**Lab 06**

**Laboratory Exercise**

**Part 1: Version Control using Git**

**Let take a look at the differences between Git, GitHub and GitLab. View this YouTube video.**

[**https://www.youtube.com/watch?v=21Gl97tkbHU&t=69s**](https://www.youtube.com/watch?v=21Gl97tkbHU&t=69s)

**LAB EXERCISE**

This lab will cover common Git fundamentals, as well as integrating with a GitHub repository to showcase Git processes in play.

**Time to Complete**

Approximately 30 Minutes

**What You Need**

You’ll need to create a new repository for this tutorial and make it available from your cluster through a URL from Git.

**GitHub Account**

1. Setup GitHub Account

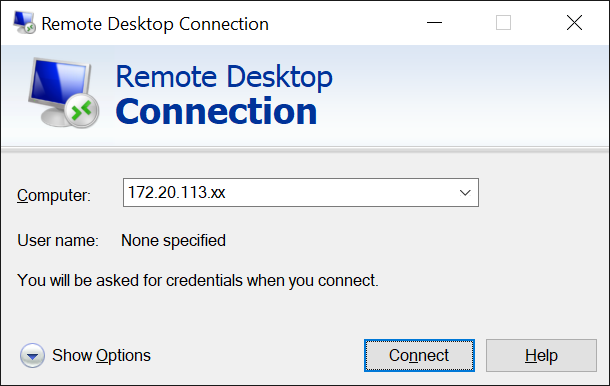
If you do not already have a GitHub account, create one by visiting <https://github.com/>

2. Sign up for an account with

email: your\_email  
 password: something secure that you will remember  
 username: your-username

From your machine logged-in to RP VPN, run Remote Desktop Connection to connect to the ubuntu Linux Virtual Machine (VM). Please login based on your assigned VM as shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S/N** | **Name** | **VM** | **IP Address** | User Name | Password |
| 1 | ABDUL SALIM BIN ABDUL RASHITH | LABC03 - 172.20.115.50 | 172.20.115.50 | soi-sddo | Republic01 |
| 2 | CASPER LEOW YU HAN (LIAO YU HANG) | LABC03 - 172.20.115.51 | 172.20.115.51 | soi-sddo | Republic01 |
| 3 | CHAN JUN ZHI, GLENN | LABC03 - 172.20.115.52 | 172.20.115.52 | soi-sddo | Republic01 |
| 4 | CHIA WAI TAT | LABC03 - 172.20.115.53 | 172.20.115.53 | soi-sddo | Republic01 |
| 5 | HOI WAI TECK | LABC03 - 172.20.115.54 | 172.20.115.54 | soi-sddo | Republic01 |
| 6 | KOH JIN CAI DAEMIAN | LABC03 - 172.20.115.55 | 172.20.115.55 | soi-sddo | Republic01 |
| 7 | KYAW KYAW OO | LABC03 - 172.20.115.56 | 172.20.115.56 | soi-sddo | Republic01 |
| 8 | LUM YOKE FAI | LABC03 - 172.20.115.57 | 172.20.115.57 | soi-sddo | Republic01 |
| 9 | MUHAMMAD FADHLI BIN MOHAMED NOOR | LABC03 - 172.20.115.58 | 172.20.115.58 | soi-sddo | Republic01 |
| 10 | MUHAMMAD HILMEE BIN MD ALI | LABC03 - 172.20.115.59 | 172.20.115.59 | soi-sddo | Republic01 |
| 11 | NG SAY WEE | LABC03 - 172.20.115.60 | 172.20.115.60 | soi-sddo | Republic01 |
| 12 | NGUI WEILY | LABC03 - 172.20.115.61 | 172.20.115.61 | soi-sddo | Republic01 |
| 13 | NU'MAN HARITH BIN NORRAIMI | LABC03 - 172.20.115.62 | 172.20.115.62 | soi-sddo | Republic01 |
| 14 | RULY JANUAR FACHMI | LABC03 - 172.20.115.63 | 172.20.115.63 | soi-sddo | Republic01 |
| 15 | SEAH SHIH WEI GEROME | LABC03 - 172.20.115.64 | 172.20.115.64 | soi-sddo | Republic01 |
| 16 | SEAN CHENG ZHI WEI | LABC03 - 172.20.115.65 | 172.20.115.65 | soi-sddo | Republic01 |
| 17 | SEY KOK SIONG | LABC03 - 172.20.115.66 | 172.20.115.66 | soi-sddo | Republic01 |
| 18 | TAN JOON YEE DOUGLAS | LABC03 - 172.20.115.67 | 172.20.115.67 | soi-sddo | Republic01 |
| 19 | WU WAI TENG VANESSA | LABC03 - 172.20.115.68 | 172.20.115.68 | soi-sddo | Republic01 |
| 20 | YAP KOON SING | LABC03 - 172.20.115.69 | 172.20.115.69 | soi-sddo | Republic01 |
| 21 | YE CHENG LIM | LABC03 - 172.20.115.70 | 172.20.115.70 | soi-sddo | Republic01 |
| 22 | SHAIFUL BIN ABDUL KARIM | LABC03 - 172.20.115.71 | 172.20.115.71 | soi-sddo | Republic01 |
| 23 | CHAI RU YI | LABC03 - 172.20.115.72 | 172.20.115.72 | soi-sddo | Republic01 |
| 24 | JWAY HWEE LING JULIE | LABC03 - 172.20.115.73 | 172.20.115.73 | soi-sddo | Republic01 |
| 25 | SAMANTHA TEO XING YEE | LABC03 - 172.20.115.74 | 172.20.115.74 | soi-sddo | Republic01 |
| 26 | ZIL AZZA HILMIAH BINTE RADUAN | LABC03 - 172.20.115.75 | 172.20.115.75 | soi-sddo | Republic01 |



Replace xx with the IP address of the VM that you have been assigned.

**First time Git setup**

3. Set up your identity. Especially important when you work with other people:

git config --global user.name "Keyon Genesis Kanan”

git config --global user.email [xxxxx@rp.edu.sg](mailto:xxxxx@rp.edu.sg)

4. Check your settings:

git config --list

5. Getting help from Git

git help config

**or**

git config –help

**or**

man git-config

**Git Basic Usage**

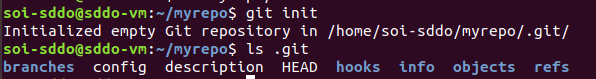
A directory (.git/) contains all information regarding the history of your code history.

6. Creating a Git repository from an existing directory

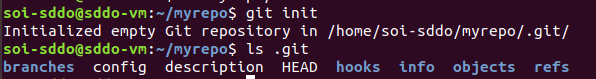
mkdir myrepo

cd myrepo

git init



Ls .git



**Create a readme.txt for the repo**

7. Create a readme file in the directory ~/myrepo

nano readme.txt

7.1. Add the below two lines of text (using your favorite editor: vi, emacs or nano) to “readme.txt”:

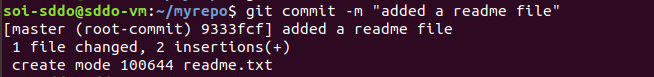
Git is a version control system.   
 Git is free software.

7.2. First add the file to the repository

git add readme.txt

7.3. Commit the file to the repository

git commit -m "added a readme file"



Some explanations

* git add: add the readme.txt to the staging area (index)
* git commit -m: commit your changes to the repo with a message (“-m”)

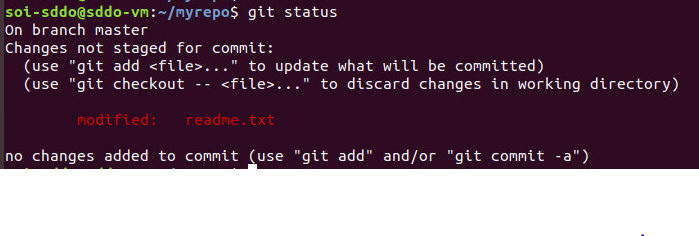
**View New Changes**

8. Now change “readme.txt” using nano readme.txt to the below contents:

Git is a ***distributed*** version control system.   
 Git is free software.

8.1. Use git status to check our results:

git status



8.2. If you do not remember the changes, use git diff

git diff

|  |
| --- |
| <Insert screen capture of results> |

9. Commit new changes to the repository. Same as the previous, two steps, git add and then git commit:

git add readme.txt

git status

|  |
| --- |
| <Insert screen capture of results> |

git commit -m "added word distributed"

git status

**Commit another change to the repository**

10. Now change “readme.txt” nano readme.txt to the below contents:  
 Git is a distributed version control system.   
 Git is free software distributed under the GPL.

10.1. Add and commit your changes:

nano readme.txt

git add readme.txt

git commit -m "appended GPL"

So far, we have 3 versions added to the Git repository “myrepo”

**Review the version history**

11. Use git log to review our version history

git log

|  |
| --- |
| <Insert screen capture of results> |

Or you can use git log --oneline for a short version

git log --oneline

**Visualizing branch topology in Git**

12. First, add a useful git command alias by copy & pasting the following command in your terminal:

git config --global alias.graph 'log --all --oneline --decorate --graph'

12.1 Then type “git graph” in your terminal:

git graph

|  |
| --- |
| <Insert screen capture of results> |

We will explain the detailed meanings of the command later

**Jump to previous versions**13. Go back to the version “added word distributed”

git graph

git reset --hard HEAD

Or you can directly use the SHA-1 hash

13.1 Then you can take a look at the content of readme.txt:

cat readme.txt

|  |
| --- |
| <Insert screen capture of results> |

**What if the previous operation is a mistake?**

14.How do I go back to the latest version?

git graph

Hint: we need to find the SHA-1 hash of that commit

15. git reflog

|  |
| --- |
| <Insert screen capture of results> |

16. Now we can reset to the specific commit (SHA-1 hash):

git reset --hard <hash\_no>

cat readme.txt

|  |
| --- |
| <Insert screen capture of results> |

**Part 2: Basic Git workflow**

**LAB EXERCISE**

This lab will cover three main sections of Git: Working Directory, Staging Area and Repository.

The Git directory (.git/ directory) where Git stores the metadata and object database for your project.

The working directory (working tree) is a single checkout (snapshot) of one version of the project, i.e. the working directory consists of files that you are currently working on (you see).

The staging area is a file that stores information about what will go into your next commit. It’s sometimes referred to as the "index", but it’s also common to refer to it as the staging area.

**Time to Complete**

Approximately 30 Minutes

**What You Need**

You’ll need a git repository for this tutorial.

**Staging area**

1. Add a new file (license.txt, “content can be arbitrary”) to the repository:

2. Add the below line to the readme.txt:

Git has a mutable index called stage.

3. Then use the git status to check our repo status

git status

|  |
| --- |
| <Insert screen capture of results> |

4. Now we only add license.txt to the staging area:

git add license.txt

git commit -m "add license"

4.1 Then check the working directory status

git status

|  |
| --- |
| <Insert screen capture of results> |

The change of readme.txt is not committed, why?

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

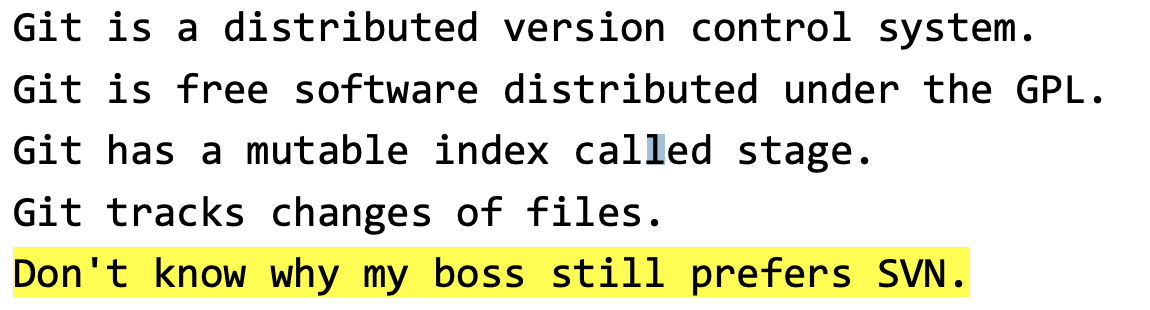
**Managing the changes**

5.1. Make the following changes to readme.txt and capture the results

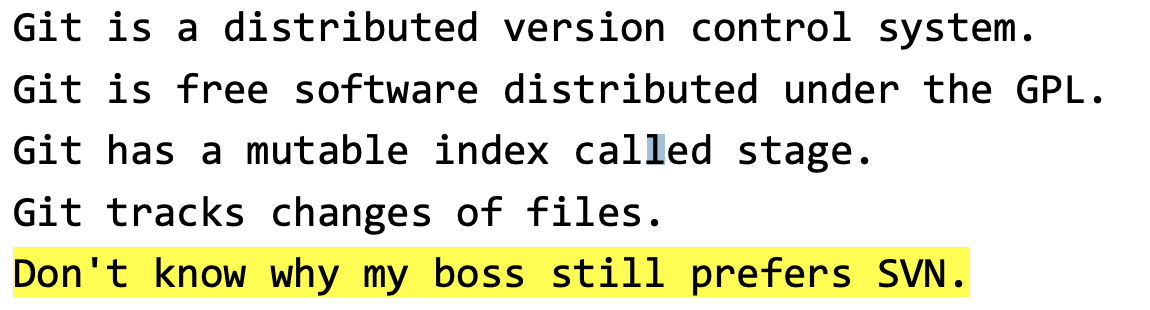
|  |  |
| --- | --- |
| **Requirements:** | **Solutions:** |
| Add a newline to readme.txt: Git tracks changes. |  |
| Add readme.txt to staging area (to discard changes in working directory)  Hint:  1) git checkout  2) cat <filename> | git checkout -- readme.txt  cat readme.txt |
| Change the last line of readme.txt to: Git tracks changes of files.  Hint:  1) nano <filename>  2) git reset HEAD  2) git checkout 3) cat <filename> | nano readme.txt  git reset HEAD readme.txt  git checkout -- readme.txt  cat readme.txt |

5.2 **Undoing the changes**

5.3 Make the following changes to readme.txt from the below image and commit the changes.



5.3 Add the last line to readme.txt from the below image and commit the changes.



5.4. How to cancel a change? e.g. what if you need to delete the last line of the filename “readme.txt” ?

5.4.1 Do screen capture the three possible situations and fill in the blank:

|  |  |  |
| --- | --- | --- |
| **Possible situations:** | **Solutions** | **Fill in the blank:** |
| Discard the changes in the working directory. | git checkout -- readme.txt  cat readme.txt | <insert image> |
| Discard the changes added to the staging area. (2 steps) | git reset HEAD readme.txt  git checkout -- readme.txt | <insert image> |
| Discard the changes that are already committed | git reset --hard <commit-hash-no> | <insert image> |

5.3 **Working with Git branch**

The scenario:

* Need to develop a new feature
* The new feature will interfere with the current functions
* Need to let the new feature separate from the main branch (dev branch)

5.3.1 Create a dev branch and switch to that branch

git branch

git checkout -b dev

git branch

|  |
| --- |
| <Insert screen capture of results> |

5.3.2 On dev branch, modify *readme.txt* by adding a line:

nano readme.txt

Creating a new branch is quick.

5.3.3 Commit the changes:

git add readme.txt

git commit -m "branch test"

cat readme.txt

|  |
| --- |
| <Insert screen capture of results> |

5.3.4 Switch to the master branch and check the readme.txt

git checkout master

cat readme.txt

|  |
| --- |
| <Insert screen capture of results> |

This verifies the change happens only on the dev branch

5.3.5 Merge the work on the dev branch to the master branch:

git branch

git graph # a predefined alias

|  |
| --- |
| <Insert screen capture of results> |

git merge dev

|  |
| --- |
| <Insert screen capture of results> |

5.3.6 Verify the branch status using our predefined command alias

git graph

|  |
| --- |
| <Insert screen capture of results> |

5.4 **Git branch Merge and conflict**

5.4.1 In dev branch, change the last line in *readme.txt* to:

Creating a new branch is quick AND simple.

git checkout -b dev

nano readme.txt # change last line to “AND simple”

git add readme.txt

git commit -m "AND simple"

|  |
| --- |
| <Insert screen capture of results> |

5.4.2 In master branch, change the last line in *readme.txt* to:

Creating a new branch is quick & simple.

git checkout master

nano readme.txt # change last line to “& simple”

git add readme.txt

git commit -m "& simple"

|  |
| --- |
| <Insert screen capture of results> |

5.4.3 Merge both the branch

git merge dev

|  |
| --- |
| <Insert screen capture of results> |

5.5 **Resolve conflict**

5.5.1 Use git status to check the conflict files

git status

5.5.2 Use your favorite editor to check the contents of readme.txt

cat readme.txt

|  |
| --- |
| <Insert screen capture of results> |

5.5.3 Manually resolve the conflicts indicated by Git (remove the following lines):

<<<<<<< HEAD

=======

>>>>>>> dev

nano readme.txt

* Remove the following lines mentioned above.

Think of these new lines as "conflict dividers". The ======= line is the "center" of the conflict. All the content between the center and the <<<<<<< HEAD line is content that exists in the current branch main which the HEAD ref is pointing to.

Alternatively all content between the center and >>>>>>> dev is content that is present in our merging branch.

5.5.4 We will resolve this conflict by changing the last line to:

Creating a new branch is quick and simple.

5.5.5 Commit your changes to complete the merge process.

git add readme.txt

git status

|  |
| --- |
| <Insert screen capture of results> |

git commit -m "resolve conflict"

6. **Verify git status and delete dev branch**

6.1 Verify git branch graph

git graph

|  |
| --- |
| <Insert screen capture of results> |

6.2 Delete the dev branch

git branch -d dev  
 git graph

|  |
| --- |
| <Insert screen capture of results> |

Summary:

* Master branch is always stable
* All development work in dev branch, merge to master when necessary
* Every developer has his/her own branch.

**Part 3: Working with remote repository**

**LAB EXERCISE**

This lab will cover Remote repositories that are versions of your project that are not on your computer.

You will learn the interaction with remote repositories

– You push changes from your computer to the remote.

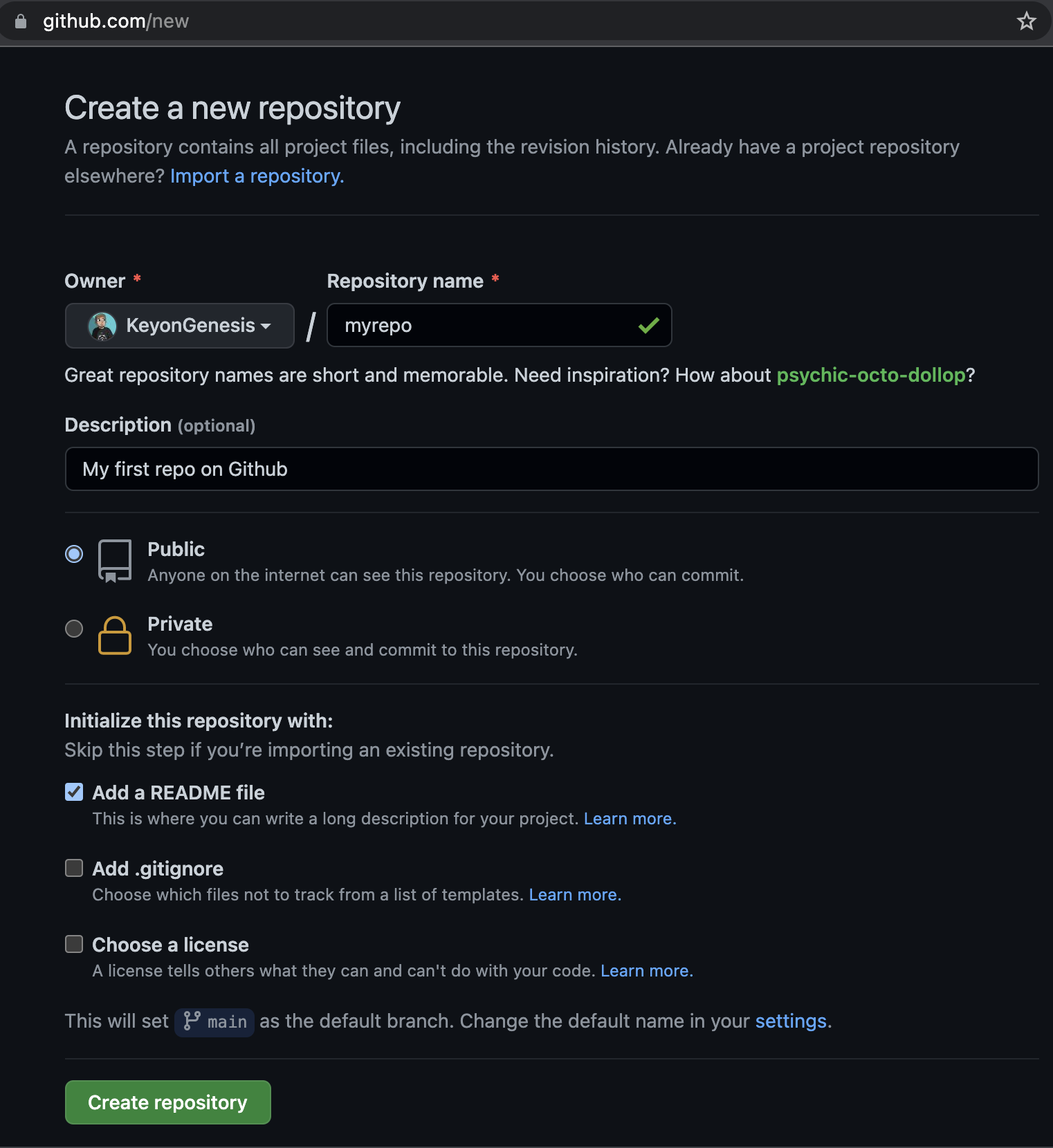
– You pull changes from the remote to your computer.

**Time to Complete**

Approximately 30 Minutes

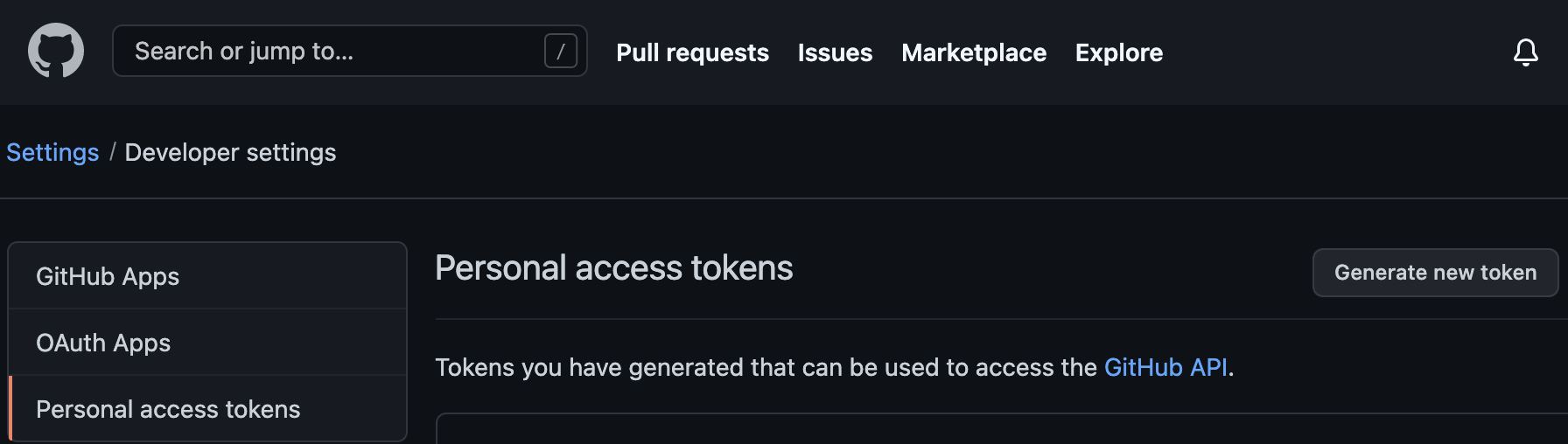
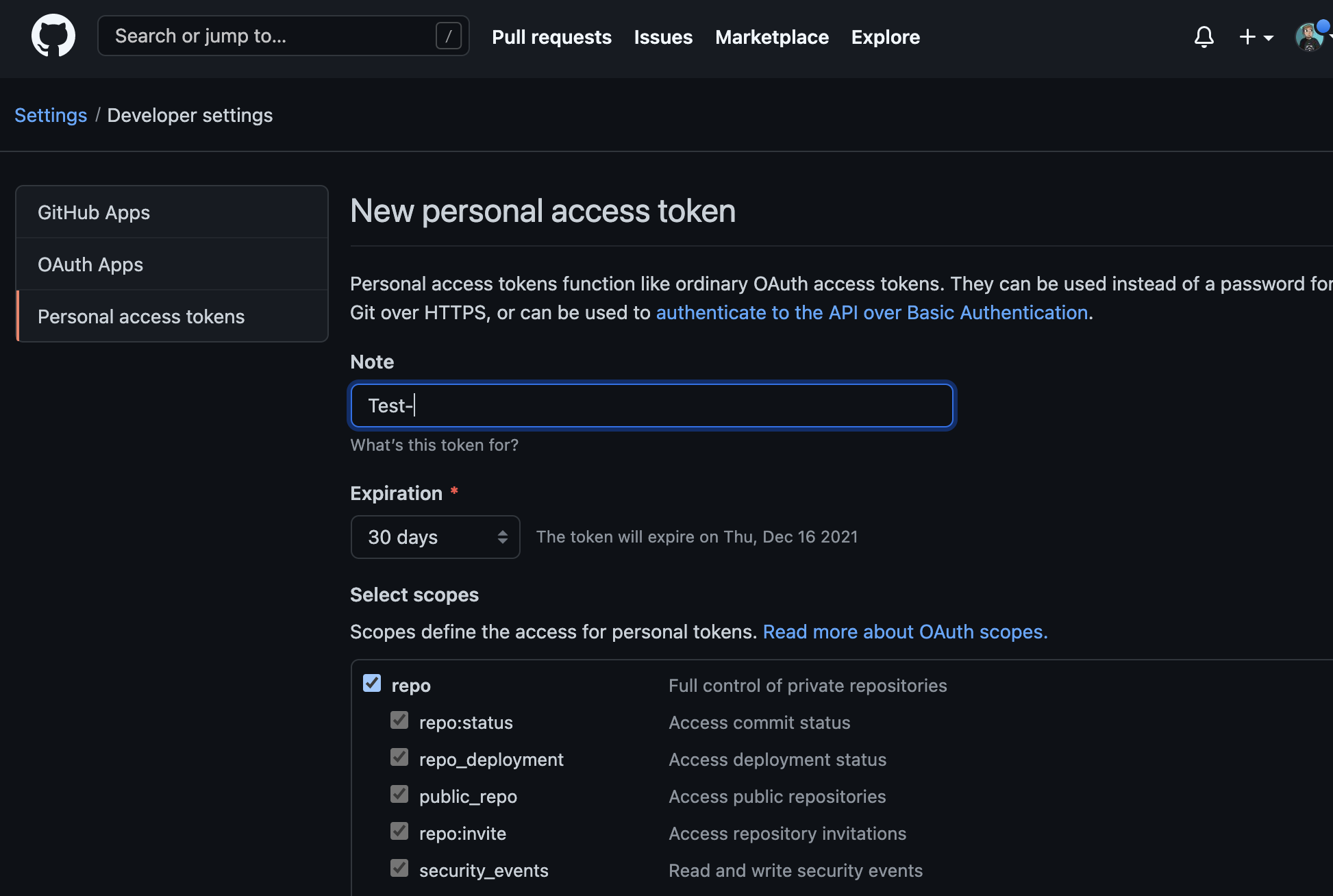
**What You Need**

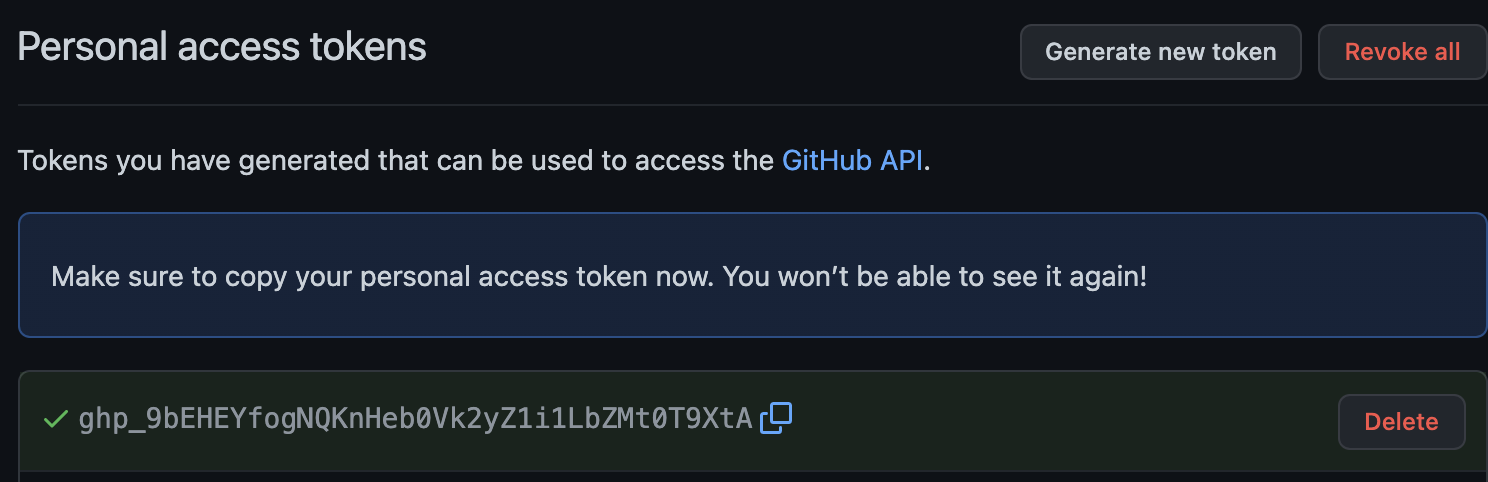
You’ll need a git account for this tutorial.

1. Login to GitHub using the same credentials for Part 1  
1.1 Start a project on GitHub  
1.2 Create a new repository  
 

1.3 Push your existing repository

1.4 Github now uses personal token

• Click the profile icon and find the settings menu   
 • In the profile settings page, scroll down to “developer settings”  
 • Click “Personal access tokens”   
 • And then “Generate new token”  
   
  
1.4.1. Give a name for the note to the access token  
1.4.2. Check the “repo” box   
   
1.4.3. “Generate token”

• Copy your token to a safe place, e.g. notepad   
 • You won’t see the token again   
 • This token will be the password when you push/pull your repository  
 

1.5 Push your current repository to GitHub

Copy and paste the commands in the previous parts to push your local master branch to GitHub.

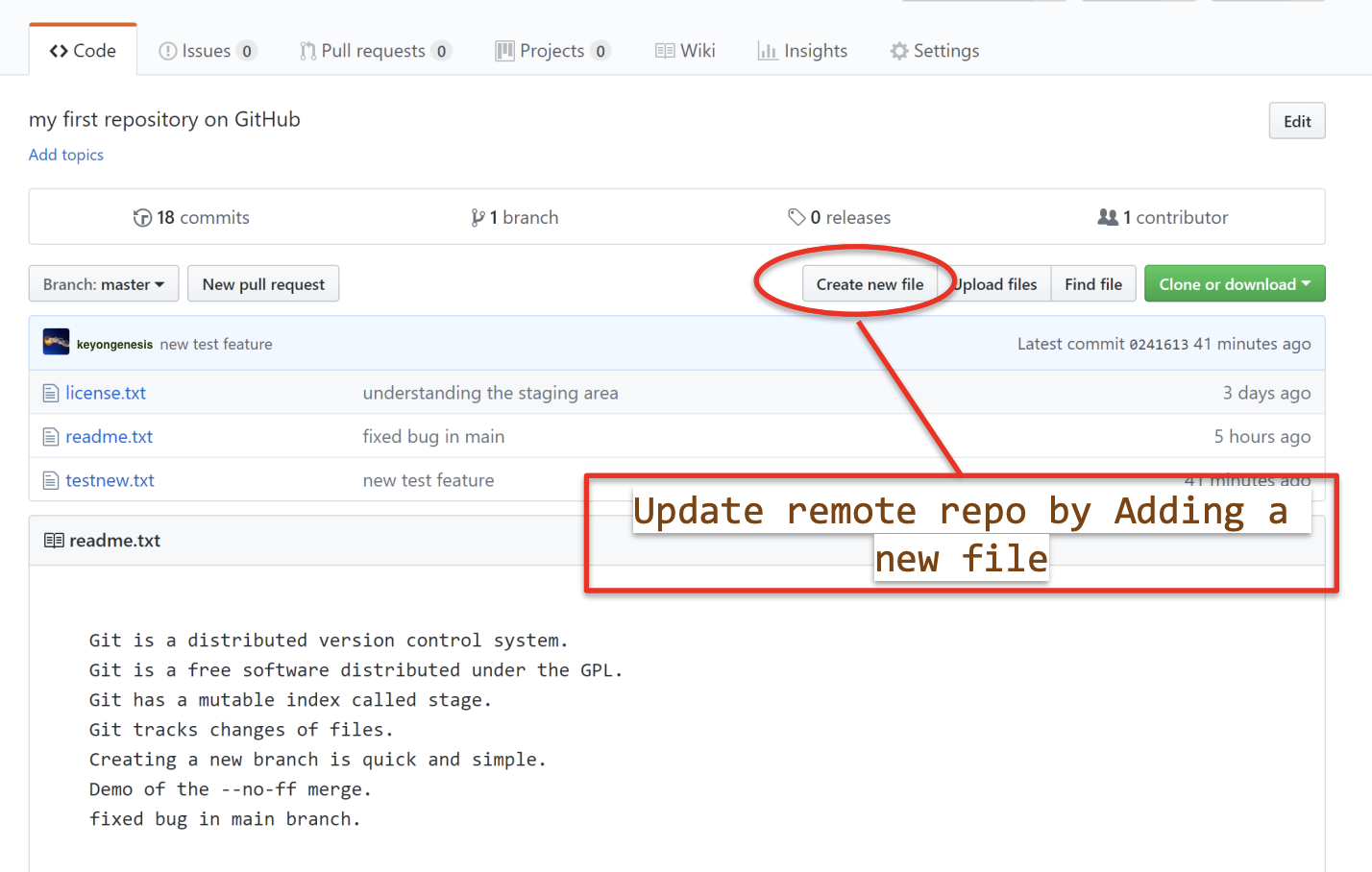
git remote add origin https://github.com/KeyonGenesis/myrepo.git  
 git push -u origin master  
 Username for '<https://github.com>': Enter your Username:   
 Password for '<https://keyongenesis@github.com>':  
 Enter or paste your access token when prompted to enter password

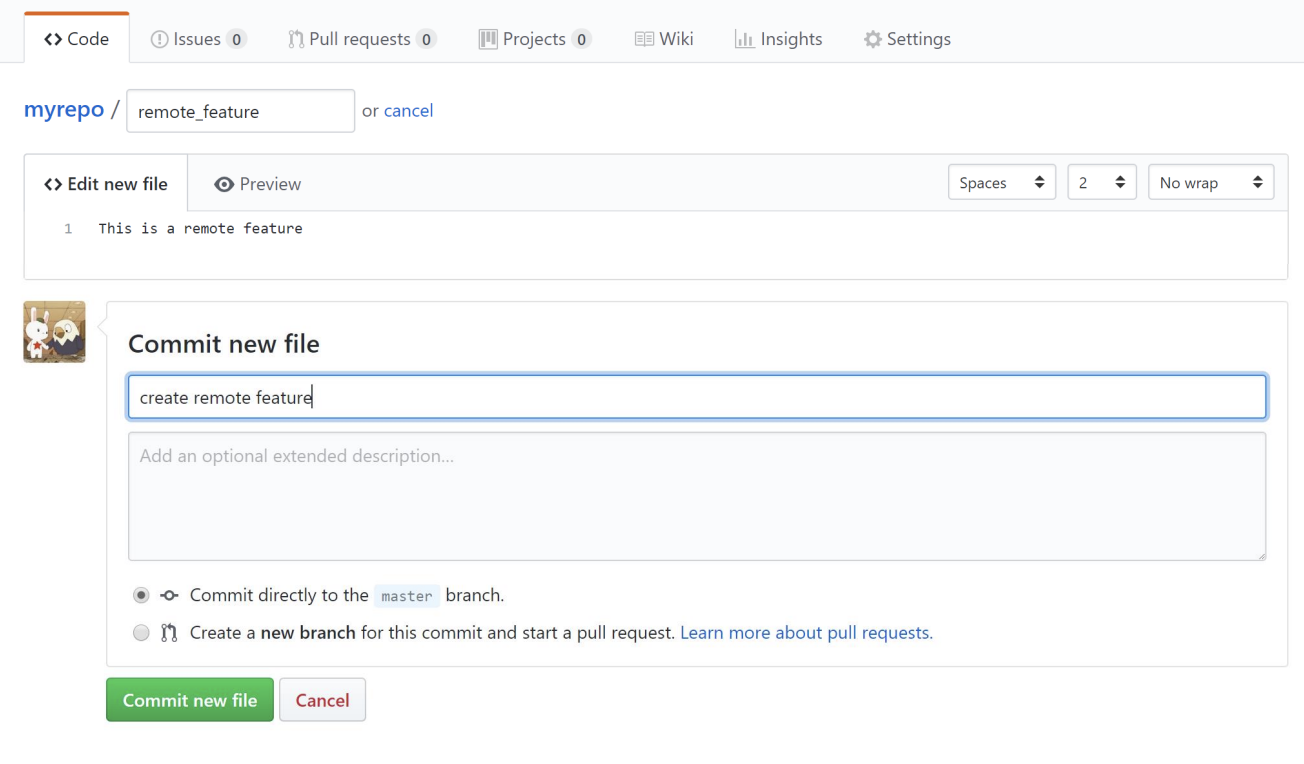
|  |
| --- |
| <Insert screen capture of results> |

git status

|  |
| --- |
| <Insert screen capture of results> |

1.6 View your remote GitHub repository



1.7 Update remote repository  
 

1.8 Updating your local copy of the remote branches

Use **git fetch** + **git merge**

* This is equivalent to "git pull"
* git pull = git fetch + git merge

git fetch  
 git merge # merge the remote branch with local

|  |
| --- |
| <Insert screen capture of results> |

1.9 Showing remote repositories

git remote -v

|  |
| --- |
| <Insert screen capture of results> |

**Summary:**

Git remote repository

* Push your local repo to the remote repo
* Update your local repo from the remote repo

**--End of Lab Exercise --**