/pathX or simply ~/pathX`.



```
 phongvcao  ~  git clone git@bitbucket.org:phongvcao/pathx.git ~/pathX
Cloning into '/home/phongvcao/pathX'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (5/5), done.
Checking connectivity... done.
 phongvcao  ~ 
```

2. After the cloning process is finished, you can check to see if the Git remote repo of your local pathX match with the SSH link of the pathX fork on your Bitbucket account by “cd-ing” into the project’s folder (which means getting inside the project folder using the command-line) and issuing the following command:

```
$ git remote -v
```



```
 phongvcao  ~  cd pathX
 phongvcao  ~  pathX  git remote -v
origin  git@bitbucket.org:phongvcao/pathx.git (fetch)
origin  git@bitbucket.org:phongvcao/pathx.git (push)
 phongvcao  ~  pathX 
```

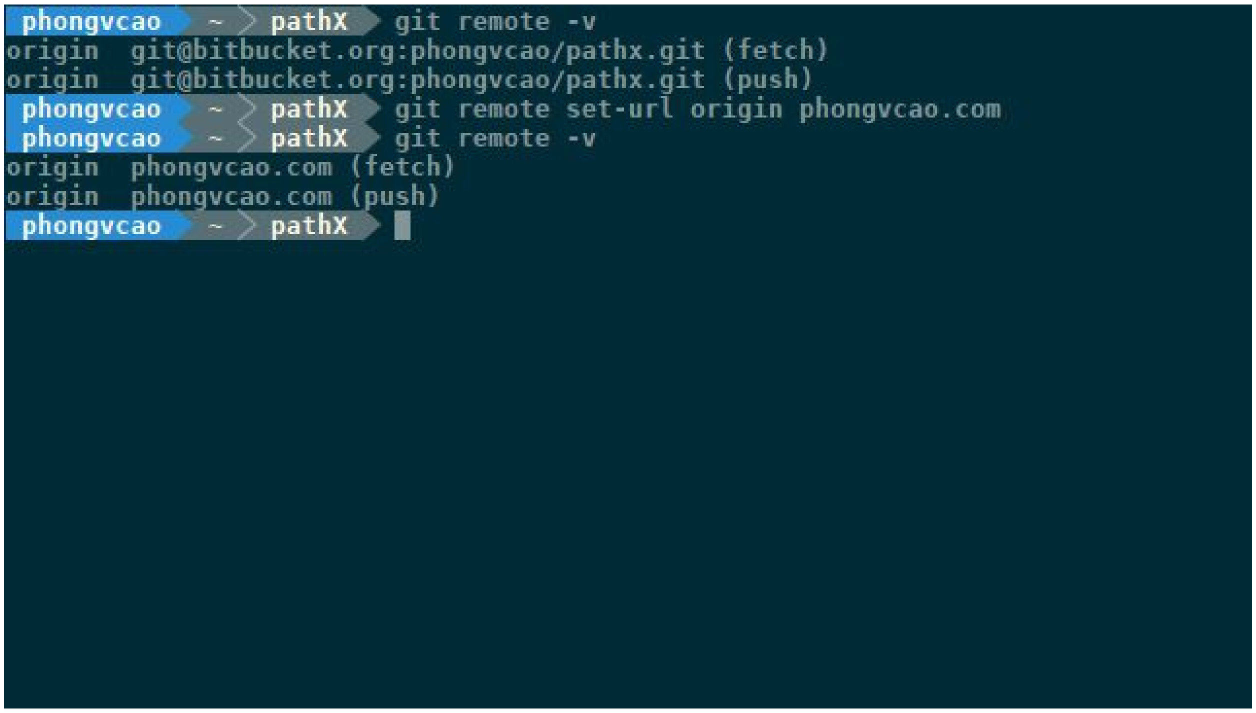
3. So in the above screenshot, I got my remote addresses correctly. The remotes, by default, are called “origin” and their addresses should match the SSH addresses of your pathX fork.

As you can see, after the URL of each remote there is a parentheses indicates what kind of remote it is. In this case, “(fetch)” remote means that when you do a Git pull and a Git fetch (you’ll see it later), the source code will be “fetched” from this URL to your local computer. When you do a Git push (which means uploading your code to your Bitbucket pathX fork), the “(push)” remote’s URL will be where your code is gonna get uploaded.

Step 11: Update, Add and Delete Remotes

1. In order to update the URL of a Git Remote, issue the following command:

```
$ git remote set-url <remote_name> <new_remote_url>
```

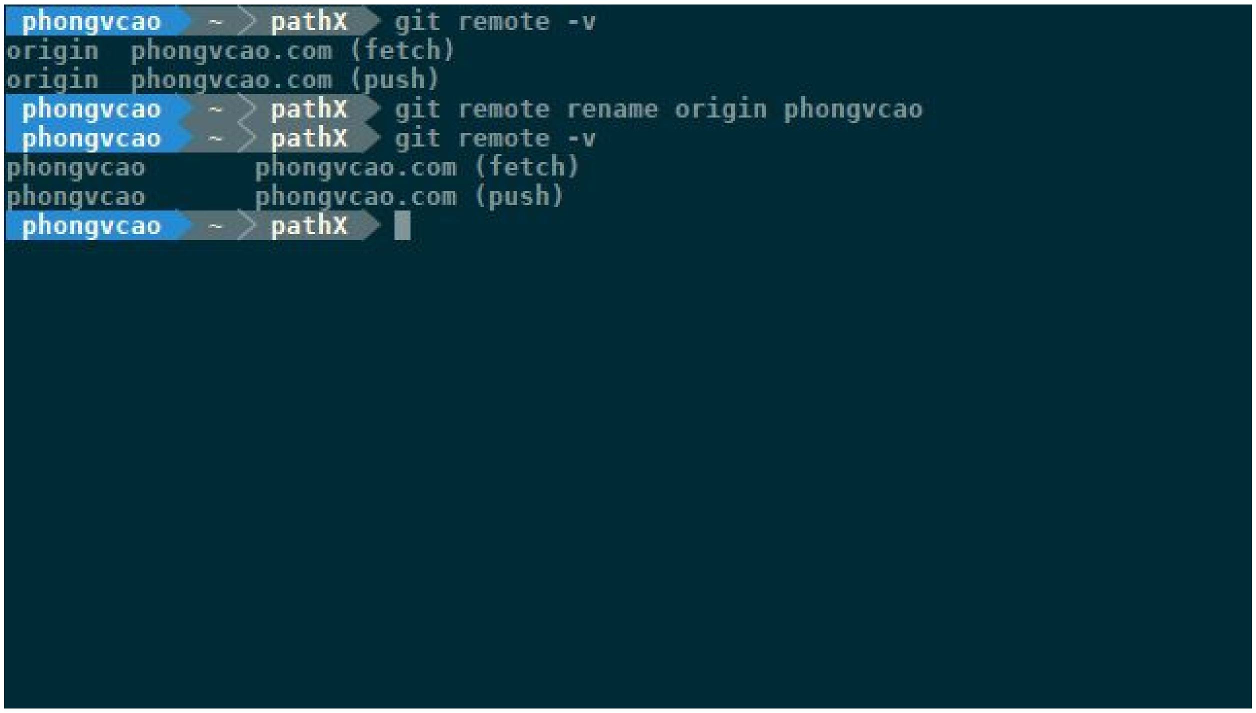


```
phongvcao  ~  pathX  git remote -v
origin  git@bitbucket.org:phongvcao/pathx.git (fetch)
origin  git@bitbucket.org:phongvcao/pathx.git (push)
phongvcao  ~  pathX  git remote set-url origin phongvcao.com
phongvcao  ~  pathX  git remote -v
origin  phongvcao.com (fetch)
origin  phongvcao.com (push)
phongvcao  ~  pathX
```

As you can see in the above screenshot, I changed the URL of the “origin” remote from `git@bitbucket.org:phongvcao/pathx.git` to phongvcao.com

2. In order to update the Name of a Git Remote, issue the following command:

```
$ git remote rename <old_remote_name> <new_remote_name>
```

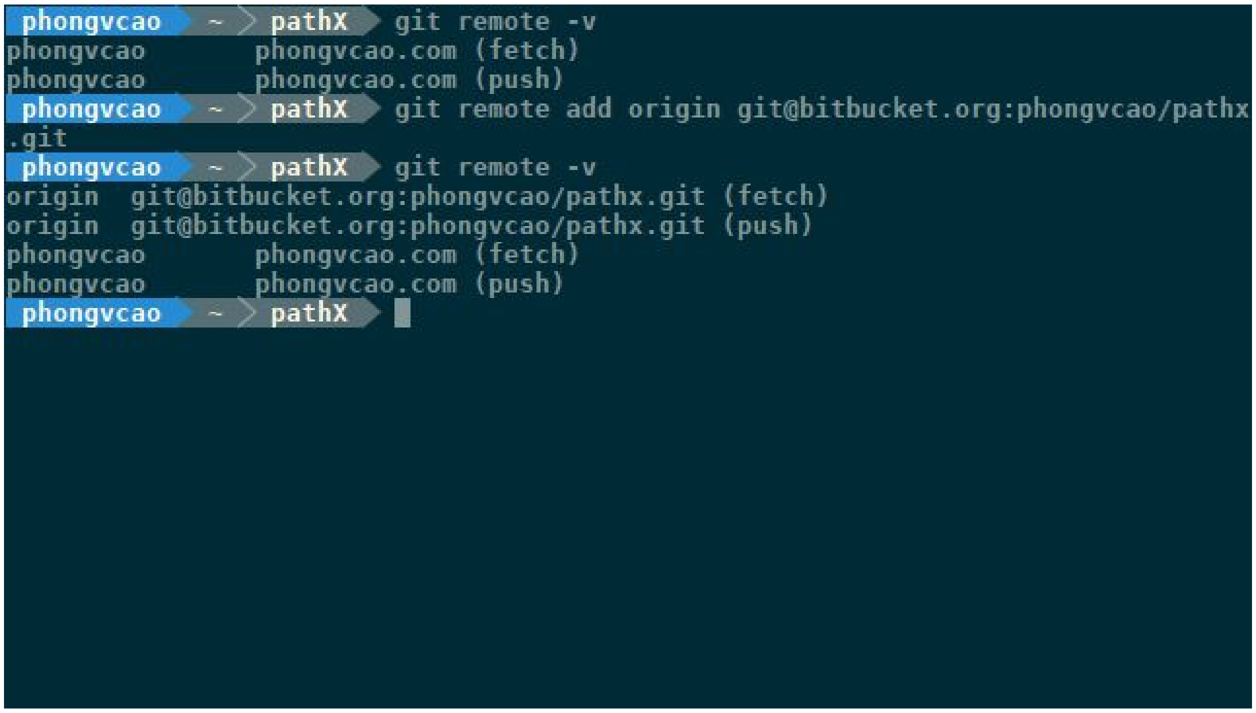


```
phongvcao  ~  pathX  git remote -v
origin  phongvcao.com (fetch)
origin  phongvcao.com (push)
phongvcao  ~  pathX  git remote rename origin phongvcao
phongvcao  ~  pathX  git remote -v
phongvcao       phongvcao.com (fetch)
phongvcao       phongvcao.com (push)
phongvcao  ~  pathX
```

As you can see in the above screenshot, I changed my “origin” remote’s name from “origin” to “phongvcao”.

3. In order to add a remote, issue the following command:

```
$ git remote add <remote_name> <remote_url>
```

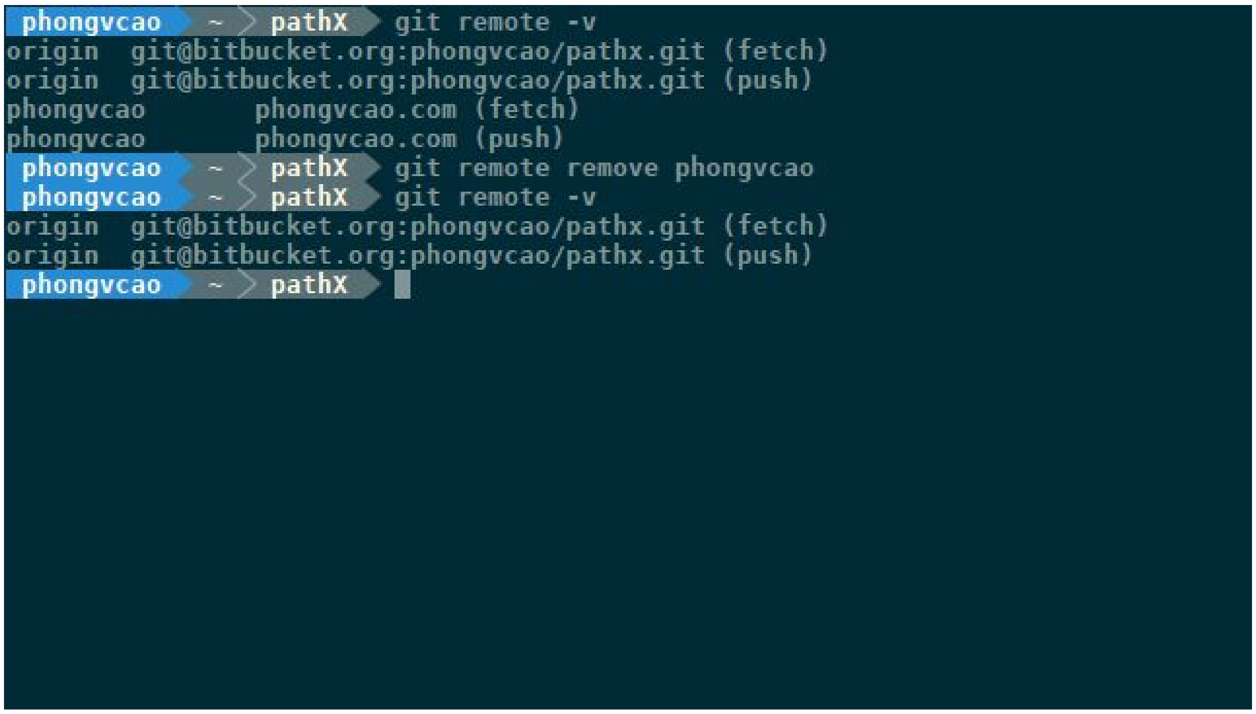


```
phongvcao  ~  pathX  git remote -v
phongvcao       phongvcao.com (fetch)
phongvcao       phongvcao.com (push)
phongvcao  ~  pathX  git remote add origin git@bitbucket.org:phongvcao/pathx
.git
phongvcao  ~  pathX  git remote -v
origin  git@bitbucket.org:phongvcao/pathx.git (fetch)
origin  git@bitbucket.org:phongvcao/pathx.git (push)
phongvcao       phongvcao.com (fetch)
phongvcao       phongvcao.com (push)
phongvcao  ~  pathX
```

As you can see in the above screenshot, I added a remote named “`origin`” (`git@bitbucket.org:phongvcao/pathx.git`) into my pathX’s Git repo.

4. In order to remove a remote, issue the following command:

```
$ git remote remove <remote_name>
```



```
phongvcao  ~  pathX  git remote -v
origin  git@bitbucket.org:phongvcao/pathx.git (fetch)
origin  git@bitbucket.org:phongvcao/pathx.git (push)
phongvcao       phongvcao.com (fetch)
phongvcao       phongvcao.com (push)
phongvcao  ~  pathX  git remote remove phongvcao
phongvcao  ~  pathX  git remote -v
origin  git@bitbucket.org:phongvcao/pathx.git (fetch)
origin  git@bitbucket.org:phongvcao/pathx.git (push)
phongvcao  ~  pathX
```

As you can see in the above screenshot, I removed the remote named "`phongvcao`" while keeping the "`origin`" remote.

Step 12: Git Commit & Git Push Basics

One of the most important basic Git concept that one must understand is "Git commit". So what is a "Git commit"?

A "Git commit" is basically a snapshot, a "photo" of your whole repository at a certain point in time. A "Git commit" lets you make some changes to the project and if things don't work out right, you can always revert back to the original, un-modified version of the project (or in other words, switch to an older, more "stable", more "original" Git commit snapshot). It is the way a Version Control System (VCS) like Git works. Instead of having to back-up, copy-paste all of your project files everytime you make new changes, you can simply issue a command and revert all of your changes. How convenient is it?

However, before you can issue a "Git commit", you have to add in the project files that have been changed, just like taking a photo with a camera. When you want to take a photo, you have to aim your camera at the object you want to shoot (in other words, you have to "bring" or "add" the object you want to shoot into the camera's view) and press the "Release" button.

The same goes for Git. Before issuing a “Git commit”, you have to issue a “Git add” to add in the files that have been changed since the last (most recent) “Git commit”. If there were no change in the local repository since your last “Git commit”, then you will not be able to issue a new “Git commit”.

After issuing a “Git commit”, the changes you made to your local repository are saved locally on your computer. However, what if your computer is lost, or your hard drive fails? In those cases, all of the changes you made locally to your repository will be lost too, forever!

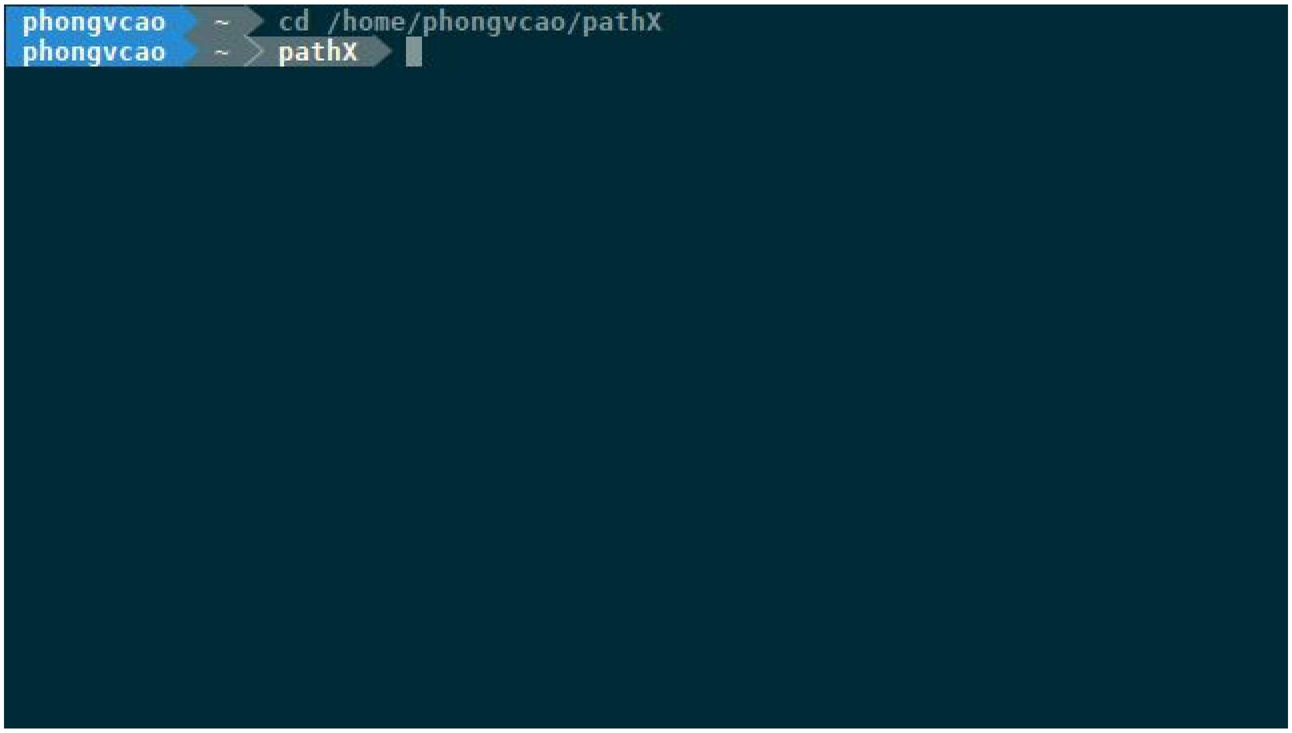
That is why “Git push” and “Git remote” were created. “Git push” lets you upload, or “push” your local Git repository to Git repository hosting site such as Bitbucket or Github, so that you can not only recover your files in case your computer is damaged, but also share your codes with others which allow you to easily collaborate with other programmers that live far away from you. By uploading your project online using “Git push”, you can work on your project repository on any computer by “Git cloning” the repository (or downloading your repository, in other words) and make some changes and some “Git commit”, and then push the newly-updated project up in the Bitbucket and Github cloud again for backing-up and sharing.

Step 13: Git Commit & Git Push Commands

Now that you’ve learned about Git Commit & Git Push Basics, let’s dive into the commands!

Note: Before being able to apply any of these commands, you must first issue “cd” into the project folder (or getting inside the project folder using command-line) or any of its sub-folder using the following command:

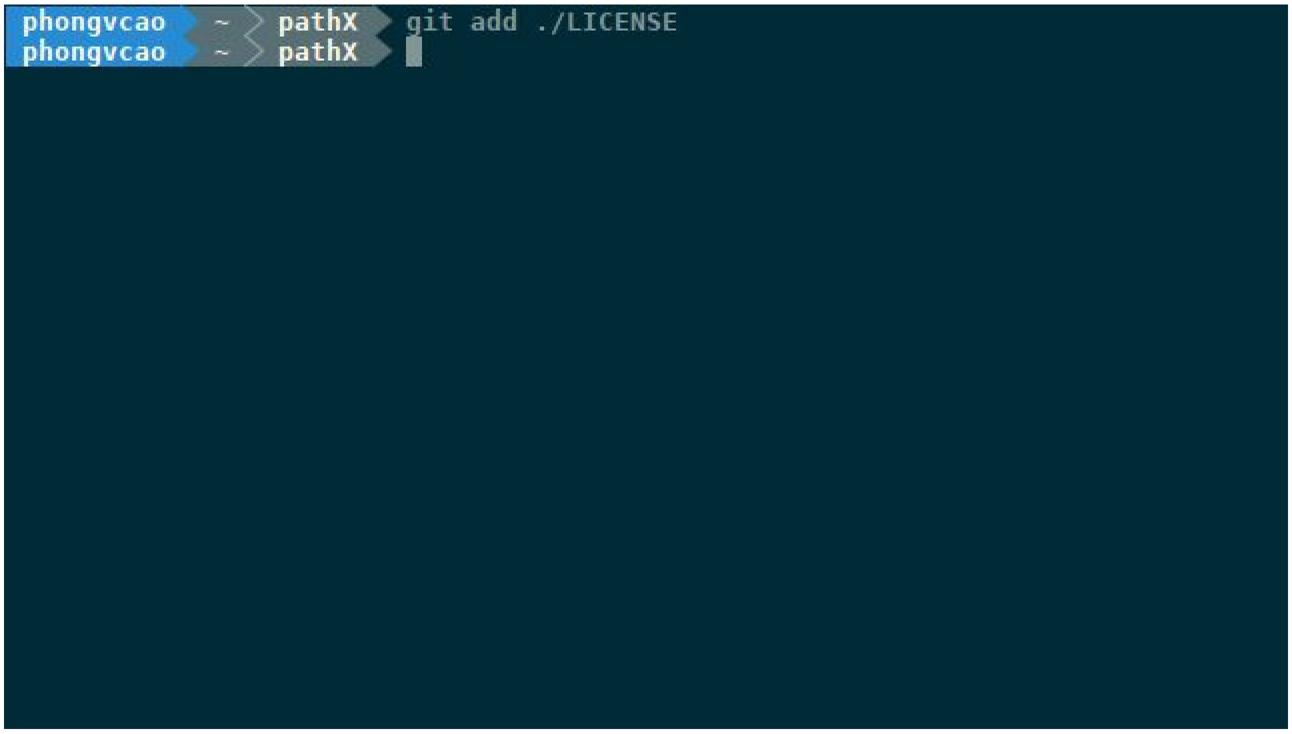
```
$ cd <path/to/project/folder>
```



```
phongvcao  ~  cd /home/phongvcao/pathX
phongvcao  ~  pathX
```

1. Add a changed file to a “Git commit”:

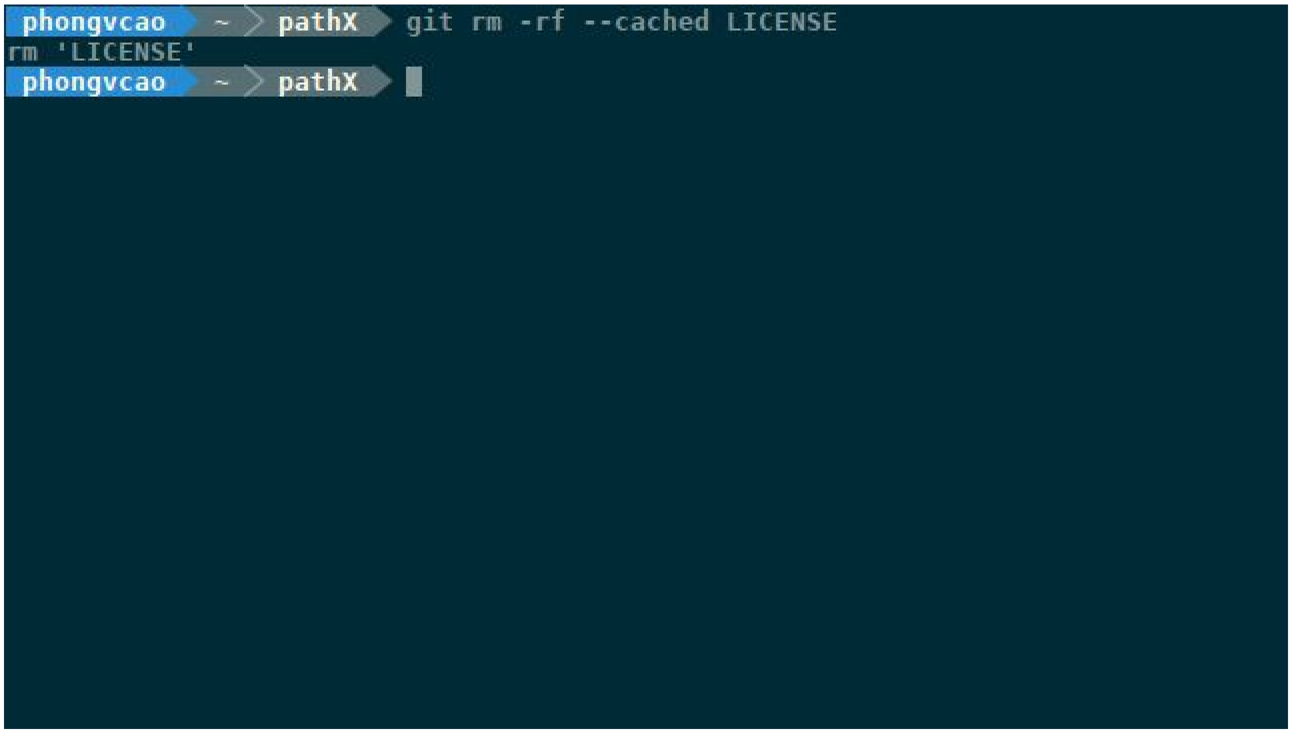
```
$ git add <path/to/file>
```



```
phongvcao  ~  pathX  git add ./LICENSE
phongvcao  ~  pathX
```

2. Remove a file (or a folder and all of its sub-folder) from a “Git commit” (without deleting it from the local filesystem):

```
$ git rm -rf --cached <path/to/file/or/folder>
```

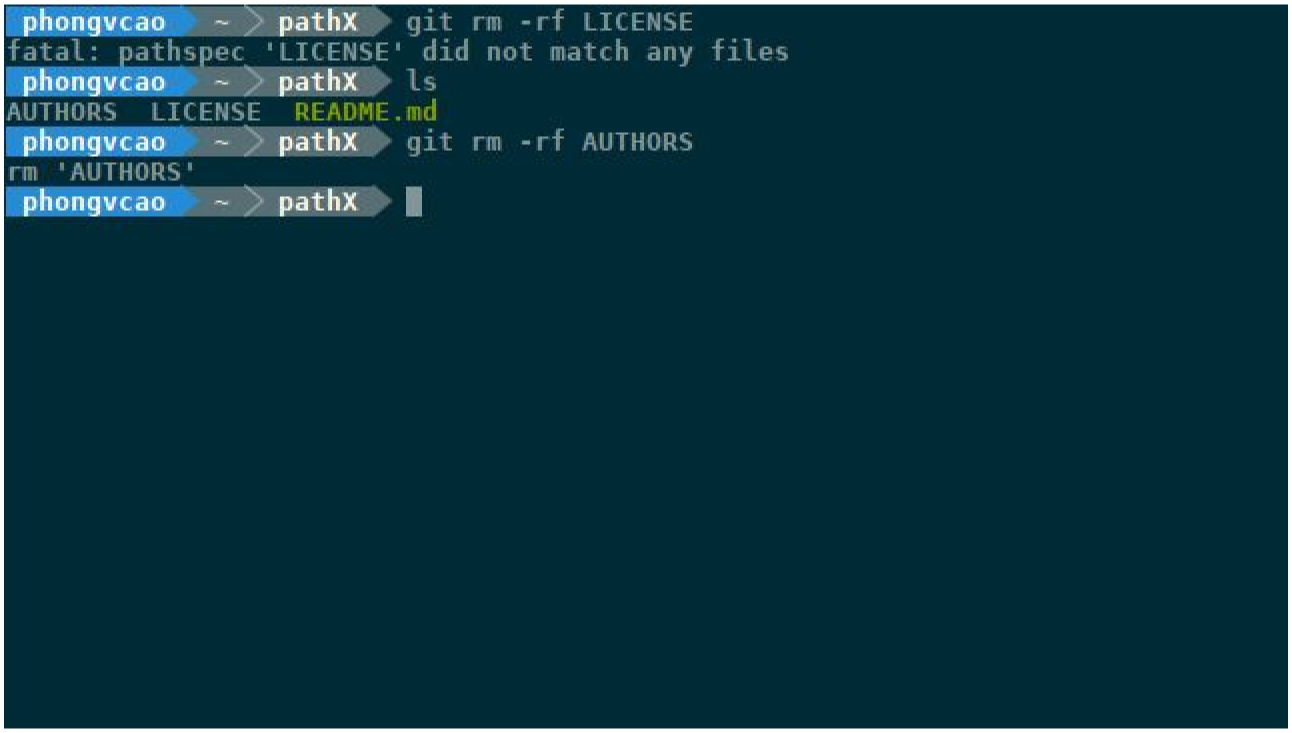


```
phongvcao  ~  pathX  git rm -rf --cached LICENSE
rm 'LICENSE'
phongvcao  ~  pathX
```

As you can see in the above screenshot, the file LICENSE was removed from the “Git commit” record, yet still existing in the local file system.

3. Remove a file (or a folder and all of its sub-folder) from a “Git commit” (and also delete it from the local filesystem):

```
$ git rm -rf <path/to/file/or/folder>
```

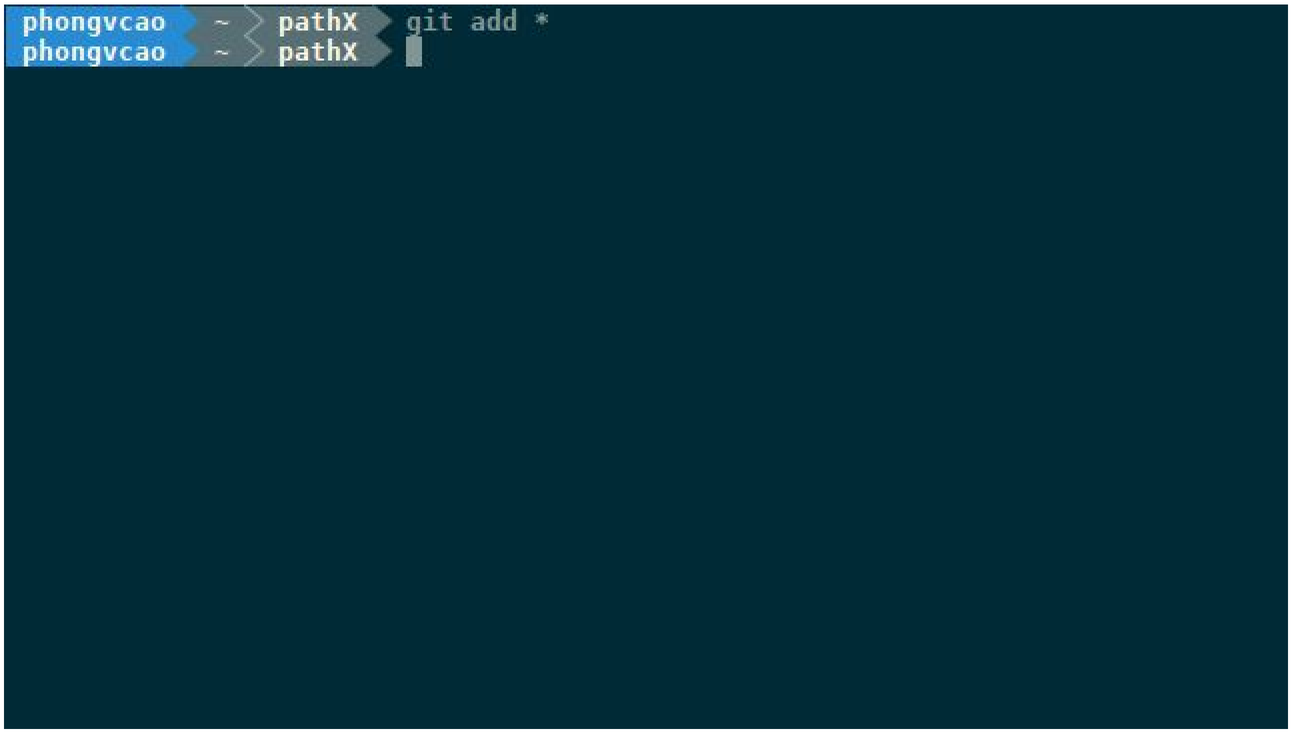


```
phongvcao  ~  pathX  git rm -rf LICENSE
fatal: pathspec 'LICENSE' did not match any files
phongvcao  ~  pathX  ls
AUTHORS  LICENSE  README.md
phongvcao  ~  pathX  git rm -rf AUTHORS
rm 'AUTHORS'
phongvcao  ~  pathX
```

As you can see from the above screenshot, my deletion of the LICENSE file was unsuccessful because the LICENSE file was deleted from the “Git commit” record in the previous step.

4. Add all files in the current project folder (or sub-folder, whatever folder your command-line client is currently at) to a “Git commit” (but still keep the deleted files):

```
$ git add *
```



```
phongvcao  ~  pathX  git add *
phongvcao  ~  pathX  
```

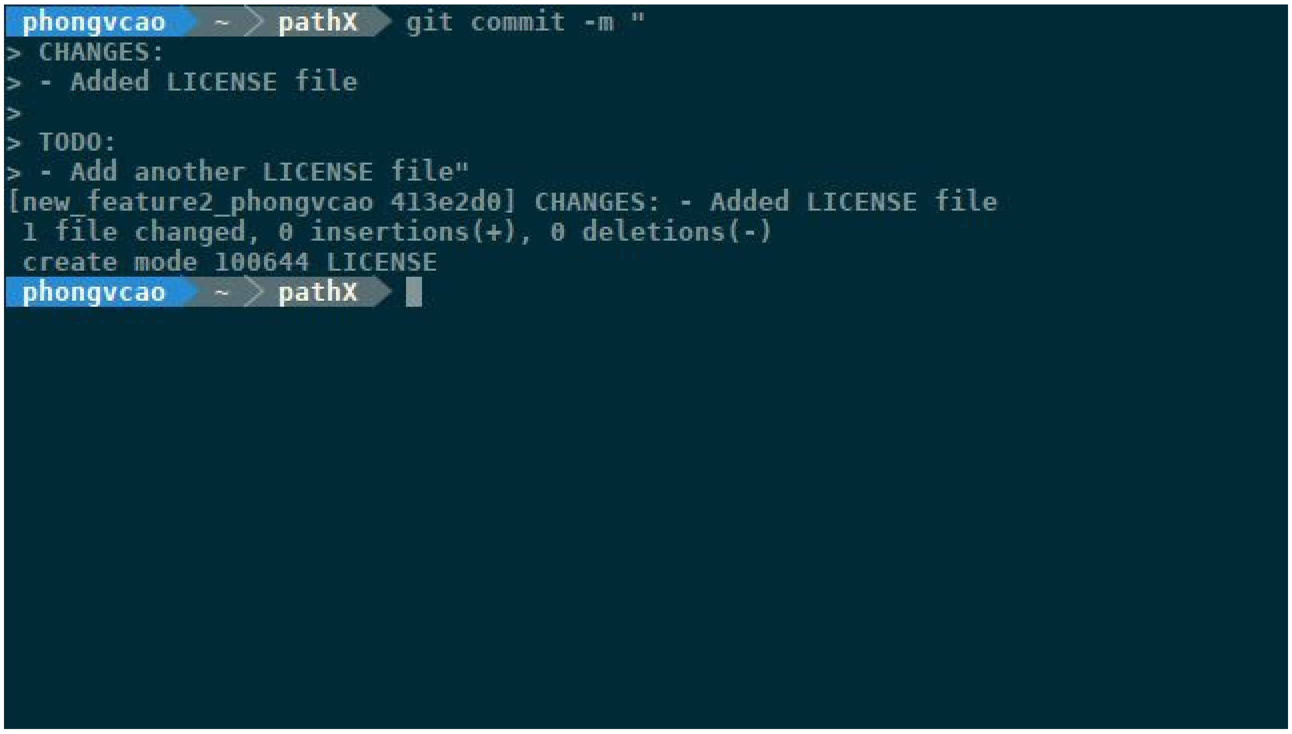
5. Add all files in the current project folder (or sub-folder, whatever folder your command-line client is currently at) to a “Git commit” (but record the deleted files):

```
$ git add -A
```

u

6. Create a “Git commit”:

```
$ git commit -m “Type your commit message here”
```



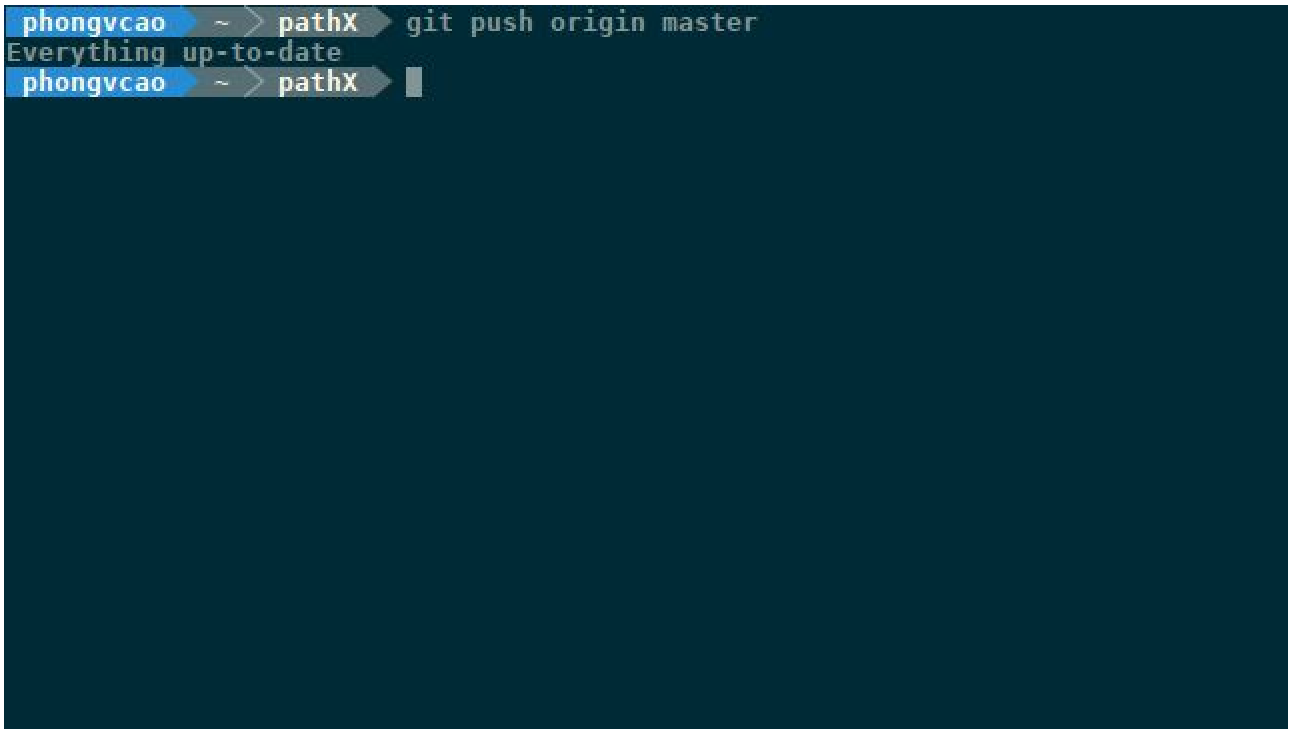
```
phongvcao  ~  pathX  git commit -m "
> CHANGES:
> - Added LICENSE file
>
> TODO:
> - Add another LICENSE file"
[new_feature2_phongvcao 413e2d0] CHANGES: - Added LICENSE file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 LICENSE
phongvcao  ~  pathX
```

The commit message typically contains the list of all of the changes you made in your project and what is still left to do.

To enter a newline like in the above screenshot, you have to press <Shift><Enter>.

7. Push the “Git commit” to Bitbucket:

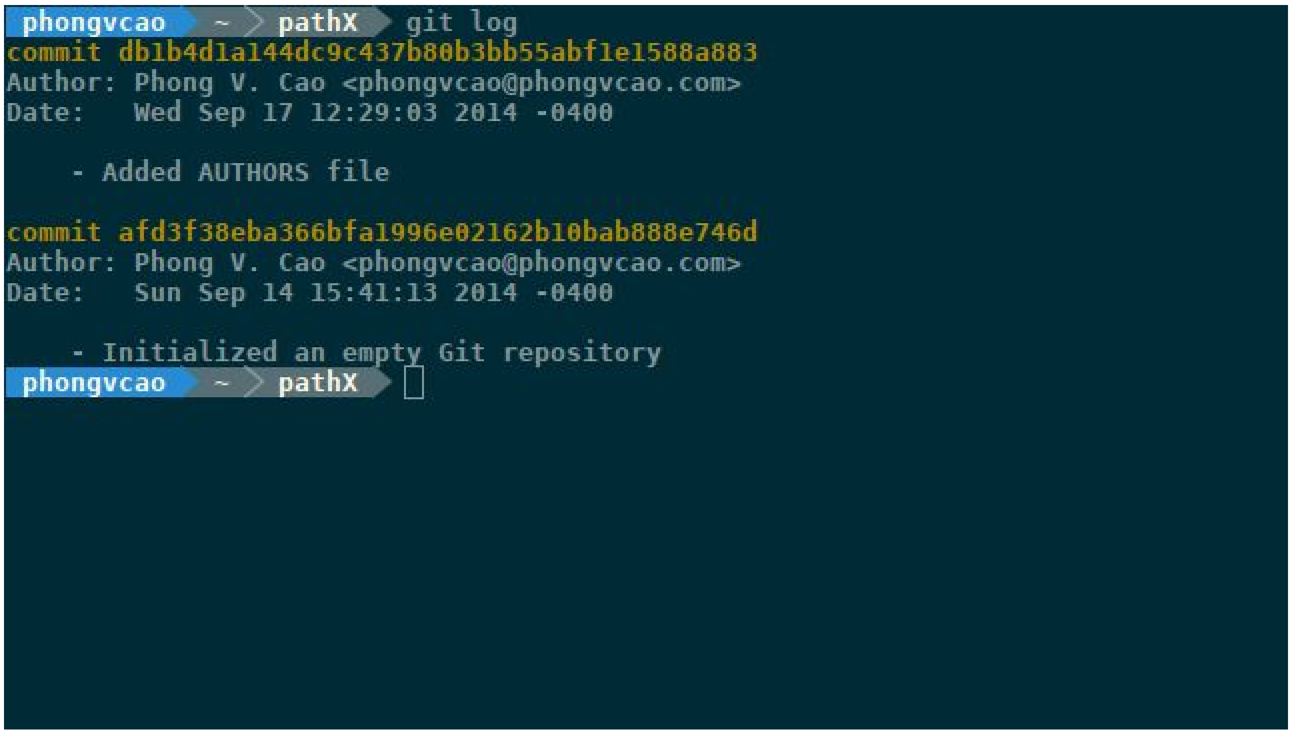
```
$ git push origin master
```



```
phongvcao  ~  pathX  git push origin master
Everything up-to-date
phongvcao  ~  pathX
```

8. List all the “Git commits” that you made:

```
$ git log
```



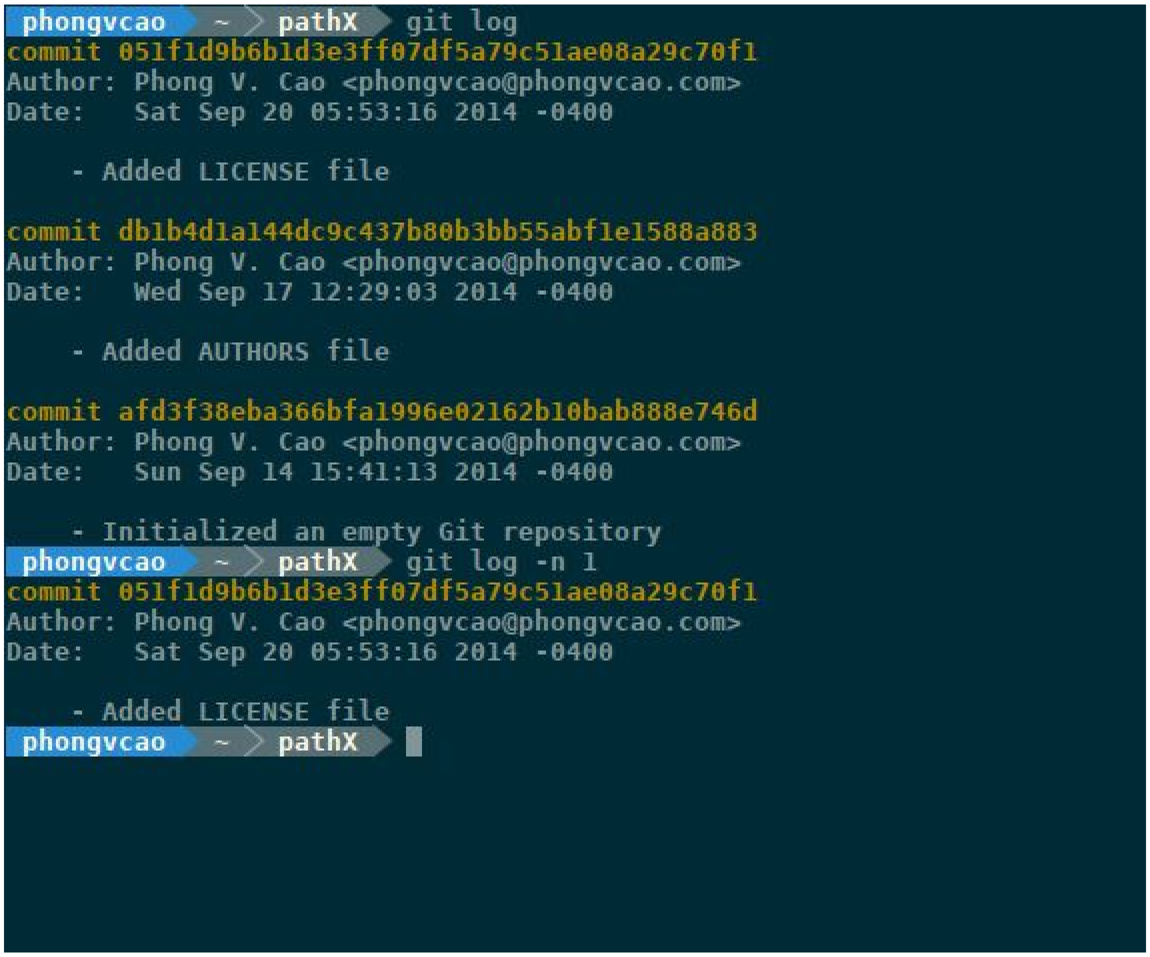
```
phongvcao  ~  pathX  git log
commit db1b4d1a144dc9c437b80b3bb55abf1e1588a883
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Wed Sep 17 12:29:03 2014 -0400

    - Added AUTHORS file

commit afd3f38eba366bfa1996e02162b10bab888e746d
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sun Sep 14 15:41:13 2014 -0400

    - Initialized an empty Git repository
phongvcao  ~  pathX  
```

See the Strings highlighted in Yellow? That is the Hash Code of the commit (The Hash Code of the commit is a VERY important and frequently-used thing, as you’ll see later).

The “Git commits” listed by “git log” are sorted in reversed chronological order, with the latest and most recent “Git commit” on top and the oldest one at the bottom.

9. View the current “Git commit” that you are at:

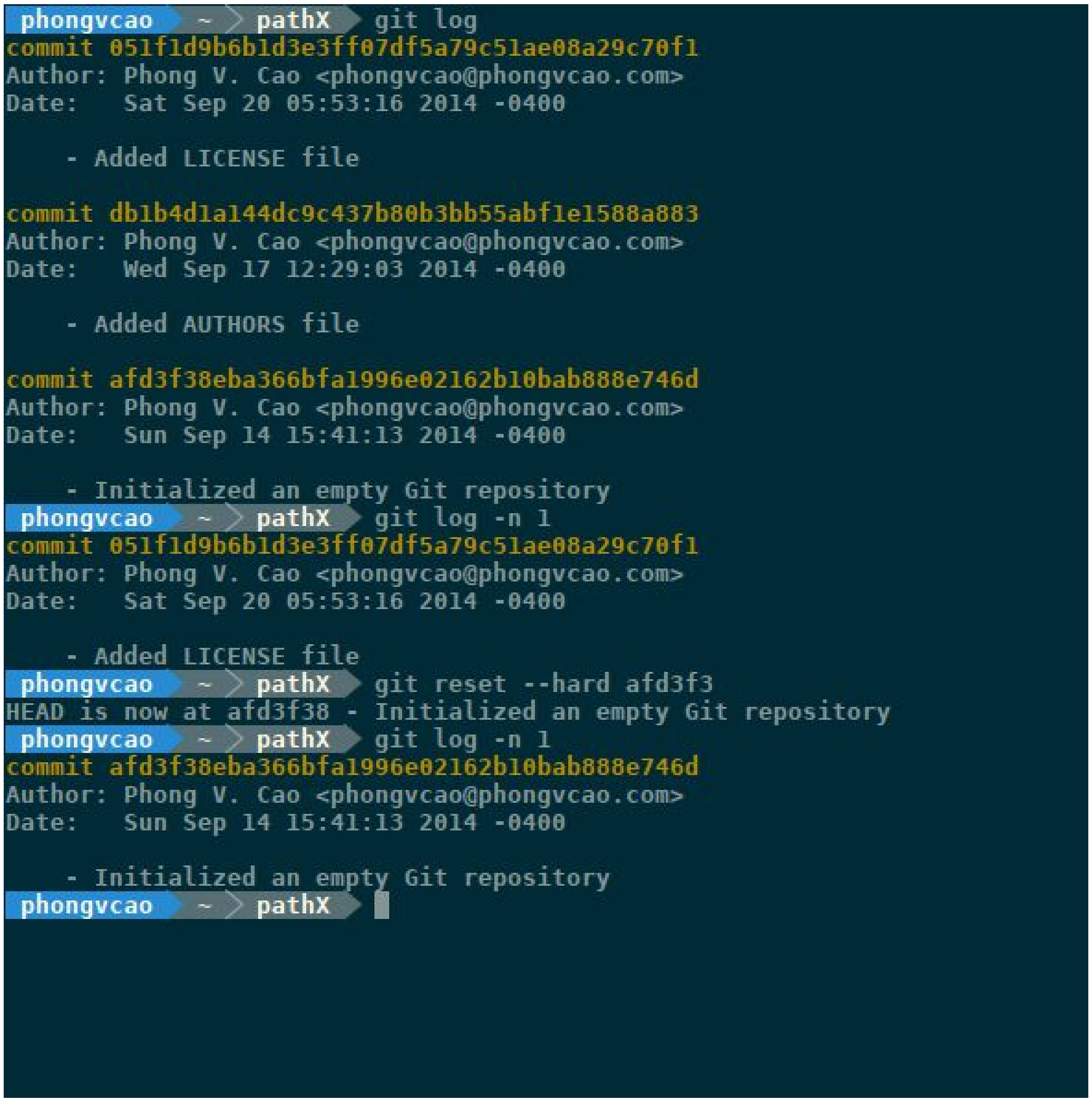
```
$ git log -n 1
```



```
phongvcao  ~  pathX  git log
commit 051f1d9b6b1d3e3ff07df5a79c51ae08a29c70f1
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sat Sep 20 05:53:16 2014 -0400

    - Added LICENSE file

commit db1b4d1a144dc9c437b80b3bb55abf1e1588a883
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Wed Sep 17 12:29:03 2014 -0400

    - Added AUTHORS file

commit afd3f38eba366bfa1996e02162b10bab888e746d
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sun Sep 14 15:41:13 2014 -0400

    - Initialized an empty Git repository
phongvcao  ~  pathX  git log -n 1
commit 051f1d9b6b1d3e3ff07df5a79c51ae08a29c70f1
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sat Sep 20 05:53:16 2014 -0400

    - Added LICENSE file
phongvcao  ~  pathX  
```

10. Switch between “Git commit”s:

```
$ git reset --hard <the_first_6_letters_of_the_Git_commit’s_Hash_Code>
```

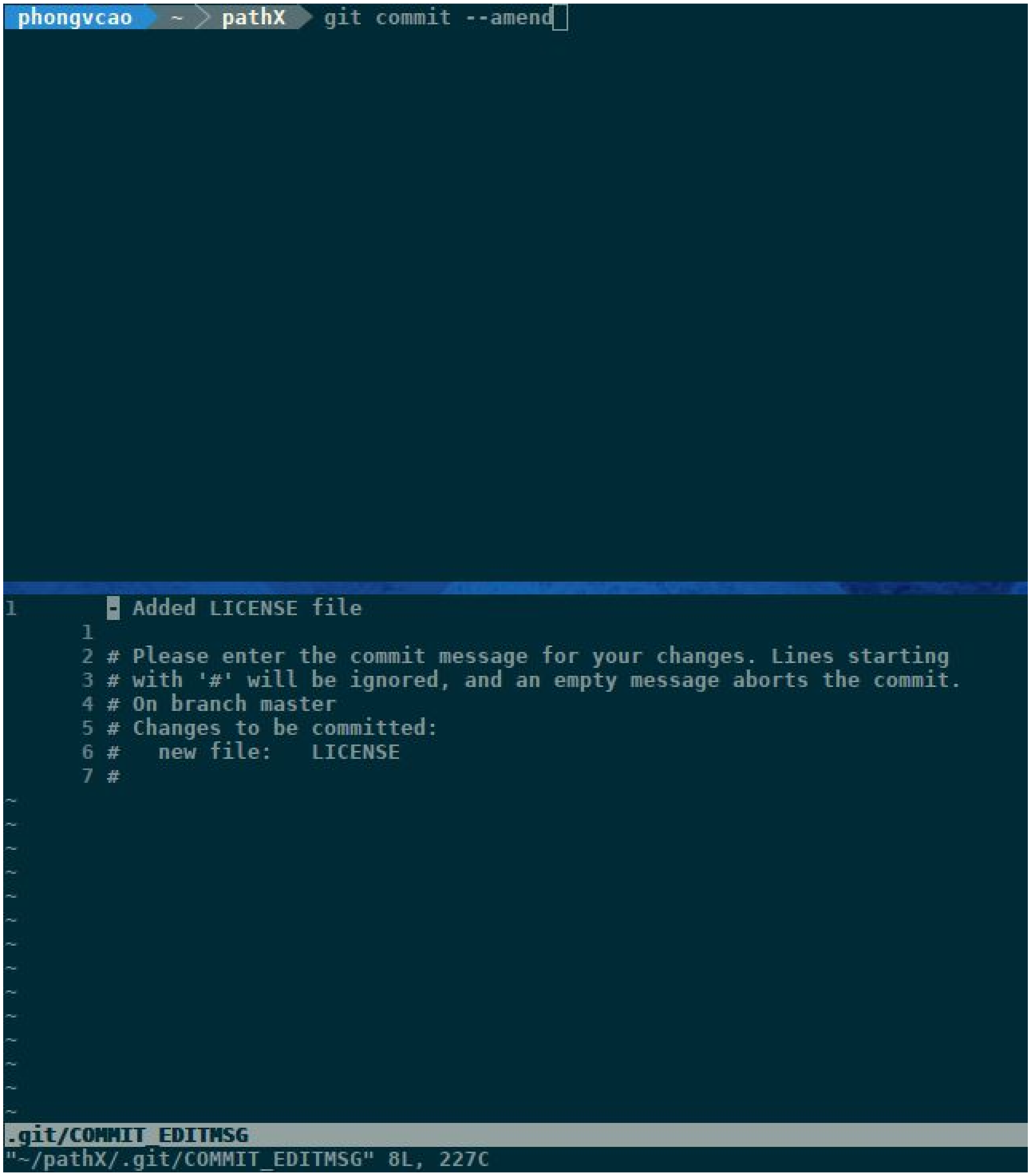


```
phongvcao  ~  pathX  git log
commit 051f1d9b6b1d3e3ff07df5a79c51ae08a29c70f1
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sat Sep 20 05:53:16 2014 -0400

    - Added LICENSE file

commit db1b4d1a144dc9c437b80b3bb55abf1e1588a883
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Wed Sep 17 12:29:03 2014 -0400

    - Added AUTHORS file

commit afd3f38eba366bfa1996e02162b10bab888e746d
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sun Sep 14 15:41:13 2014 -0400

    - Initialized an empty Git repository
phongvcao  ~  pathX  git log -n 1
commit 051f1d9b6b1d3e3ff07df5a79c51ae08a29c70f1
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sat Sep 20 05:53:16 2014 -0400

    - Added LICENSE file
phongvcao  ~  pathX  git reset --hard afd3f3
HEAD is now at afd3f38 - Initialized an empty Git repository
phongvcao  ~  pathX  git log -n 1
commit afd3f38eba366bfa1996e02162b10bab888e746d
Author: Phong V. Cao <phongvcao@phongvcao.com>
Date:   Sun Sep 14 15:41:13 2014 -0400

    - Initialized an empty Git repository
phongvcao  ~  pathX 
```

As you can see in the above screenshot, I switched from my current “Git commit” which is “`051f1d`” to the first initial “Git commit” which is “`afd3f3`”.

11. Edit a “Git commit” message (in an external editor):

```
$ git commit --amend
```

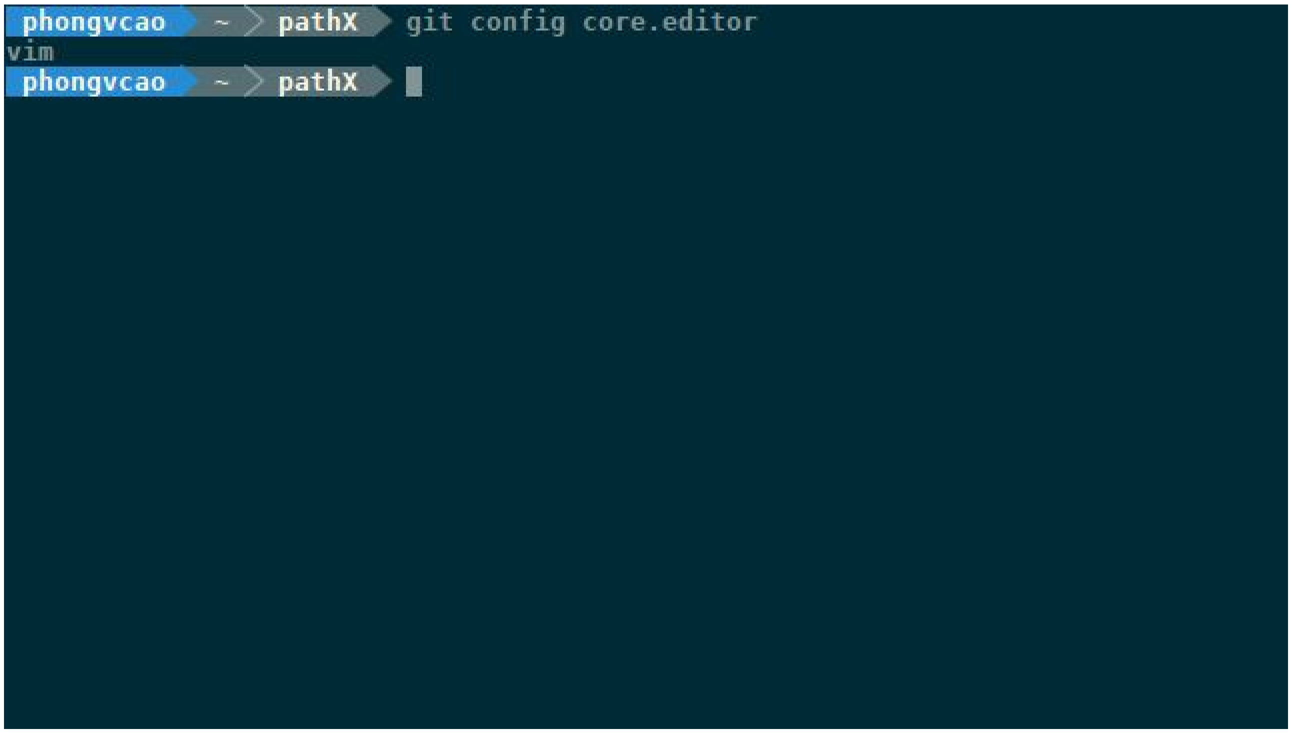


```
phongvcao  ~  pathX  git commit --amend
```

```
1       Added LICENSE file
      1
      2 # Please enter the commit message for your changes. Lines starting
      3 # with '#' will be ignored, and an empty message aborts the commit.
      4 # On branch master
      5 # Changes to be committed:
      6 #   new file:   LICENSE
      7 #
~
~
~
~
~
~
~
~
~
~
~
~
~
~
.git/COMMIT_EDITMSG
"~/pathX/.git/COMMIT_EDITMSG" 8L, 227C
```

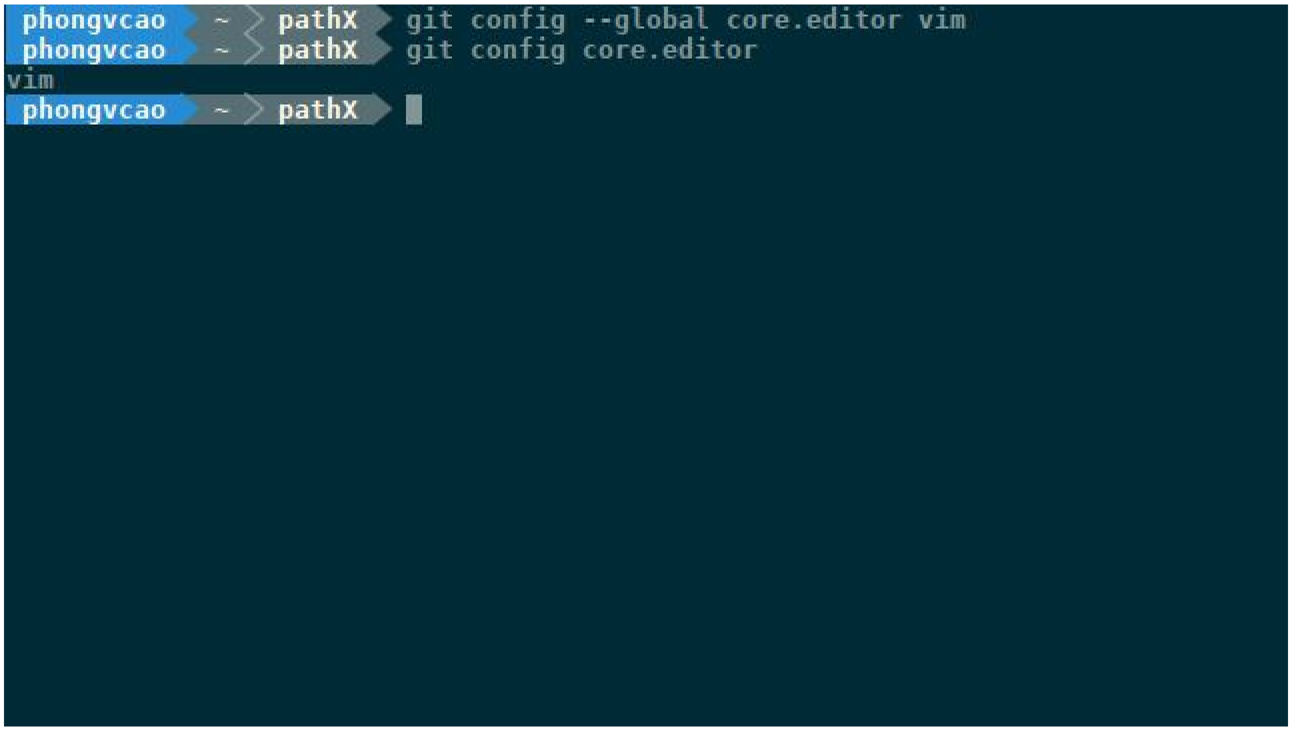
Note: Before issuing this command, make sure that the core.editor variable of your Git global settings has been set to your favorite text editor (see Step 3.2 above) by issuing the following command:

```
$ git config core.editor
```


```
phongvcao  ~  pathX  git config core.editor
vim
phongvcao  ~  pathX
```

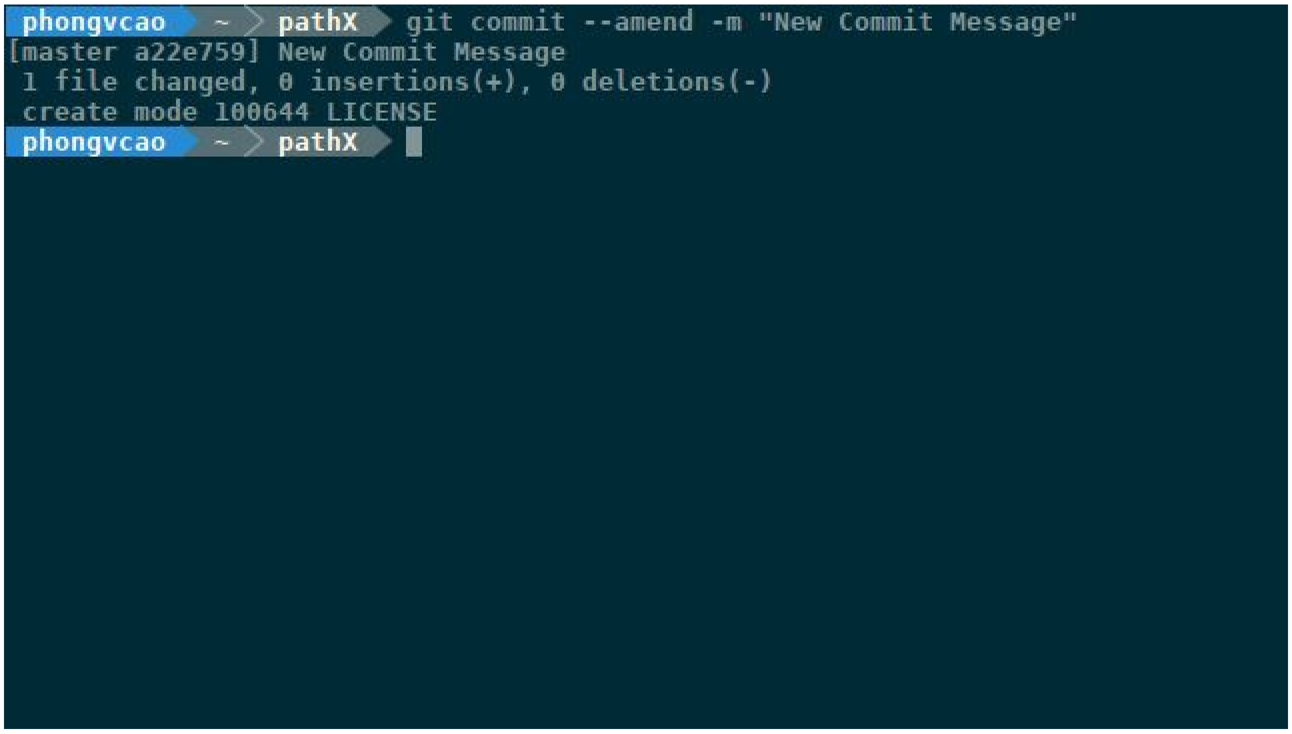
If your “Git commit” global editor is not yet set, you can set it using the following command (mentioned in Step 3.2):

```
$ git config --global core.editor <Your Text Editor of Choice>
```


```
phongvcao  ~  pathX  git config --global core.editor vim
phongvcao  ~  pathX  git config core.editor
vim
phongvcao  ~  pathX
```

12. Replace a “Git commit” message:

```
$ git commit --amend -m “<New Commit Message>”
```

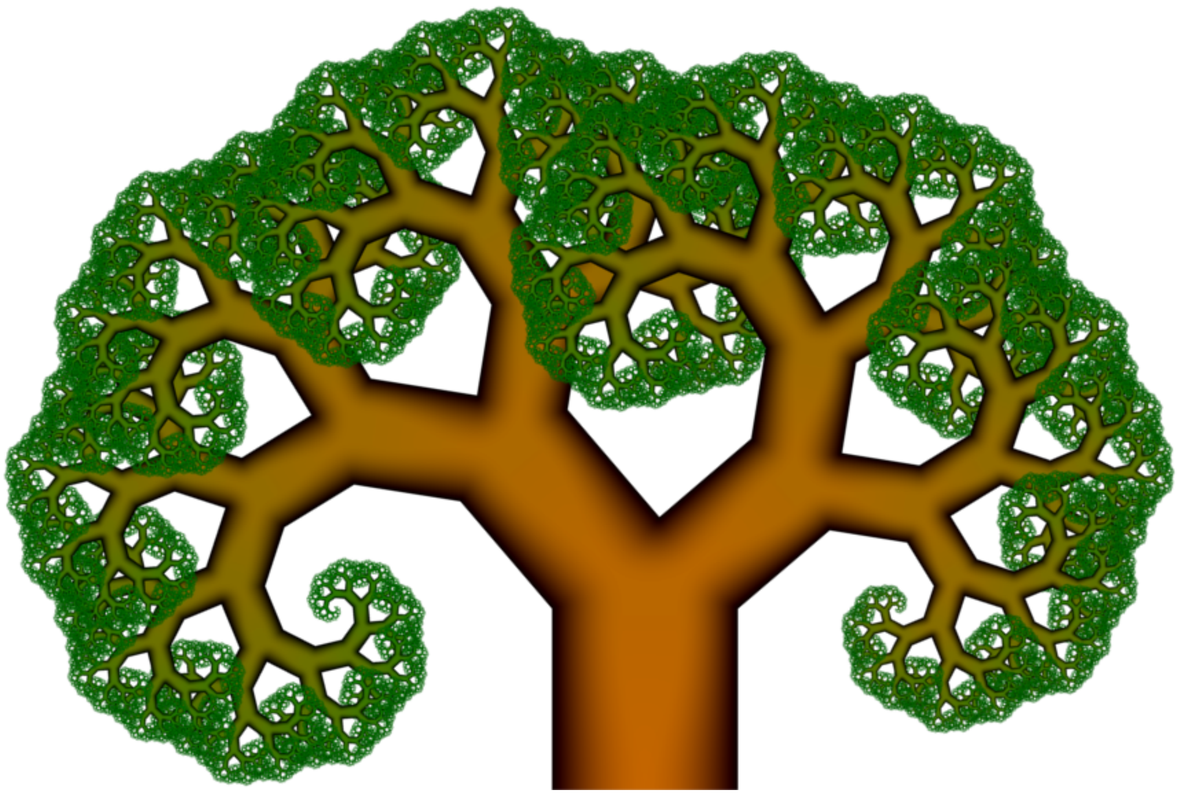


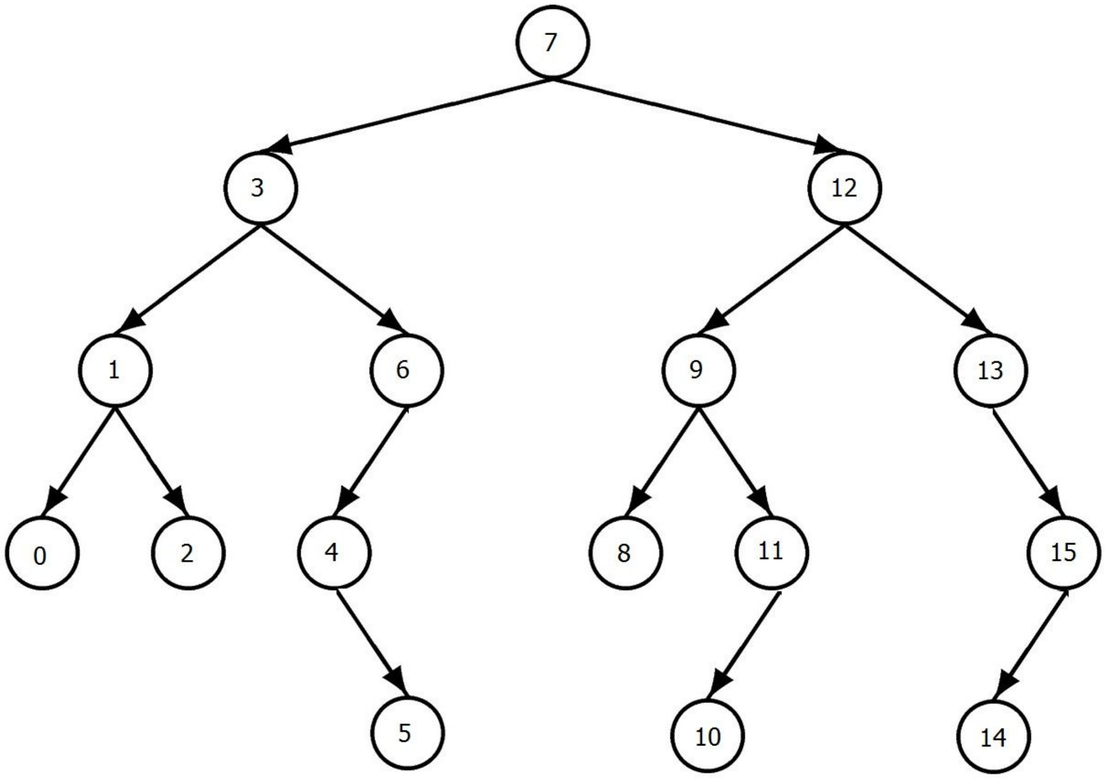
```
phongvcao  ~  pathX  git commit --amend -m "New Commit Message"
[master a22e759] New Commit Message
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 LICENSE
phongvcao  ~  pathX  
```

Step 14: Fundamentals of Git branch

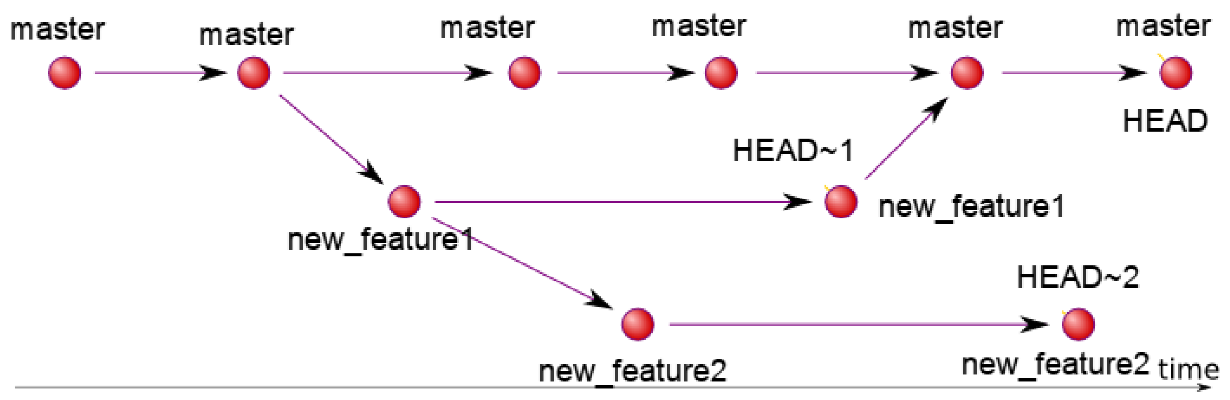
1. Now before going further into other Git stuffs, let’s learn about a very important concept of Git that we need to understand: Git branch.

So what is a “Git branch”? Well have you ever seen a tree before (any kind of tree, either a tree in nature or a binary or n-ary tree)?





And here is what "Git branches" look like (very much like a "tree", but I would say it looks more like a directed graph!):



So Git branch is basically a way for you to make changes to your local pathX repository without actually making any changes. Wow so confusing is it? Sound like magic to me!

Well that is the real power of Git (or any other VCS, but Git is the best at Branching and that is why Silicon Valley loves it!). When you initialize a Git repository on your local computer by “cd-ing” into the folder and issuing the command “git init”, Git automatically structures the project like a tree with a root node (but without any branch - or children nodes). This root node is called the “master” branch. The master branch represents the stable version (the least-buggy and most rigorously-tested branch) of a software.

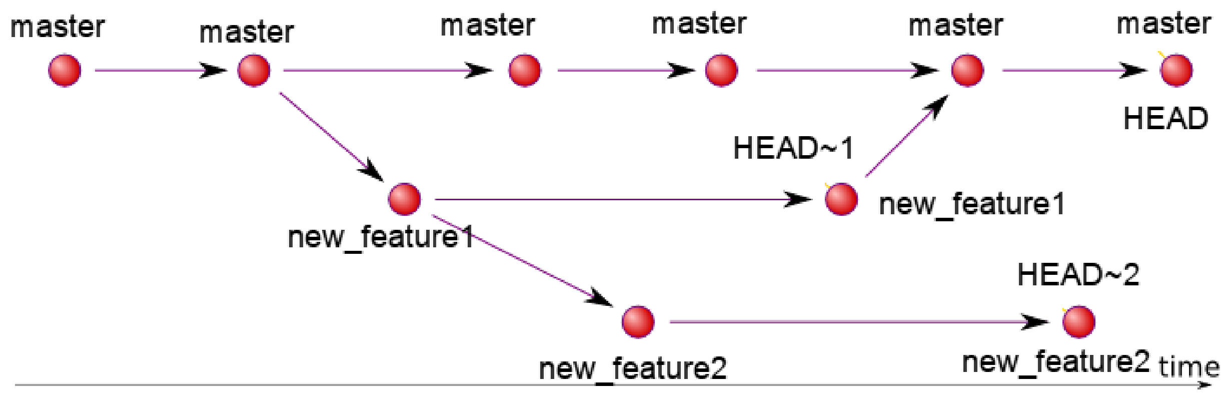
As a rule of thumb, whenever you want to make some small changes or add some new features on the software, you have to (yes, you read that correctly, YOU HAVE TO, or the project owners/administrators will not accept your changes) create another “Git branch” that deviates from the “master” branch and then work from there. After finishing all the changes and issuing a “git commit” to record the changes into Git, all you need to do is merging your own development branch into the “master” branch.

Look at the above nice graph (again? LOL). Suppose you want to implement a new feature called “new_feature1”. What you need to do is to create a branch (we will talk about how to do this later - it is very easy, much easier than understanding how Git branch works) called “new_feature1” branch. When you create “new_feature1”, Git rapidly takes a snapshot of your “master” branch and names it “new_feature1” branch. You can then switch to the “new_feature1” branch (which is merely a snapshot of “master” branch) and do whatever you want with it without actually changing the “master” branch. When you are finished making changes to “new_feature1” branch, all you need to do is to “merge” “new_feature1” branch to the “master” branch.

The same goes for the “new_feature2” branch in the graph above. Let’s say while working on “new_feature1”, you have another idea for implementing “new_feature1”. However, you don’t know if this new idea is gonna work, so you just want to try out this new idea and if it works out, you will merge it back into “new_feature1” later. In this case, you can create a “branch of branch” (also called a “child branch” or “nested branch”) from “new_feature1”, and name it “new_feature2”, to implement the new idea that you have.

Since “new_feature2” branch is a *branch” of “new_feature1” branch, “new_feature2” will contain a snapshot of the “new_feature1” branch and not the “master” branch. When doing a “Git merge” (you’ll see it later), you have to merge “new_feature2” to “new_feature1” first, and then merge “new_feature1” (which now contains “new_feature2”) into “master” branch.

2. Now let’s look at another concept that is just as important (if not more) than Git branch: Git HEAD.



So what is a Git HEAD?

As you can see from the above graph, a single Git branch contains many “Git commits” (which is issued everytime you want to record the changes you made in your source code to your Git repo). Whenever you do a “Git commit”, Git internally creates a “commit node” (look at the graph above) and appends that “commit node” to the end of the branch (like in a Linked List). A Git HEAD is a pointer to the current “commit node” that you are on in a particular branch. When you switch to another branch, Git automatically switches you to the “commit node” pointed to by the Git HEAD pointer of that branch.

Each HEAD pointer is named differently and automatically by Git (look at the above graph). The HEAD pointer to the current “Git commit” in the “master” branch is called “HEAD” and subsequent HEAD pointer to the current “commit node” in other branches are usually named “HEAD~1”, “HEAD~2”, and so on.

Whenever you fork and then clone pathX (or any other project on Bitbucket or Github) to your local computer, your Git HEAD pointer will always points to the latest “commit node” in the “master” branch of the project (unless you specify in the “Git clone” parameter that you want another branch. This feature is quite trivial since you can easily switch to another branch after cloning, which I will talk about in the next Step).

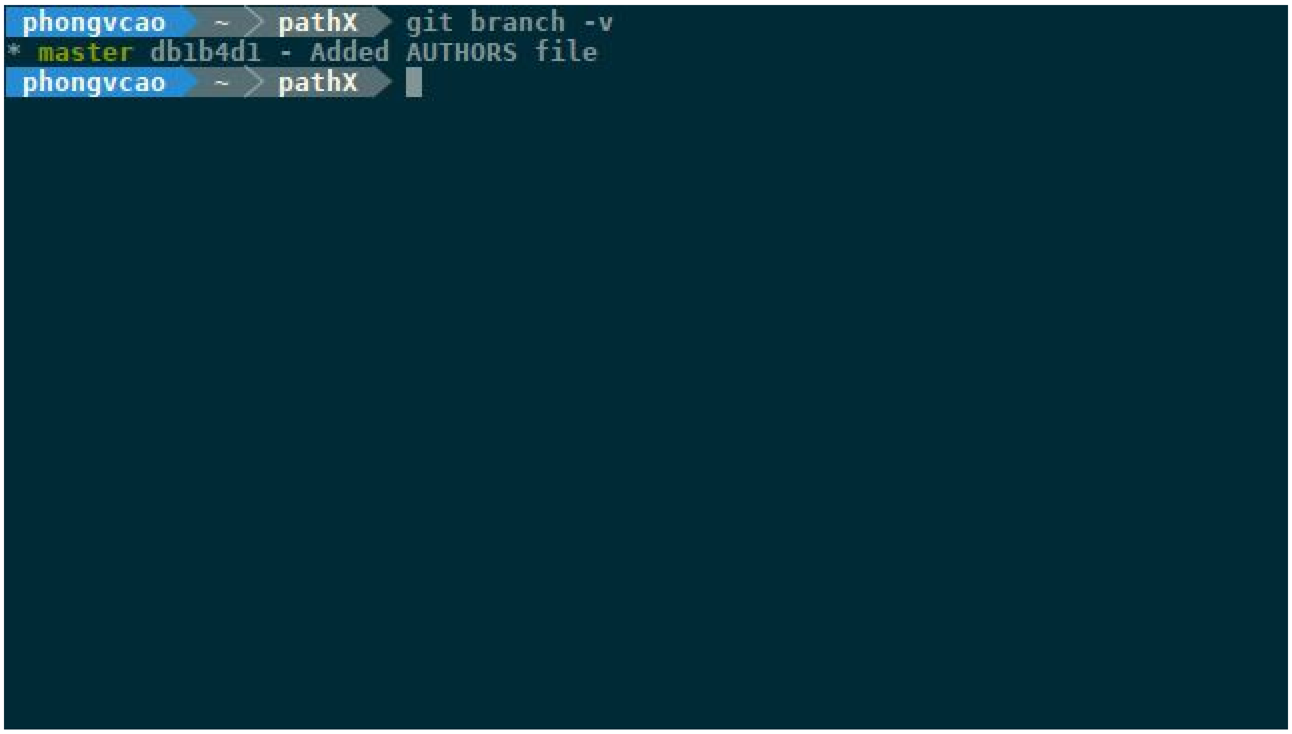
If you’re currently at the latest “master” node (which means the Git HEAD pointer is currently pointing at the last “master”), and then you switch to “new_feature1” branch, the Git HEAD pointer will point to your current “commit node” location on the “new_feature1” branch, which is where the HEAD~1 pointer is pointing to. The same goes for HEAD~2.

Step 15: Git branch commands

Now that you’ve learned about Git branches fundamentals, let’s look at the concrete examples and commands used to manipulate Git branches.

1. List all Git branches:

```
$ git branch -v
```

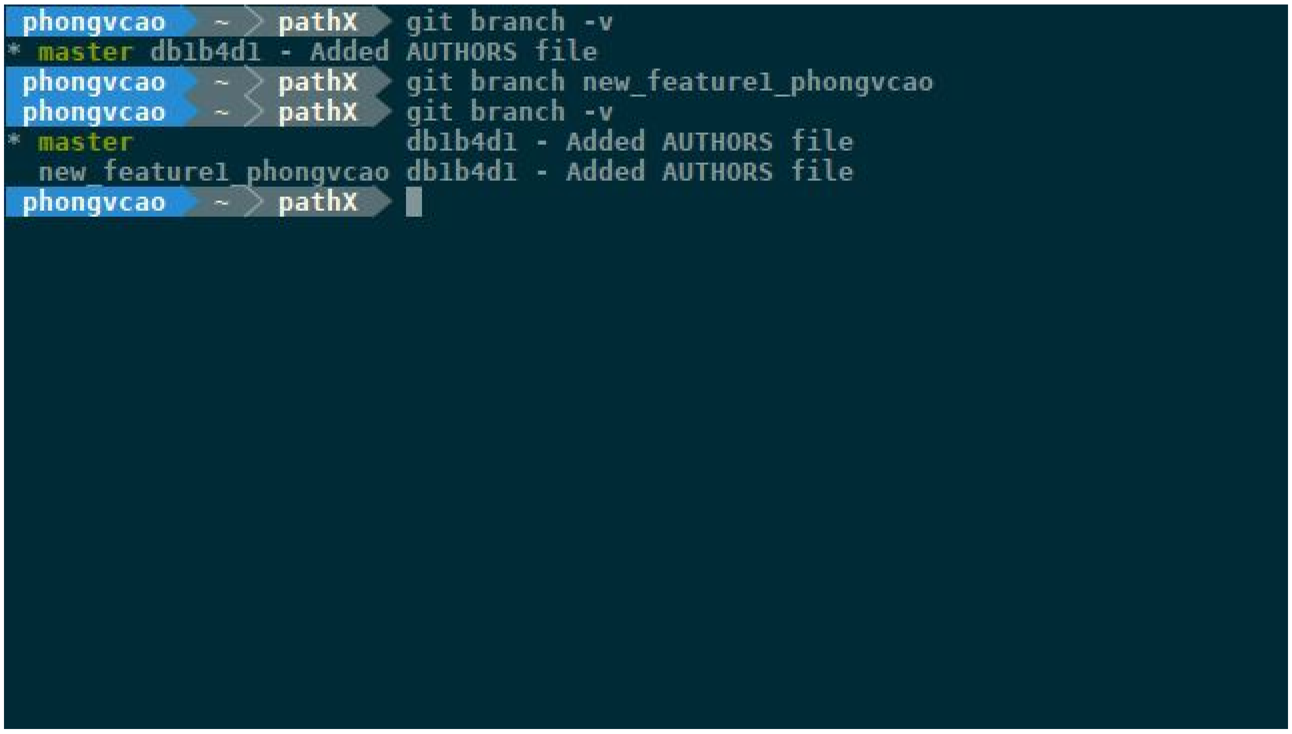


```
phongvcao  ~  pathX  git branch -v
* master db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  
```

The branch marked with the * character is your current branch

2. Create a new Git branch:

```
$ git branch <newBranchName_authorBitbucketUsername>
```



```
phongvcao  ~  pathX  git branch -v
* master db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  git branch new_feature1_phongvcao
phongvcao  ~  pathX  git branch -v
* master                db1b4d1 - Added AUTHORS file
  new_feature1_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  
```

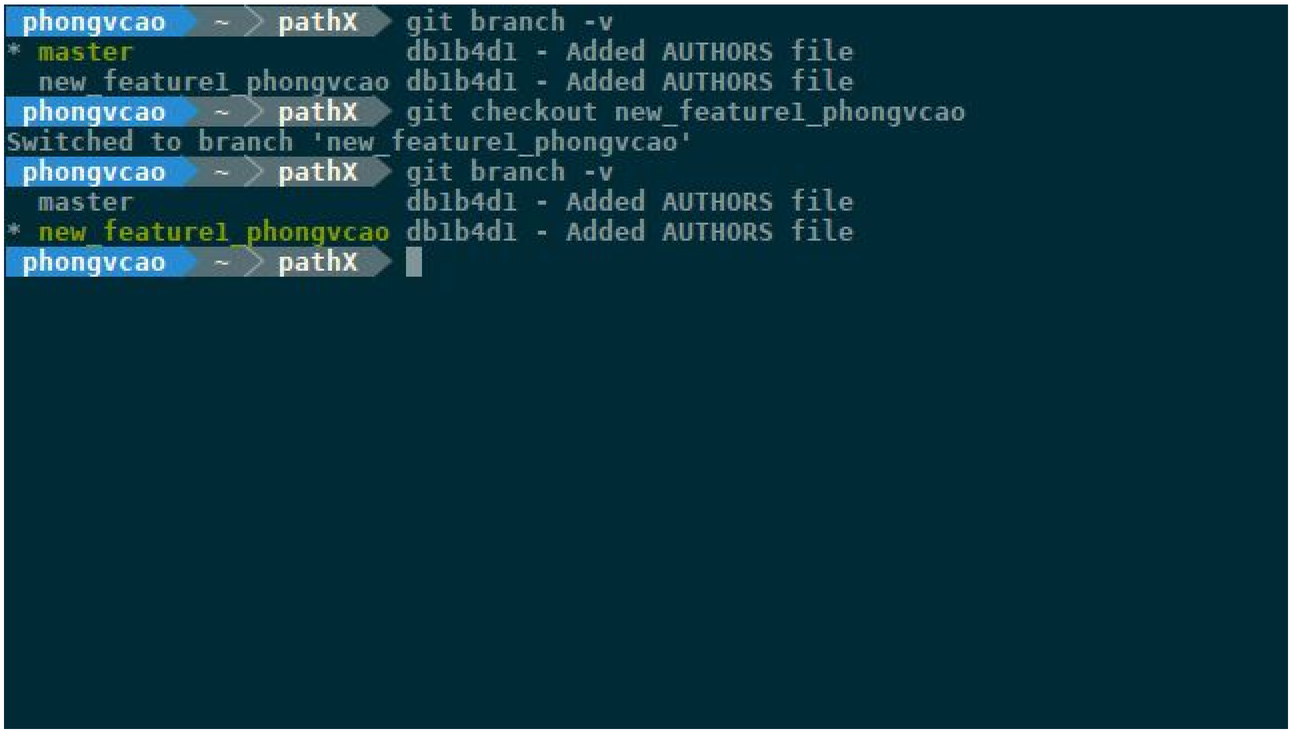
As you can see in the above screenshot, the “new_feature1” branch has been successfully created.

As a convention when you create a new branch please indicates your Bitbucket username in the name of the branch, separated by underscore “_” so we know who is mainly responsible for working on this particular branch.

Also note that when you create a new Git branch, your Git HEAD pointer is still pointing to the branch you’re currently at. You have to explicitly issue a Git switch (“Git checkout”) command to switch to the new Git branch you just created.

3. Switch to another Git branch:

```
$ git checkout <branch_name>
```

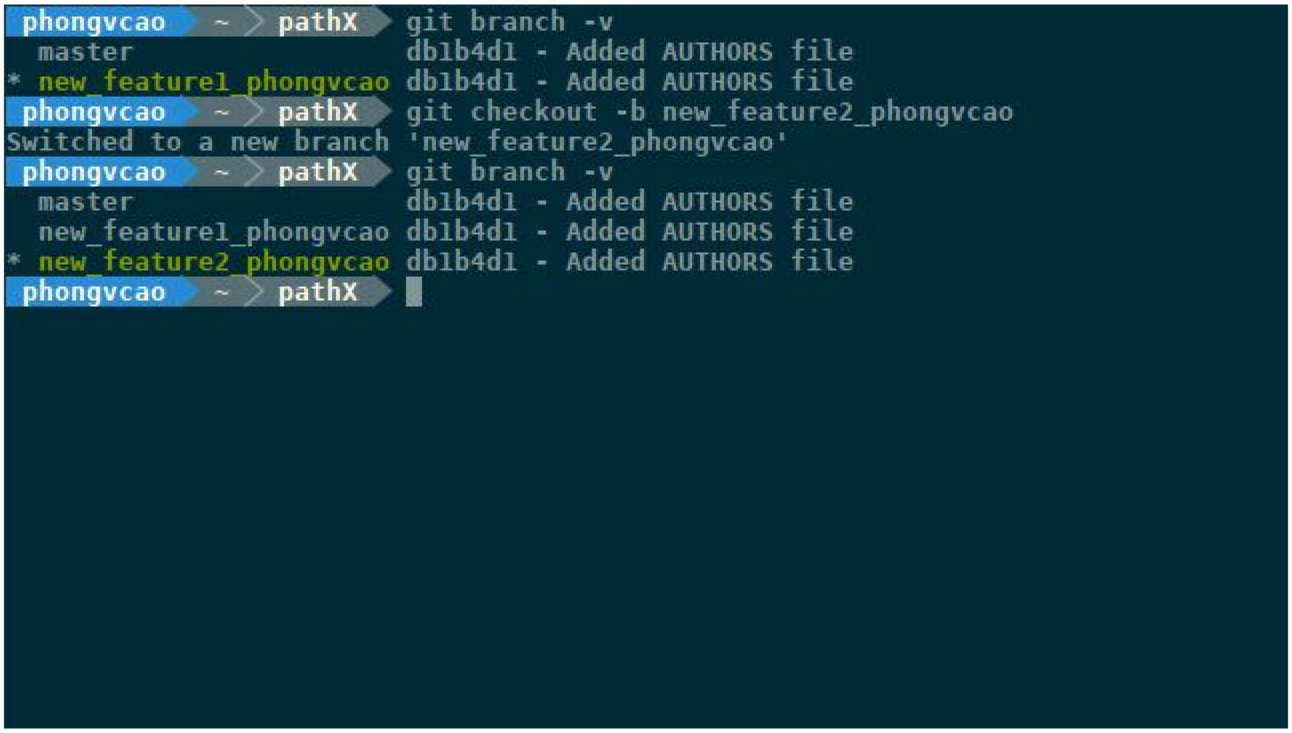


```
phongvcao  ~  pathX  git branch -v
* master                 db1b4d1 - Added AUTHORS file
  new_feature1_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  git checkout new_feature1_phongvcao
Switched to branch 'new_feature1_phongvcao'
phongvcao  ~  pathX  git branch -v
  master                 db1b4d1 - Added AUTHORS file
* new_feature1_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX
```

As you can see in the above screenshot, the * character first highlighted the “master” branch. Then after the command is issue the * character highlighted the “new_feature1” branch.

4. Create and switch to a new Git branch (at once):

```
$ git checkout -b <new_branch_name>
```



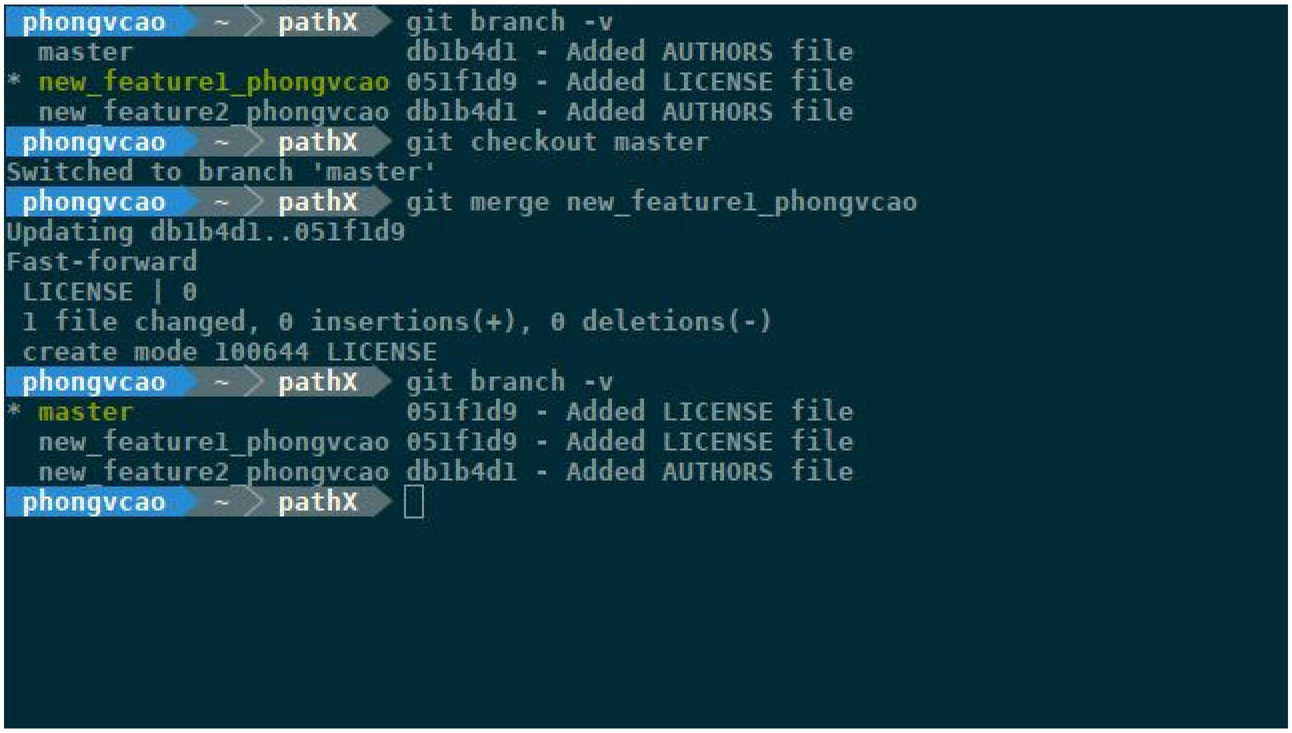
```
phongvcao  ~  pathX  git branch -v
  master                 db1b4d1 - Added AUTHORS file
* new_feature1_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  git checkout -b new_feature2_phongvcao
Switched to a new branch 'new_feature2_phongvcao'
phongvcao  ~  pathX  git branch -v
  master                 db1b4d1 - Added AUTHORS file
  new_feature1_phongvcao db1b4d1 - Added AUTHORS file
* new_feature2_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX
```

As you can see in the above screenshot, the * character first highlighted the “new_feature1” branch. Then after the command is issue the * character highlighted the newly-created “new_feature2” branch.

5. Merge your Git branch with another branch:

In order to merge your current Git branch to another branch (for example, the "master" branch), you have to first switch to the branch that is more "stable" using "git checkout". Then you can issue the following command to merge your current more-stable branch to another less-stable branch:

```
$ git checkout <more_stable_branch>
$ git merge <less_stable_branch>
```

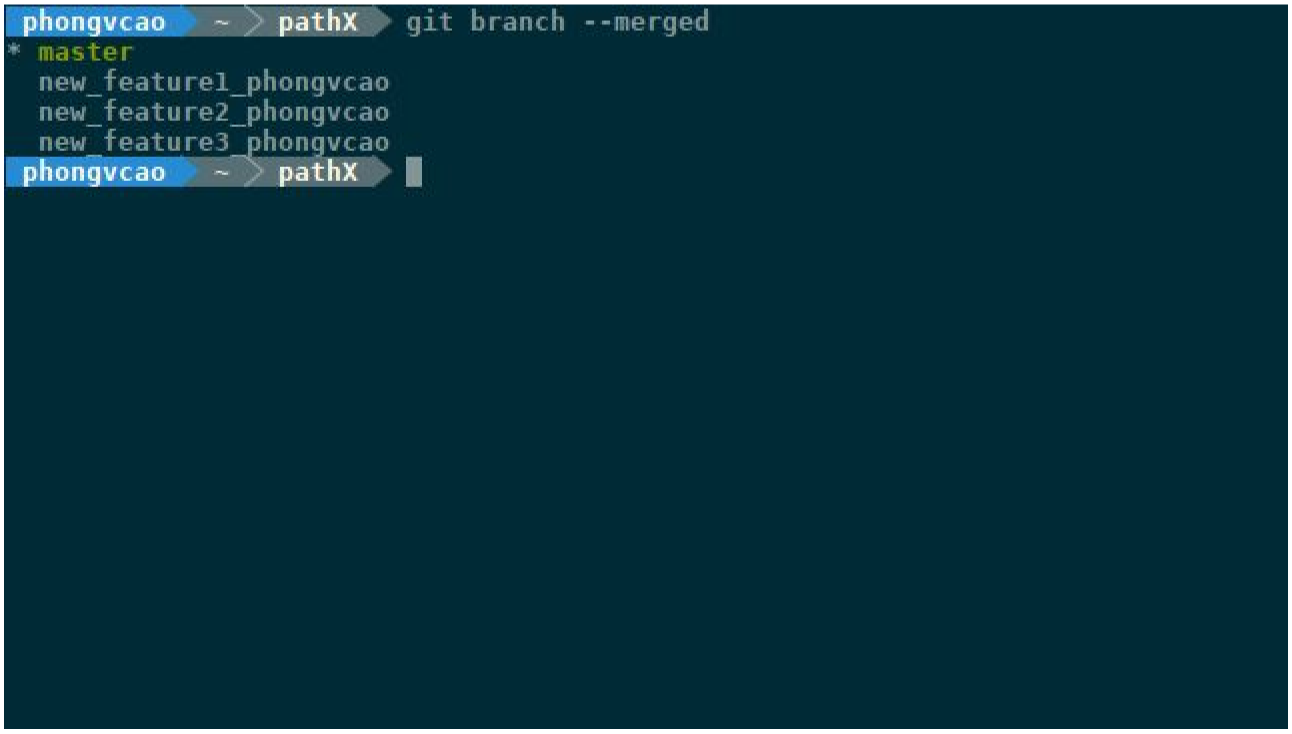


```
phongvcao  ~  pathX  git branch -v
  master                  db1b4d1 - Added AUTHORS file
* new_feature1_phongvcao 051f1d9 - Added LICENSE file
  new_feature2_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX  git checkout master
Switched to branch 'master'
phongvcao  ~  pathX  git merge new_feature1_phongvcao
Updating db1b4d1..051f1d9
Fast-forward
 LICENSE | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 LICENSE
phongvcao  ~  pathX  git branch -v
* master                  051f1d9 - Added LICENSE file
  new_feature1_phongvcao 051f1d9 - Added LICENSE file
  new_feature2_phongvcao db1b4d1 - Added AUTHORS file
phongvcao  ~  pathX
```

As you can see in the above screenshot, I first checkout the "master" branch which is more stable than "new_feature1". After that, I merge the "master" branch with the "new_feature1" branch.

6. List all Git branches that WERE merged to your current branch:

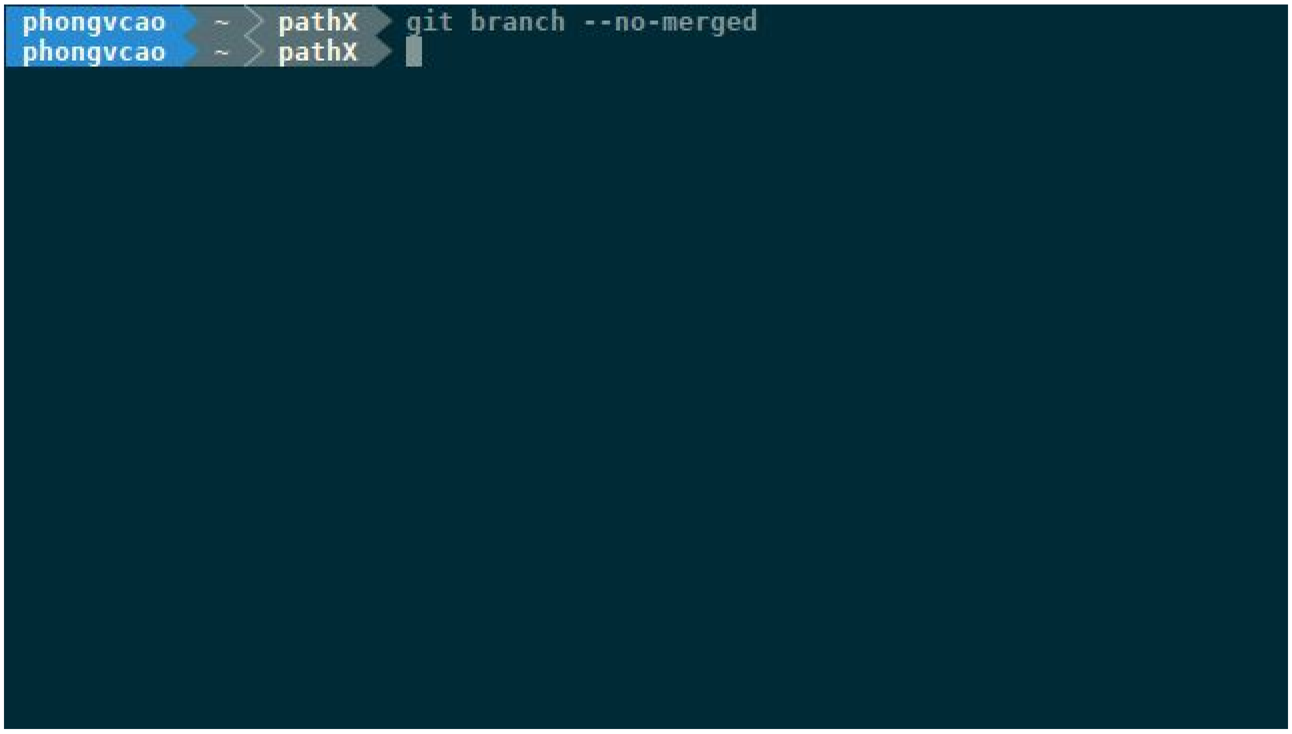
```
$ git branch --merged
```



```
phongvcao  ~  pathX  git branch --merged
* master
  new_feature1_phongvcao
  new_feature2_phongvcao
  new_feature3_phongvcao
phongvcao  ~  pathX
```

7. List all Git branches that WERE NOT merged to your current branch:

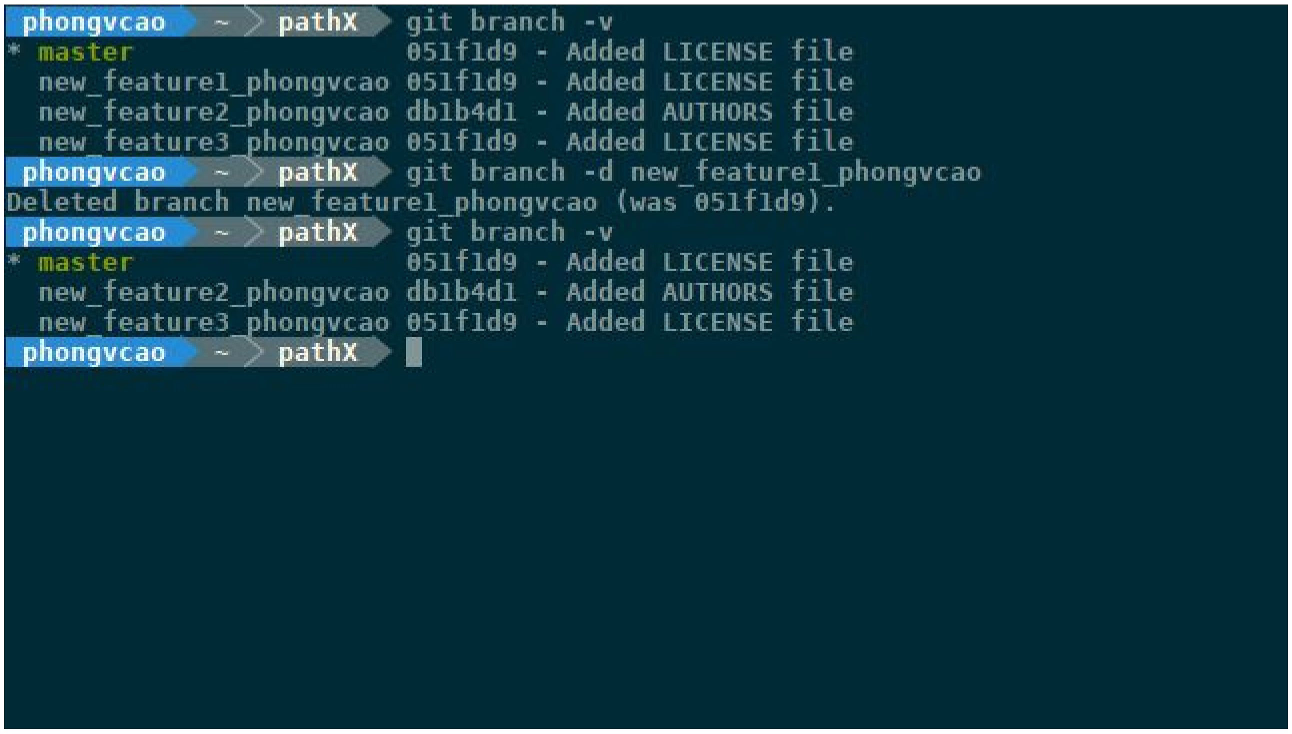
```
$ git branch --no-merged
```



```
phongvcao  ~  pathX  git branch --no-merged
phongvcao  ~  pathX
```

8. Delete a local Git branch

```
$ git branch -d <branch_name>
```

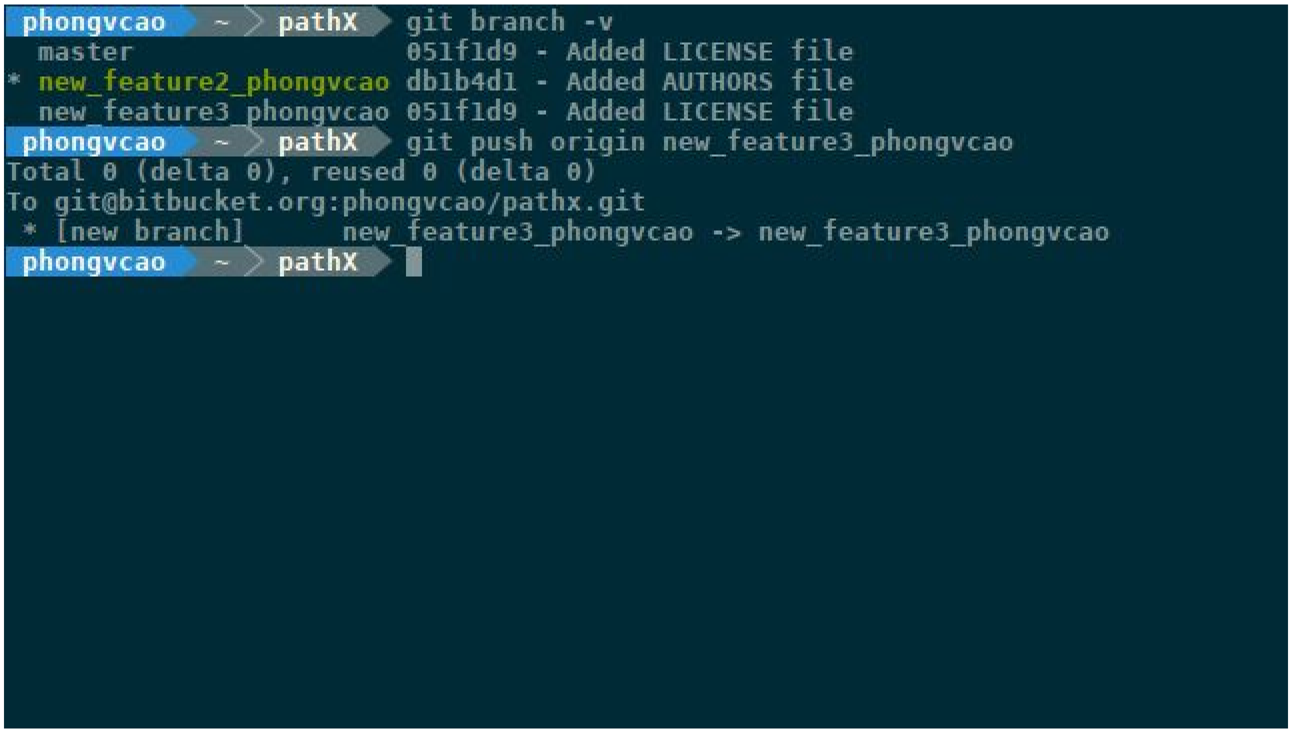

```
phongvcao  ~  pathX  git branch -v
* master                 051f1d9 - Added LICENSE file
  new_feature1_phongvcao 051f1d9 - Added LICENSE file
  new_feature2_phongvcao db1b4d1 - Added AUTHORS file
  new_feature3_phongvcao 051f1d9 - Added LICENSE file
phongvcao  ~  pathX  git branch -d new_feature1_phongvcao
Deleted branch new_feature1_phongvcao (was 051f1d9).
phongvcao  ~  pathX  git branch -v
* master                 051f1d9 - Added LICENSE file
  new_feature2_phongvcao db1b4d1 - Added AUTHORS file
  new_feature3_phongvcao 051f1d9 - Added LICENSE file
phongvcao  ~  pathX
```

After this command is executed, the branch <branch_name> is deleted from the local repository on your computer. However, if you've already pushed this <branch_name> to Bitbucket, the branch <branch_name> still exists online on Bitbucket.

9. Push a local Git branch to Bitbucket:

To push a local branch to Bitbucket, issue the following command:

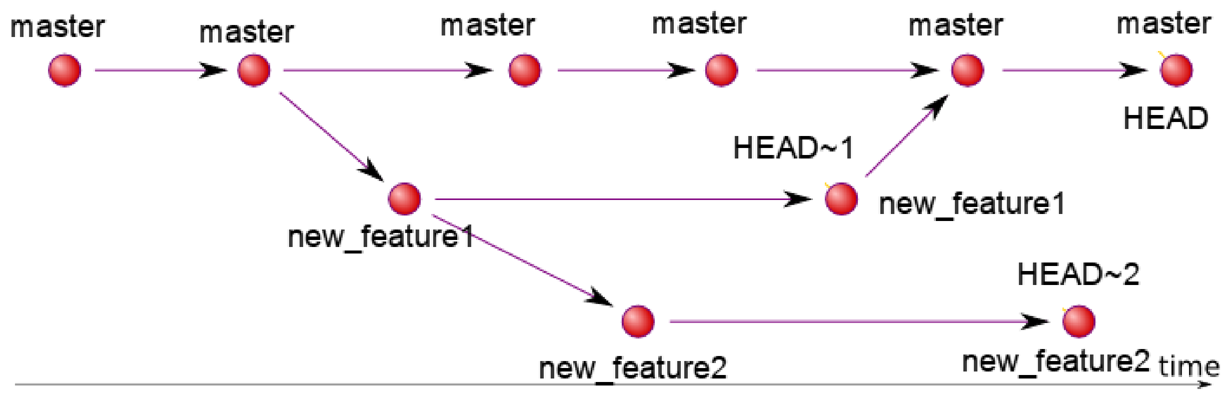
```
$ git push <remote_name> <branch_name>
```


```
phongvcao  ~  pathX  git branch -v
  master                 051f1d9 - Added LICENSE file
* new_feature2_phongvcao db1b4d1 - Added AUTHORS file
  new_feature3_phongvcao 051f1d9 - Added LICENSE file
phongvcao  ~  pathX  git push origin new_feature3_phongvcao
Total 0 (delta 0), reused 0 (delta 0)
To git@bitbucket.org:phongvcao/pathx.git
 * [new branch]      new_feature3_phongvcao -> new_feature3_phongvcao
phongvcao  ~  pathX
```

Remember the "`$ git push origin master`" command from Step 10 and Step 11 (when I first introduced you to "Git commit" and "Git push" and that you just kinda blindly followed the command without understanding what the hell is this "origin master" thing doing in there LOL =))))? Now, to be precise, the word "origin" here is the <remote_name> that you're going to push your local repository to and "master" here is the <branch_name>, the name of the local Git branch on your local computer that you want to push to your pathX fork on Bitbucket.

So simply put, "`$ git push origin master`" command pushes your "`master`" branch from your local repository to the "`master`" branch of the "`origin`" remote (which is your pathX fork on Bitbucket!). How amazing hah?

10. Relationship between "Git commit" and "Git branch":



So how does "Git commit" and "Git branch" relate to one another? Well here is the answer: When you're doing a "Git commit", you're only doing it on the "Git branch" you're currently on.

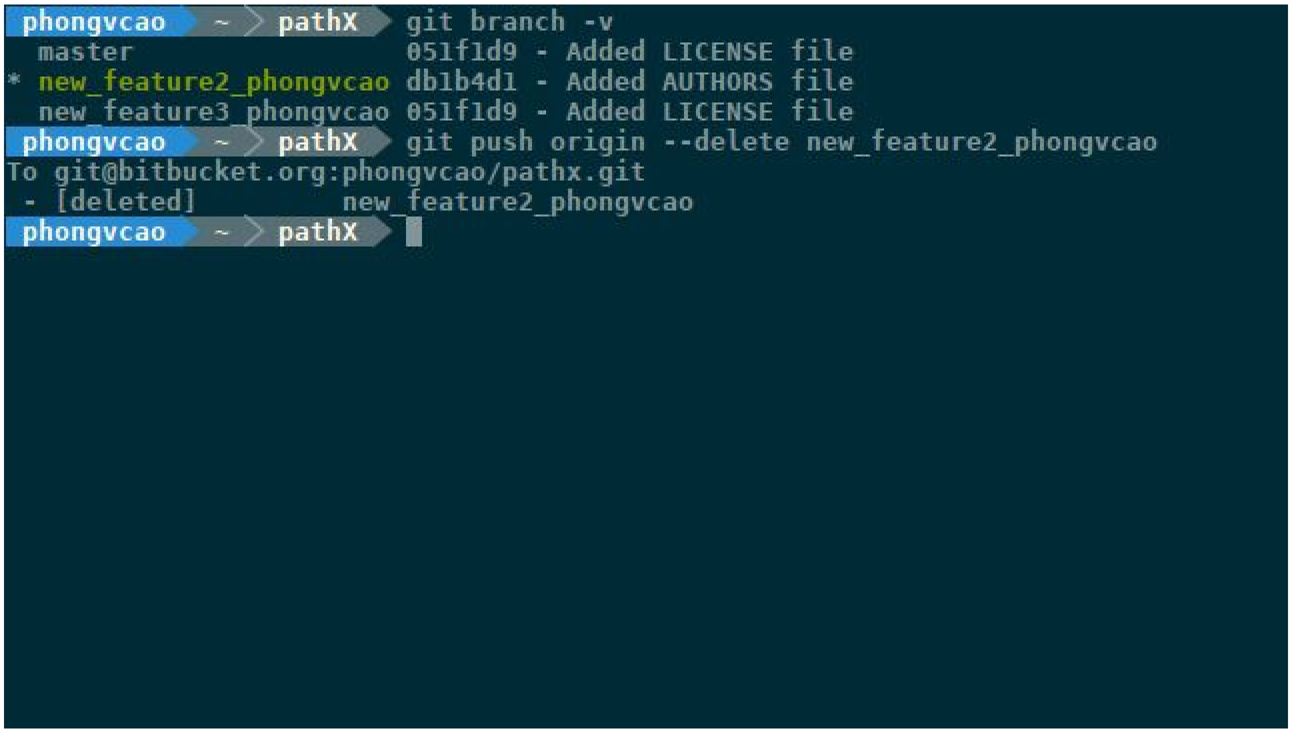
Let's say you've made some changes to the "master" branch. Then without doing any "Git commit", you switch to the "new_feature1" branch, make some changes to the files and do a "Git commit" there. Now if you switch back (again) to the "master" branch, you won't see the "Git commit" you just did because the "Git commit" you just did is "appended" to the "new_feature1" branch and not the "master" branch (See the graph above). So whenever you do a "Git commit", that "Git commit" only applies to the current Git branch that you're on at the time.

Thus, you can create a "Git commit", delete a "Git commit", moving back and forth between "Git commits" normally on any branch, and all of the "Git commit" Basics and Commands that you learned in Step 10 & Step 11 still hold true. However, remember that whatever you're doing with "Git commit" applies only to the "Git branch" that you're currently on and not the whole project. You CANNOT "Git commit" the whole project (and you shouldn't even if that is possible) in one single "git commit" command. You CAN only do "Git commit" on a branch and then switch to other branches and do the same.

11. Delete a remote Git branch:

As you can see in the preceding section, the “git branch -d” command only deletes the local <branch_name> branch on your computer. When you do a “Git commit” again and issue a “Git push”, the remote <branch_name> branch hosted on Bitbucket will not get deleted. To also delete the remote <branch_name> branch, issue the following command:

```
$ git push --delete <branch_name>
```



```
phongvcao  ~  pathX  git branch -v
  master                   051f1d9 - Added LICENSE file
* new_feature2_phongvcao db1b4d1 - Added AUTHORS file
  new_feature3_phongvcao 051f1d9 - Added LICENSE file
phongvcao  ~  pathX  git push origin --delete new_feature2_phongvcao
To git@bitbucket.org:phongvcao/pathx.git
 - [deleted]         new_feature2_phongvcao
phongvcao  ~  pathX
```

Step 16: Update (Synchronize) your fork with the original repository

Suppose that you’ve made some changes to the “new_feature1” branch of the pathX (fork) repository that you have in your local computer. Then you “Git merge” (locally) this “new_feature1” branch with the “master” branch and “Git push” these changes up to your online pathX fork on Bitbucket.

In the meantime, Prof. Richard McKenna who is an administrator of the pathX project also made some changes in the “master” branch of the *original* pathX repository (that you created your fork from) and then “Git commit” and “Git push” these changes to Bitbucket.

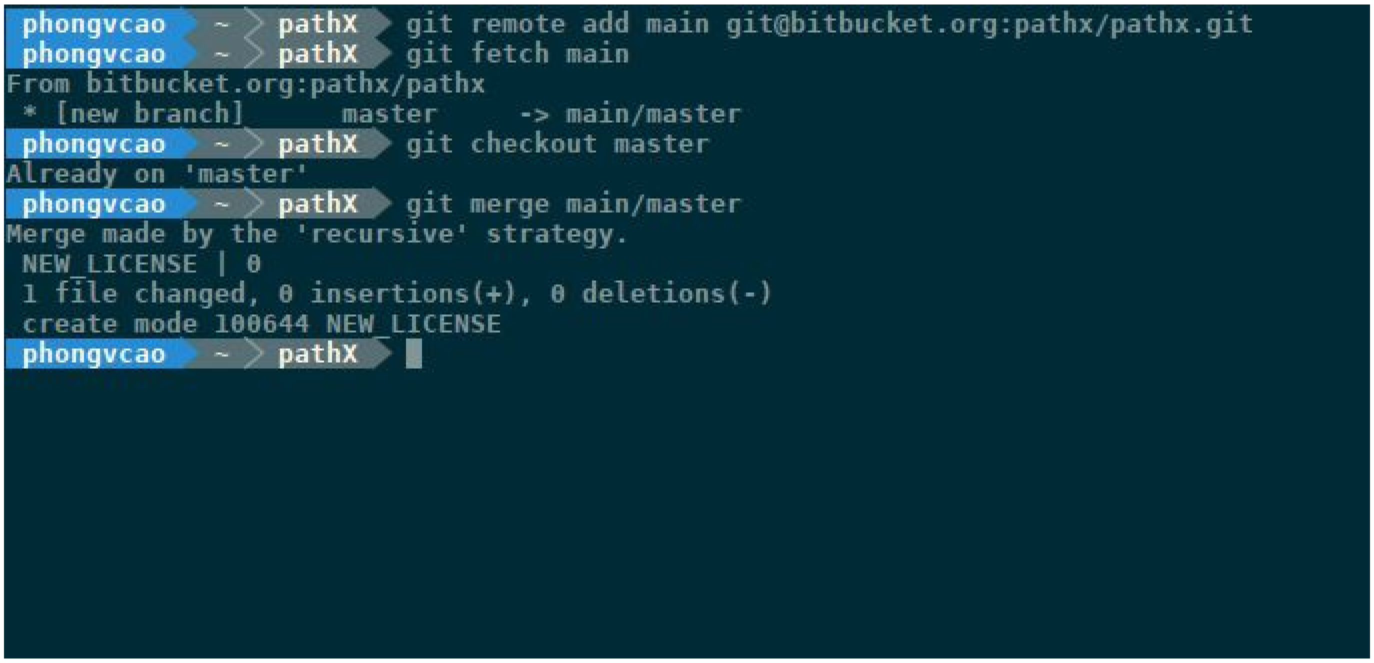
So what does this all mean? It means that the “master” branch of your local pathX repository is now **outdated** compared to the “master” branch of the *original* pathX repository (that you “forked” from) on Bitbucket. When Prof. Richard McKenna tries to do a “Git pull” and then a “Git merge” to merge your fork with his *original* “master” Git branch, the merge will not succeed.

In addition to merging failure, a local **outdated** “master” branch runs the risk (very likely actually) of making your “new_feature1” branch not working stably and coherently with the *original* “master” branch that Prof. Richard McKenna has. Sometimes this can result in huge and serious bugs in the project.

So in order to solve this problem, what you need to do is to Update (Synchronize) your fork with the *original* pathX repository before finalizing your “new_feature1” and Creating a Pull Request (more on this later).

Here are the commands to update (“synchronize”) your fork with the *original* pathX repository:

```
$ git remote add <a_name_for_pathX_original_remote> <original_pathX_remote_url>
$ git fetch <a_name_for_pathX_original_remote>
$ git checkout master
$ git merge <a_name_for_pathX_original_remote>/master
```



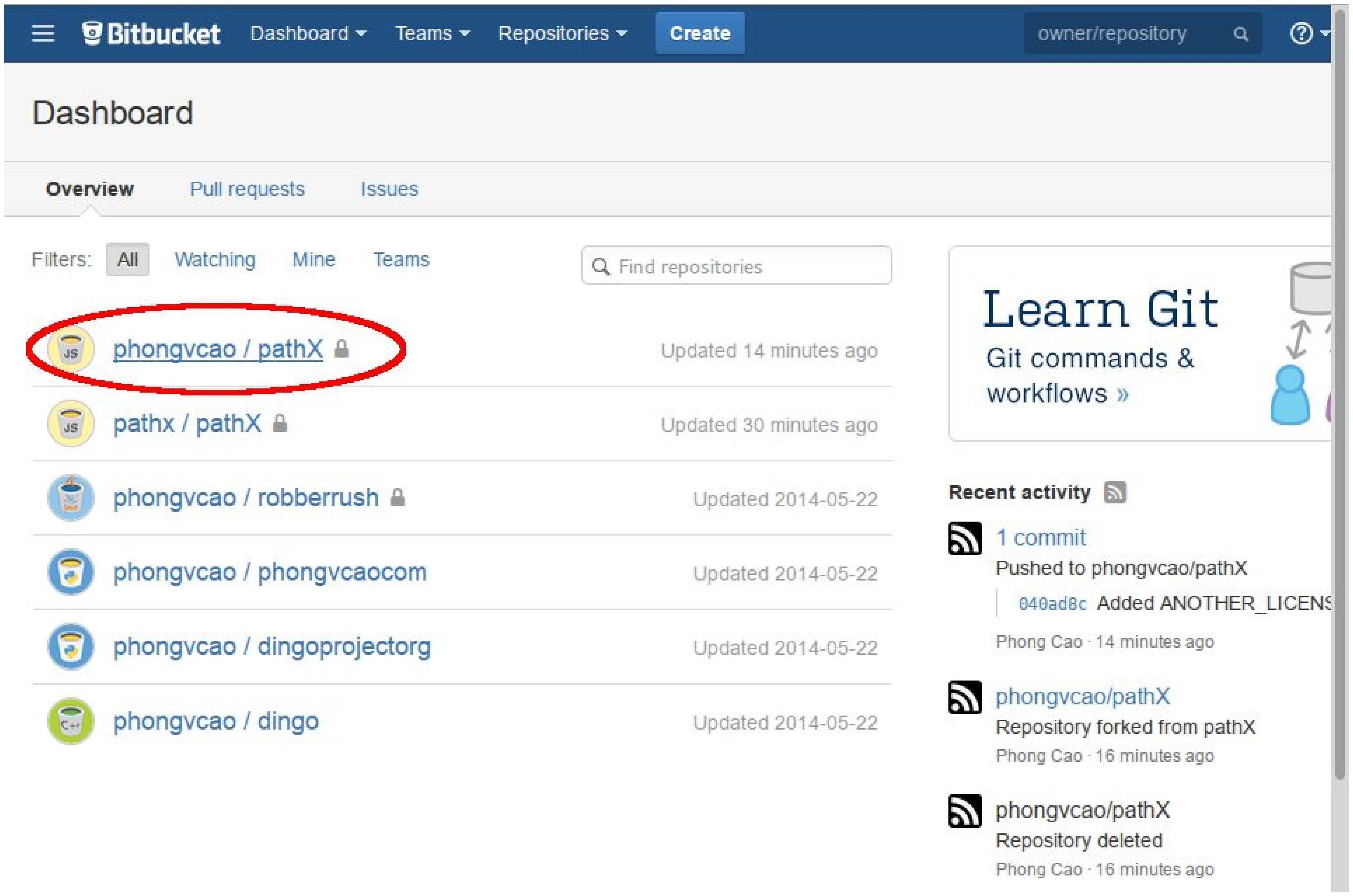
```
phongvcao  ~  pathX  git remote add main git@bitbucket.org:pathx/pathx.git
phongvcao  ~  pathX  git fetch main
From bitbucket.org:pathx/pathx
 * [new branch]      master     -> main/master
phongvcao  ~  pathX  git checkout master
Already on 'master'
phongvcao  ~  pathX  git merge main/master
Merge made by the 'recursive' strategy.
 NEW_LICENSE | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 NEW_LICENSE
phongvcao  ~  pathX  
```

As you can see in the above screenshot, I name the official, **original** pathX remote “main” (so basically I replaced <a_name_for_pathX_original_remote> with “main” and <original_pathX_remote_url> with git@bitbucket.org:pathx/pathx.git which is the URL to the official, **original** pathX remote on Bitbucket.

After this, you can do a “`$ git push origin master`” normally to push your newly-updated local pathX fork to the pathX fork repository on your Bitbucket account.

Step 17: Create a Pull Request

1. After you’ve confirmed the stability of the changes you made, head over to your Bitbucket account and go to the homepage of your pathX fork repository:



Dashboard

Overview Pull requests Issues

Filters: All Watching Mine Teams

Find repositories

phongvcao / pathX — Updated 14 minutes ago
pathx / pathX — Updated 30 minutes ago
phongvcao / robberrush — Updated 2014-05-22
phongvcao / phongvcaocom — Updated 2014-05-22
phongvcao / dingoprojectorg — Updated 2014-05-22
phongvcao / dingo — Updated 2014-05-22

Learn Git
Git commands & workflows »

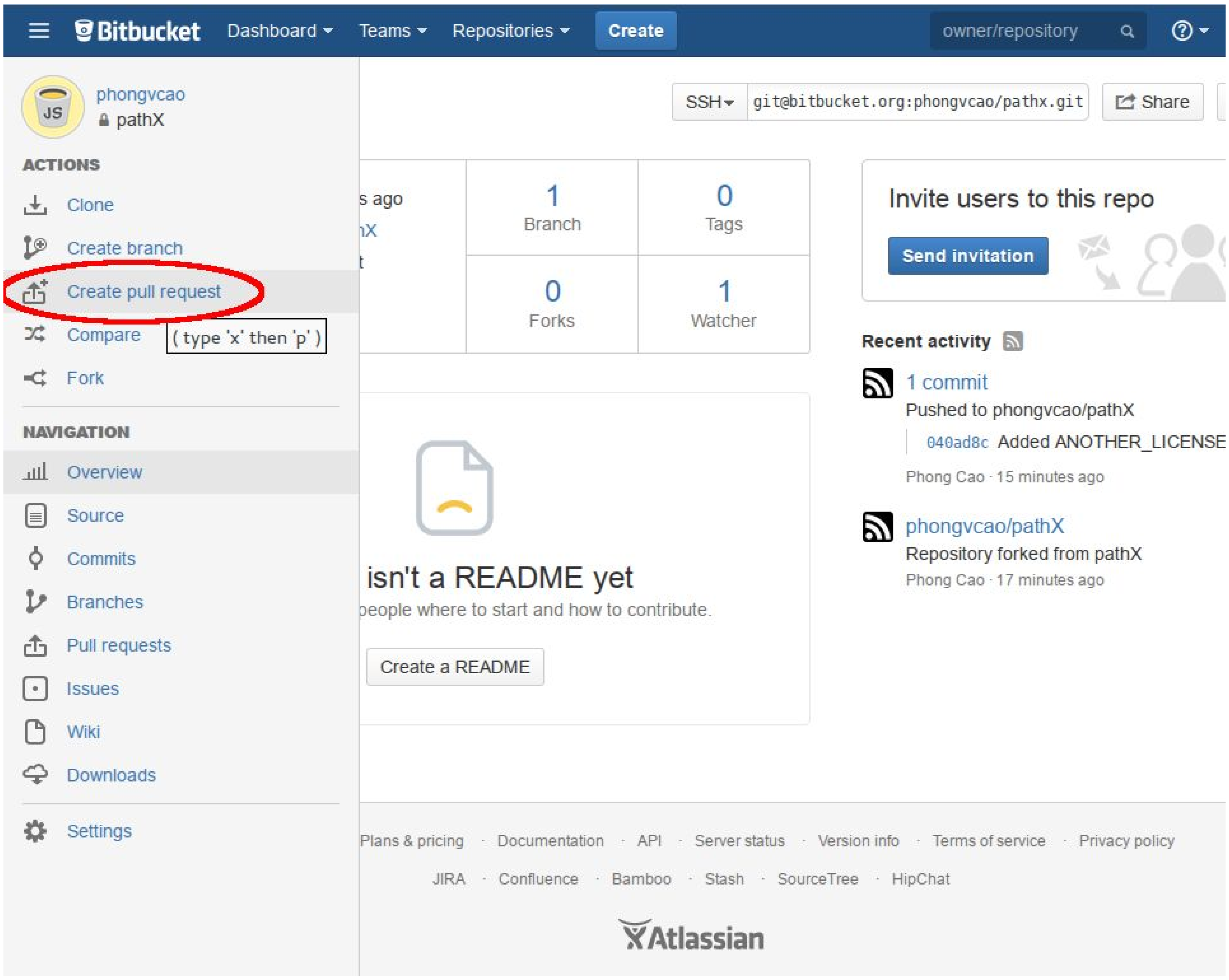
Recent activity

1 commit
Pushed to phongvcao/pathX
040ad8c Added ANOTHER_LICENS
Phong Cao · 14 minutes ago

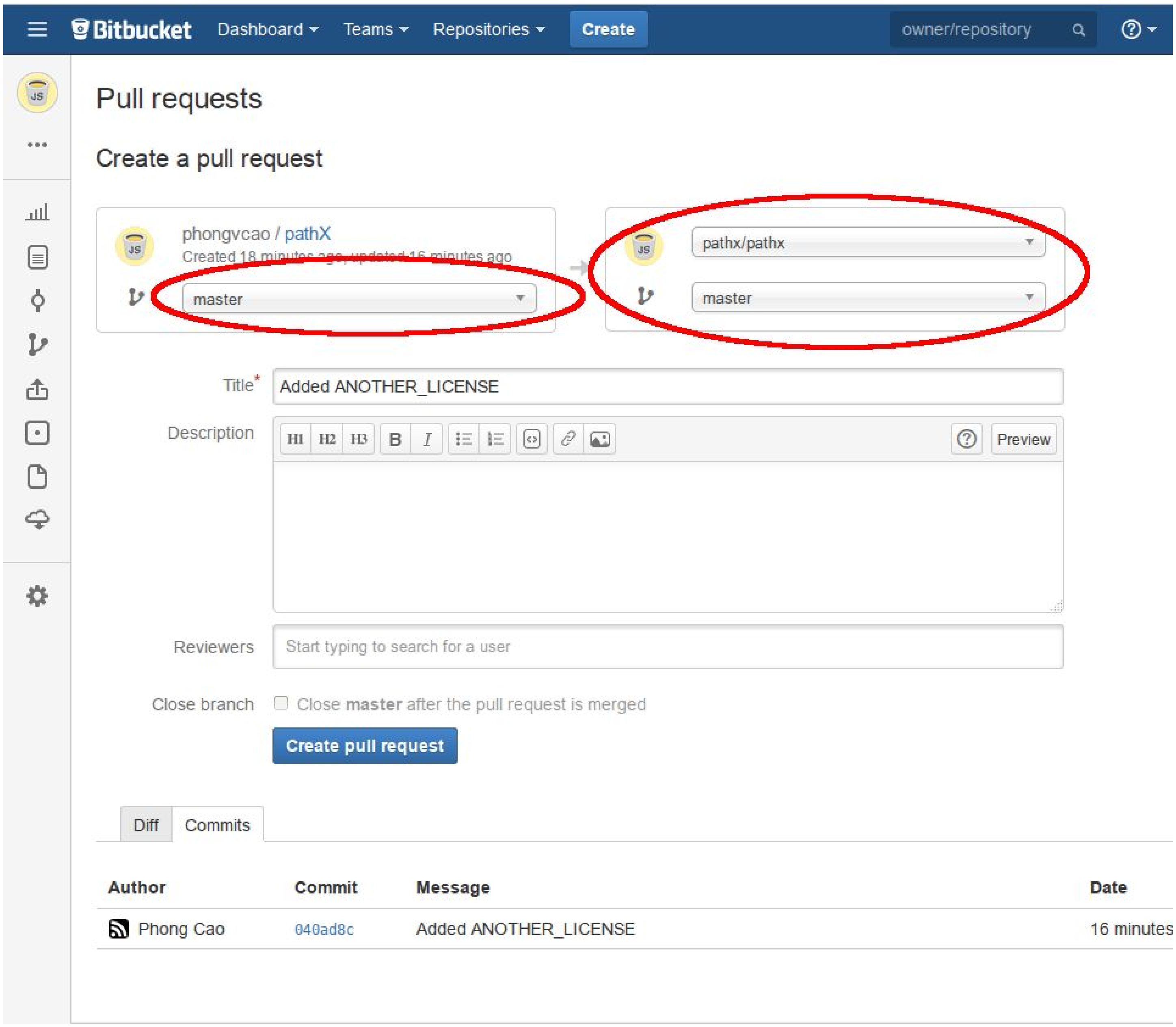
phongvcao/pathX
Repository forked from pathX
Phong Cao · 16 minutes ago

phongvcao/pathX
Repository deleted
Phong Cao · 16 minutes ago

2. After getting into your pathX fork repository on Bitbucket, expand the Bitbucket’s left Sidebar (see Step 7.2). Click on the option called “Create a Pull Request”, which will lead you to the “Create Pull Request” page:


Bitbucket
Dashboard
Teams
Repositories
Create
owner/repository
phongvcao
pathX
SSH
git@bitbucket.org:phongvcao/pathx.git
Share
ACTIONS
Clone
Create branch
Create pull request
Compare
(type 'x' then 'p')
Fork
NAVIGATION
Overview
Source
Commits
Branches
Pull requests
Issues
Wiki
Downloads
Settings
1 Branch
0 Tags
0 Forks
1 Watcher
Invite users to this repo
Send invitation
Recent activity
1 commit
Pushed to phongvcao/pathX
040ad8c Added ANOTHER_LICENSE
Phong Cao · 15 minutes ago
phongvcao/pathX
Repository forked from pathX
Phong Cao · 17 minutes ago
isn't a README yet
people where to start and how to contribute.
Create a README
Plans & pricing · Documentation · API · Server status · Version info · Terms of service · Privacy policy
JIRA · Confluence · Bamboo · Stash · SourceTree · HipChat
Atlassian

3. On the “Create a Pull Request” page, make sure that the Git branches and Git repositories information are correct (meaning they are what you want to merge):



Pull requests

Create a pull request

phongvcao / pathX
Created 18 minutes ago, updated 16 minutes ago
master

pathx/pathx
master

Title* Added ANOTHER_LICENSE

Description

Reviewers: Start typing to search for a user

Close branch: Close **master** after the pull request is merged

Create pull request

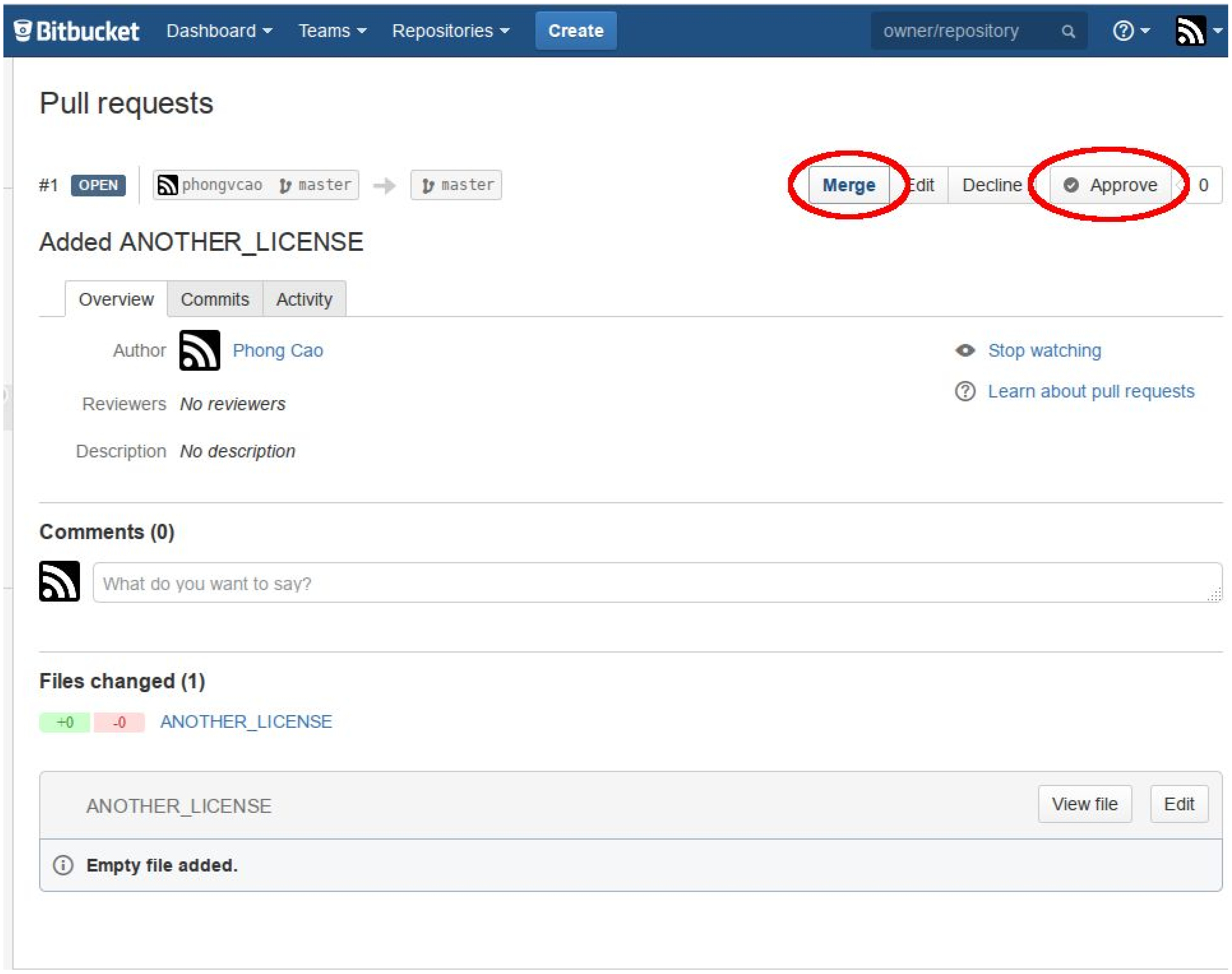
Diff | Commits

Author	Commit	Message	Date
Phong Cao	040ad8c	Added ANOTHER_LICENSE	16 minutes

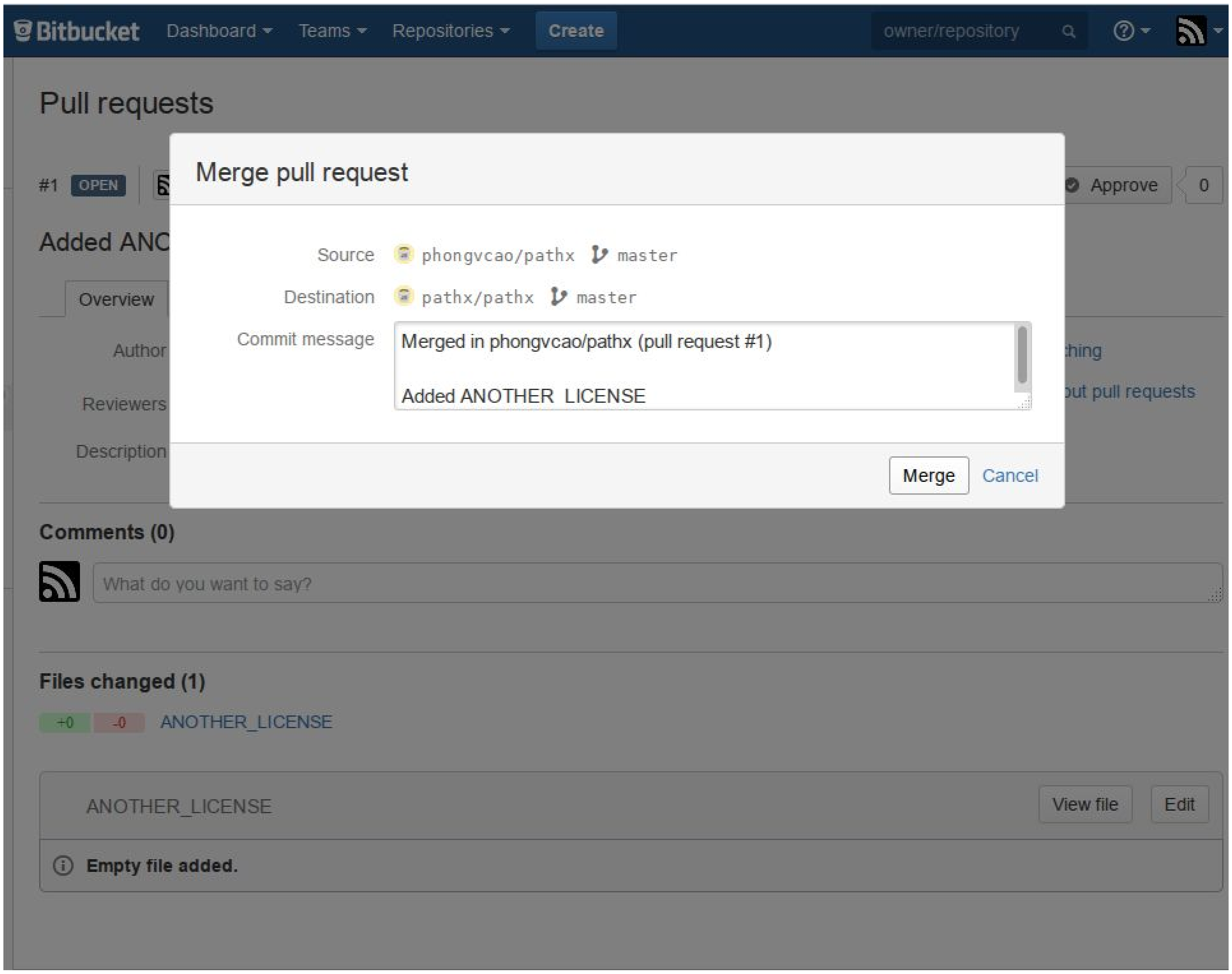
When you're done reviewing, click on "Create pull request" button to start the Pulling process from the team member's pathX fork repository to the official, **original** pathX repository.

Step 18: (For Administrators) Merge the Pull Requests

<To be edited and reviewed>
. After the above step is finished, Bitbucket will lead you to a page where you can edit information about the Git merge. Edit anything if needed, and when you're done please click on "Approve" and then "Merge" to start the Merging process:



6. Now you'll be presented a Merge dialog. Just click on "Merge" to finish the Merging process:



Merge pull request

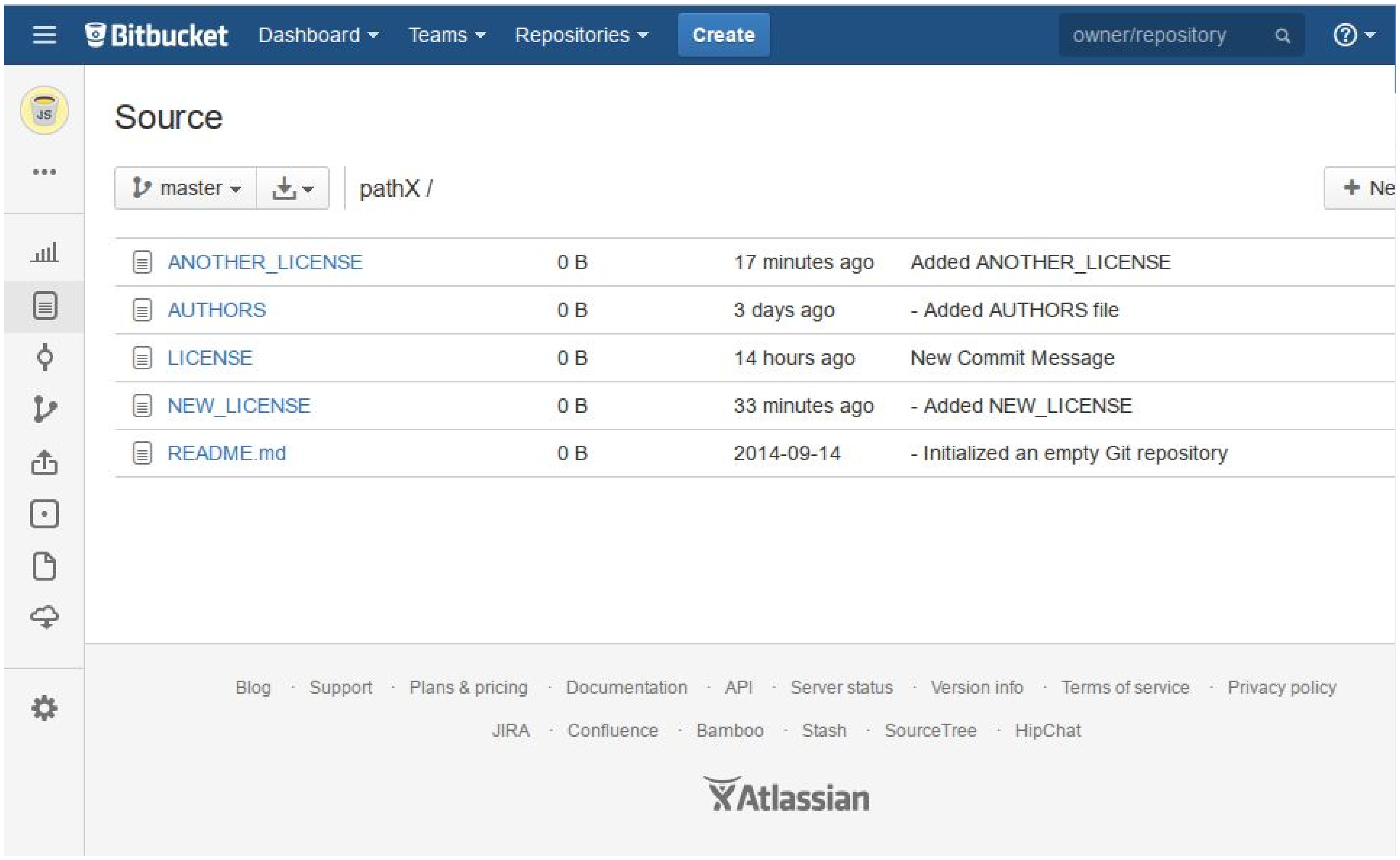
Source phongvcao/pathx master

Destination pathx/pathx master

Commit message

```
Merged in phongvcao/pathx (pull request #1)

Added ANOTHER_LICENSE
```

7. You're now finished! The team member's pathX fork repository has been successfully merged to the official, **original** pathX repo on Bitbucket. You should now notify other team members about the Merge so they know to update their local pathX fork repository accordingly (see Step 14):


Bitbucket
Dashboard
Teams
Repositories
Create
owner/repository
Source
master
pathX /
+ Ne
Blog · Support · Plans & pricing · Documentation · API · Server status · Version info · Terms of service · Privacy policy
JIRA · Confluence · Bamboo · Stash · SourceTree · HipChat
Atlassian

ANOTHER_LICENSE	0 B	17 minutes ago	Added ANOTHER_LICENSE
AUTHORS	0 B	3 days ago	- Added AUTHORS file
LICENSE	0 B	14 hours ago	New Commit Message
NEW_LICENSE	0 B	33 minutes ago	- Added NEW_LICENSE
README.md	0 B	2014-09-14	- Initialized an empty Git repository