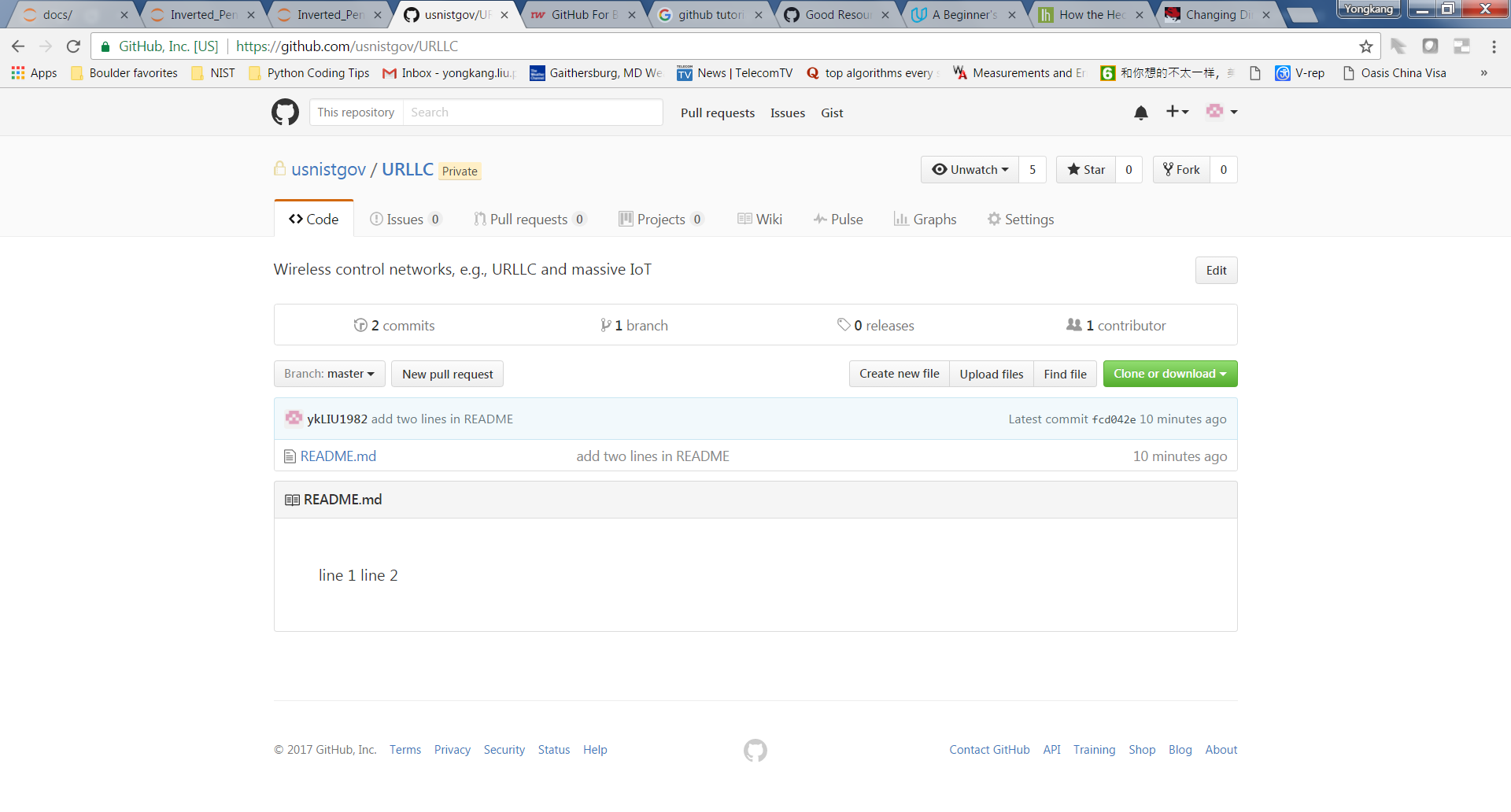
Steps of Using GitHub

Use Scenario 1: There is an online repository in the GitHub, we would like to download all files in it and use in the local computer.

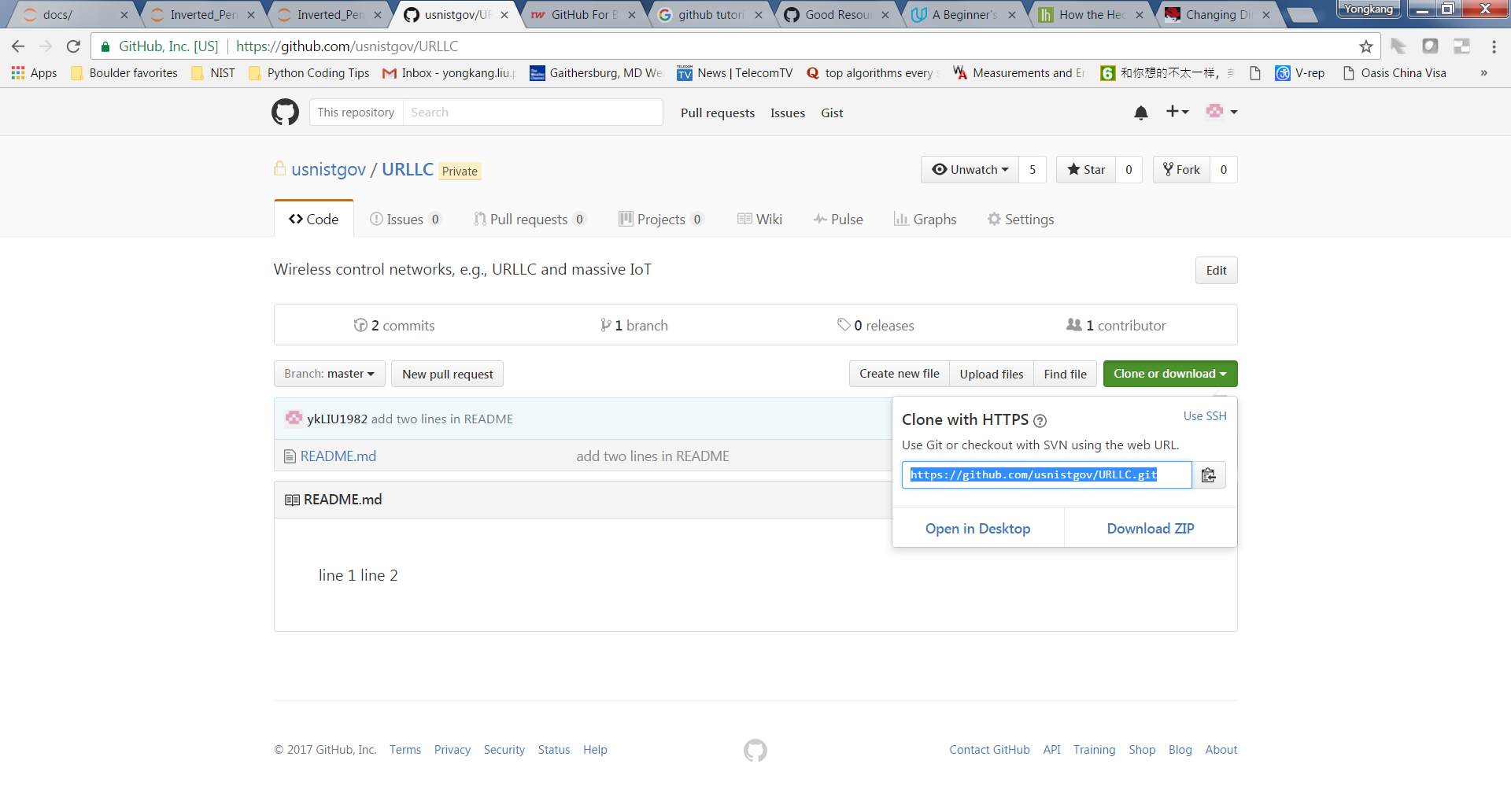
Solution: Using “Clone” command.

Steps:

1. Get the repository URL from the GitHub, which is ended with “.git”

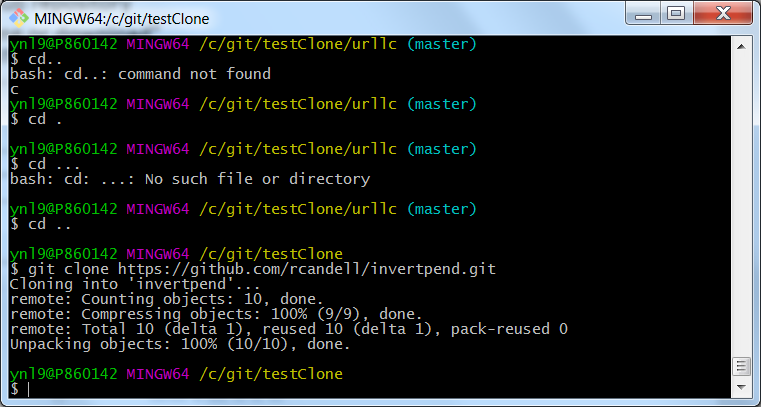


* 1. Enter the webpage of the GitHub repository, e.g., usnistgov/URLLC
  2. Press the green button of “Clone or download”



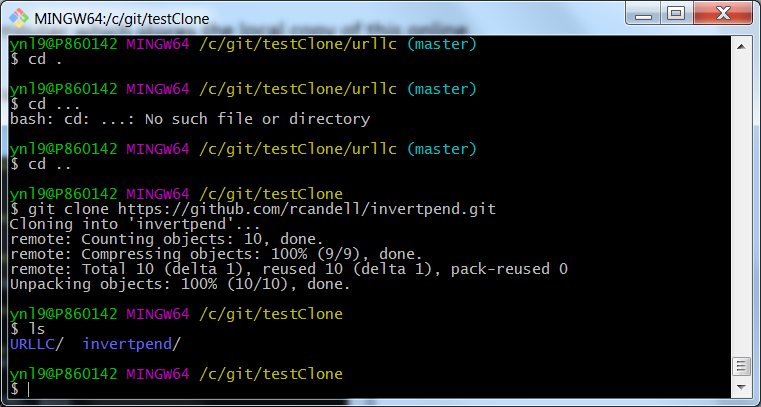
* 1. Copy the URL into clickboard, i.e., https://github.com/rcandell/invertpend.git

1. Select the parent folder in the local computer which stores the local copy of this online repository
   1. Change directory to the designated location, e.g., cd C:/git/testClone

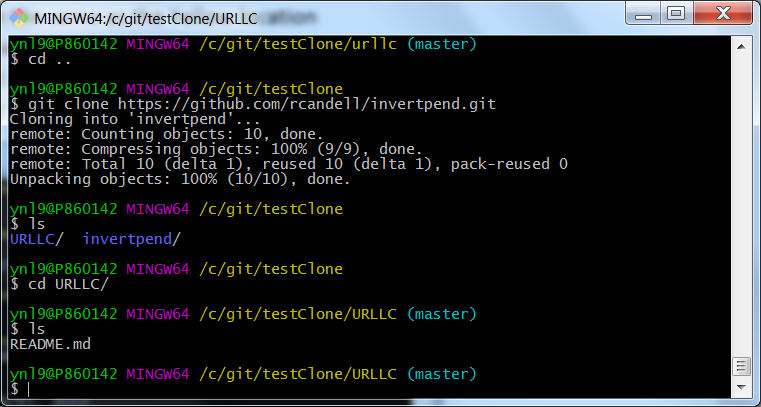


* 1. Make sure the current location the same as the Yellow location

1. Type in the Git Bash: git clone [the copied URL]
2. All files in this repository will be stored in the folder named “URLLC” within testClone
   1. Type “ls”, it will list all files and folders under testClone



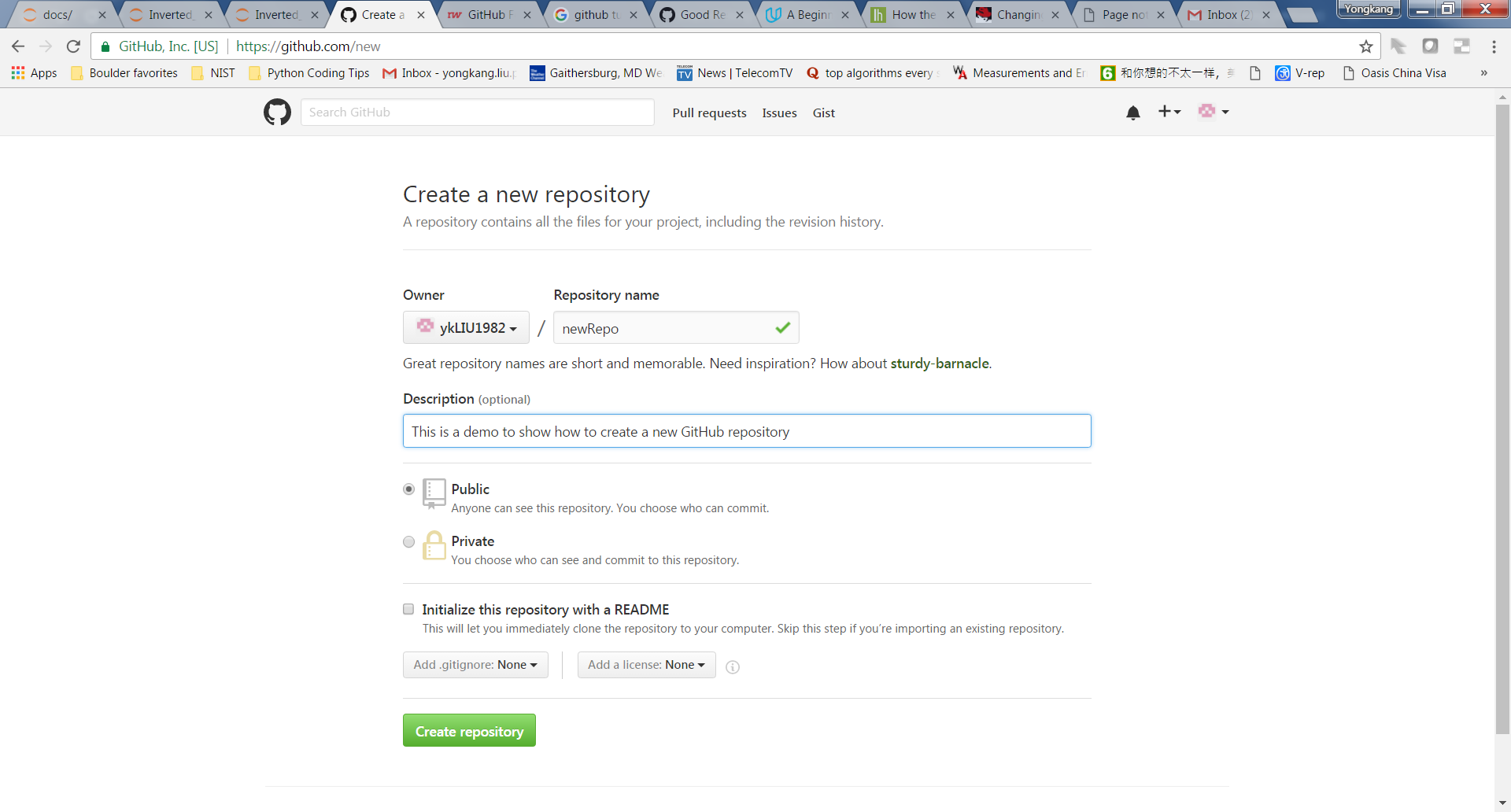
* 1. Enter the folder, type “ls” again, it will list all files in testClone which are the same as the files in the online repository of “URLLC”



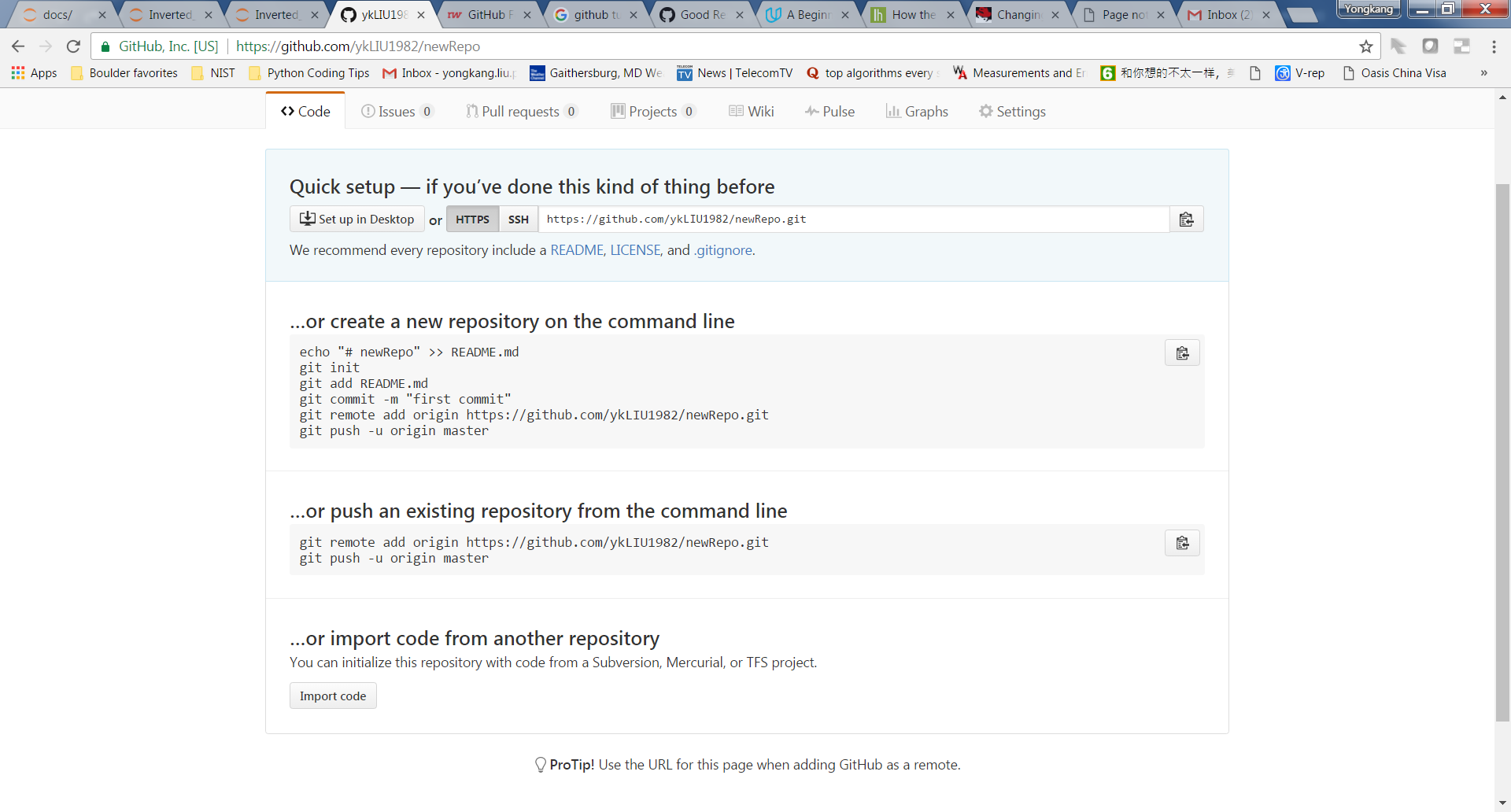
Use Scenario 2: We would like to create an empty GitHub online repository and synchronize it with the copy in the local computer

Steps:

1. Create a new repository online in the GitHub, e.g., “newRepo”

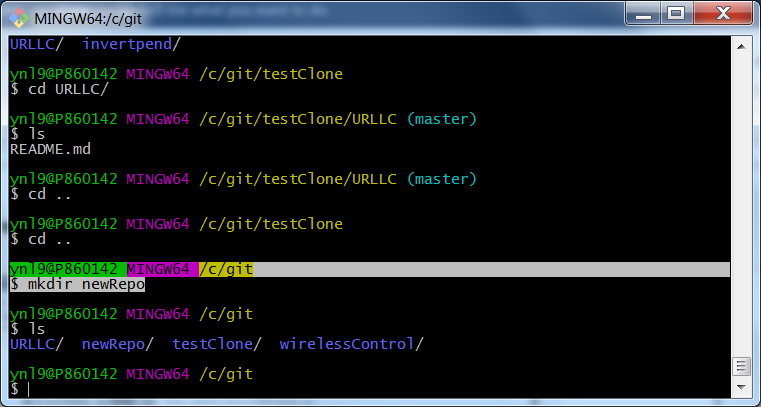


1. Once it is created, the GitHub shows several ways to set it up

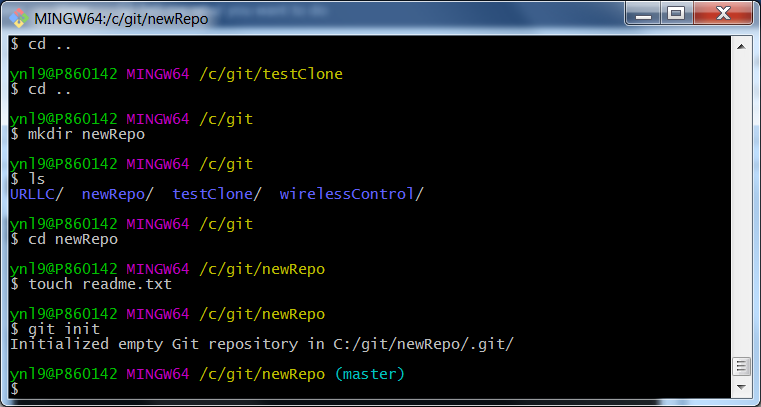


1. As we create the local mirror folder from scratch, then create an empty directory in the local computer, e.g., C:/Git/newRepo

Make a directory with the same name as “newRepo”

`

1. Since the current directory “C:/Git/newRepo” is just a local folder in the computer, we need to make it known to the Git
   1. Type “git init” when we are in this directory

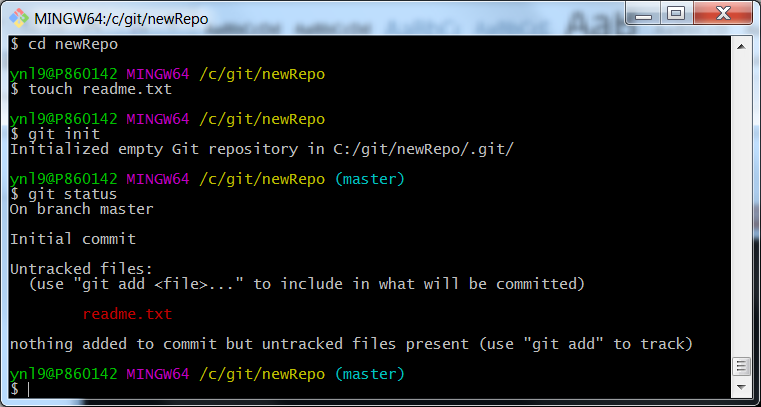


The Git then knows this repository and will be able to manage it later.

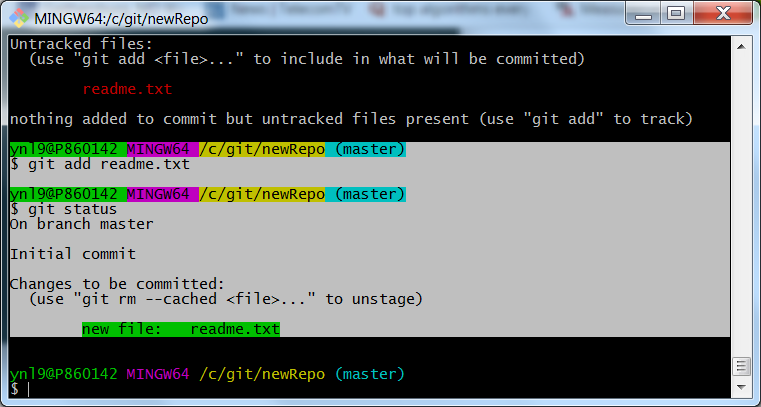
User scenario 3: Work on a local copy of the Git repository, make changes, stage them, commit them and push the changes back to the online repository.

Solution: In this scenario, it shows the normal operations on the files in an up-to-date local directory (no matter it is created locally or cloned from the online repository) and synchronize the changes with the online repository after the changes. The changes may be creating a new file, some modifications on the existing one

1. Make some changes in this directory. Enter this repository, create a “readme.txt”, e.g., in the command line, type “touch readme.txt”, then an empty txt file is created with the name of “readme”
   1. Type “git status” to check the status of current directory in Git



* 1. As the file “readme.txt” has never be tracked by Git since its creation, we first “stage” this untracked file. “git add readme.txt”



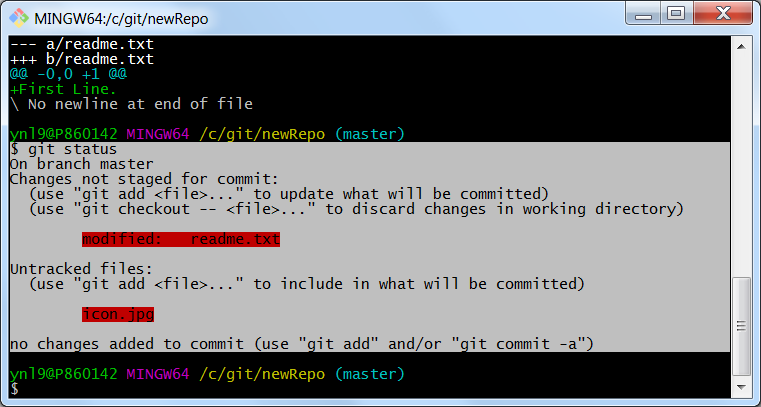
Use “git status”, it will show the updated status.

* 1. Commit the change to the Git. Committing is to add a meaningful commit message to Git, the version control system, to explain why the change was made.

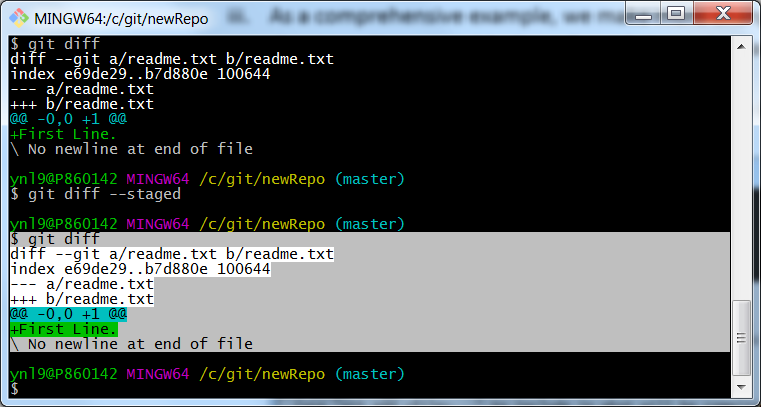
git commit -m”[the commit message]”



* 1. As a comprehensive example, we make some more changes to the directory.
     1. First, we add a line of words in readme and create another file, e.g, an image of “icon.jpg”. Use “git status”, we can see the status of each file in the directory

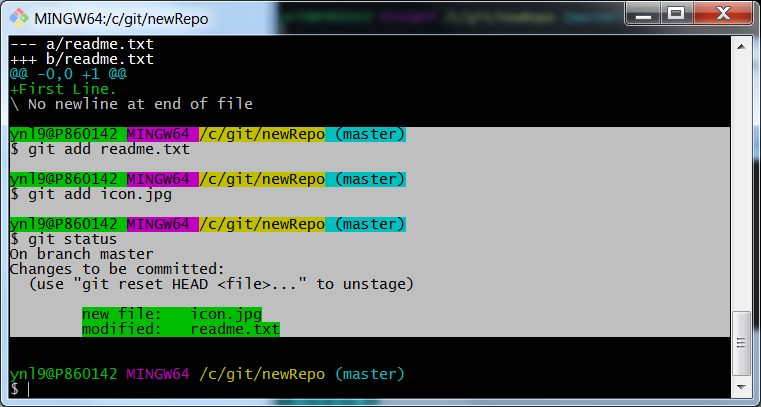


Use “git diff” can compare the difference between the staged file and the latest version.

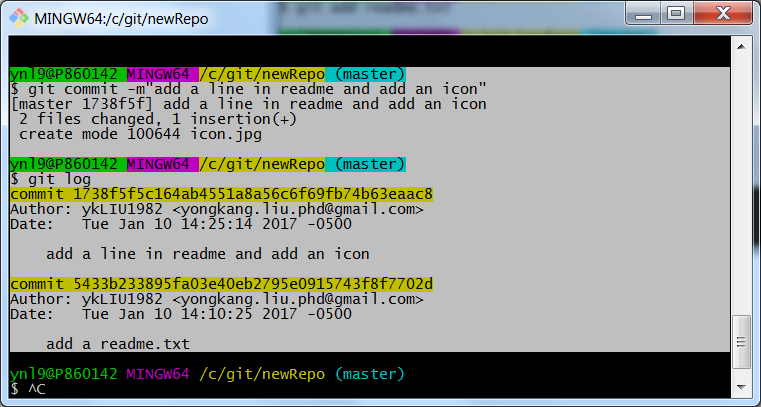


* + 1. Add these changes in Git

“git add readme.txt” and “git add icon.jpg”, respectively

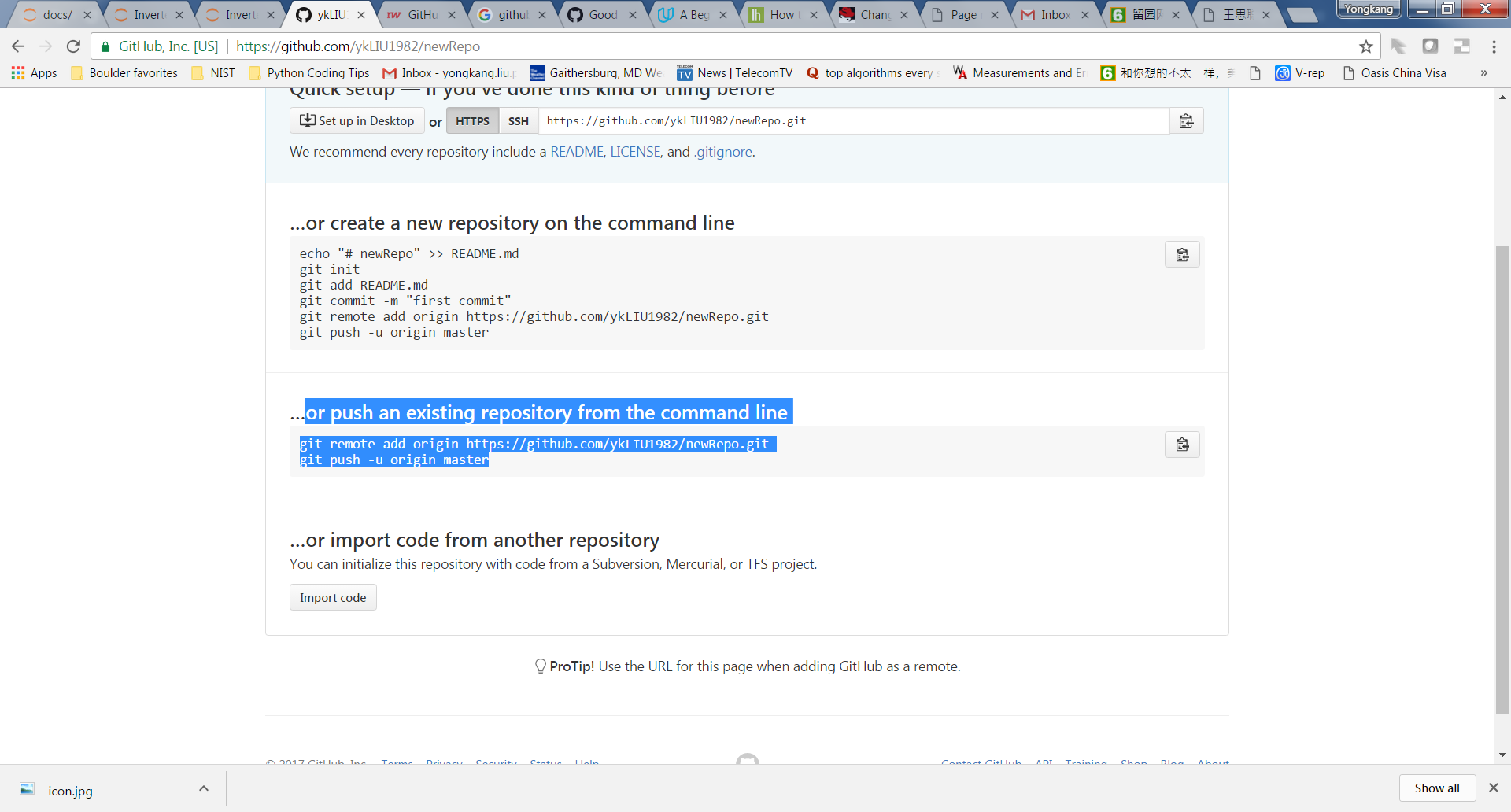


* + 1. Commit the changes



* + 1. Using “git log”, we can get the history of the tracked changes with the commit messages

1. Once all the changes have been added to Git and committed. We will push these changes to the GitHub repository and update the files online. As GitHub suggested, type the two lines of commands.



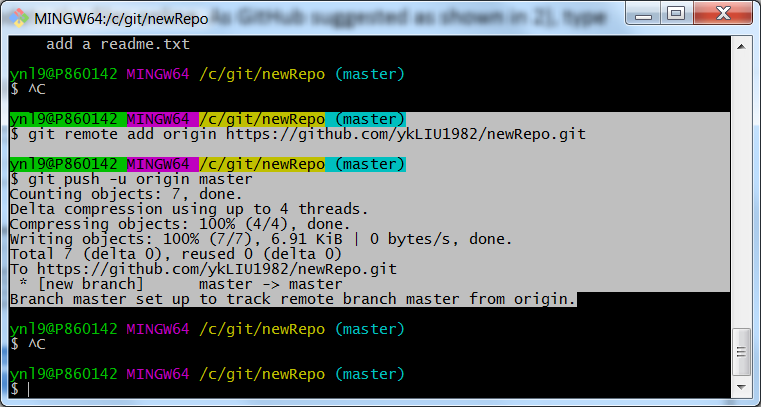
$ git remote add origin <https://githhub.com/ykLIU1982/newRepo.git>

This one set that your local repository is the remote of the GitHub Repository at <https://githhub.com/ykLIU1982/newRepo.git>, which is known/referred as “origin” in the following command lines.

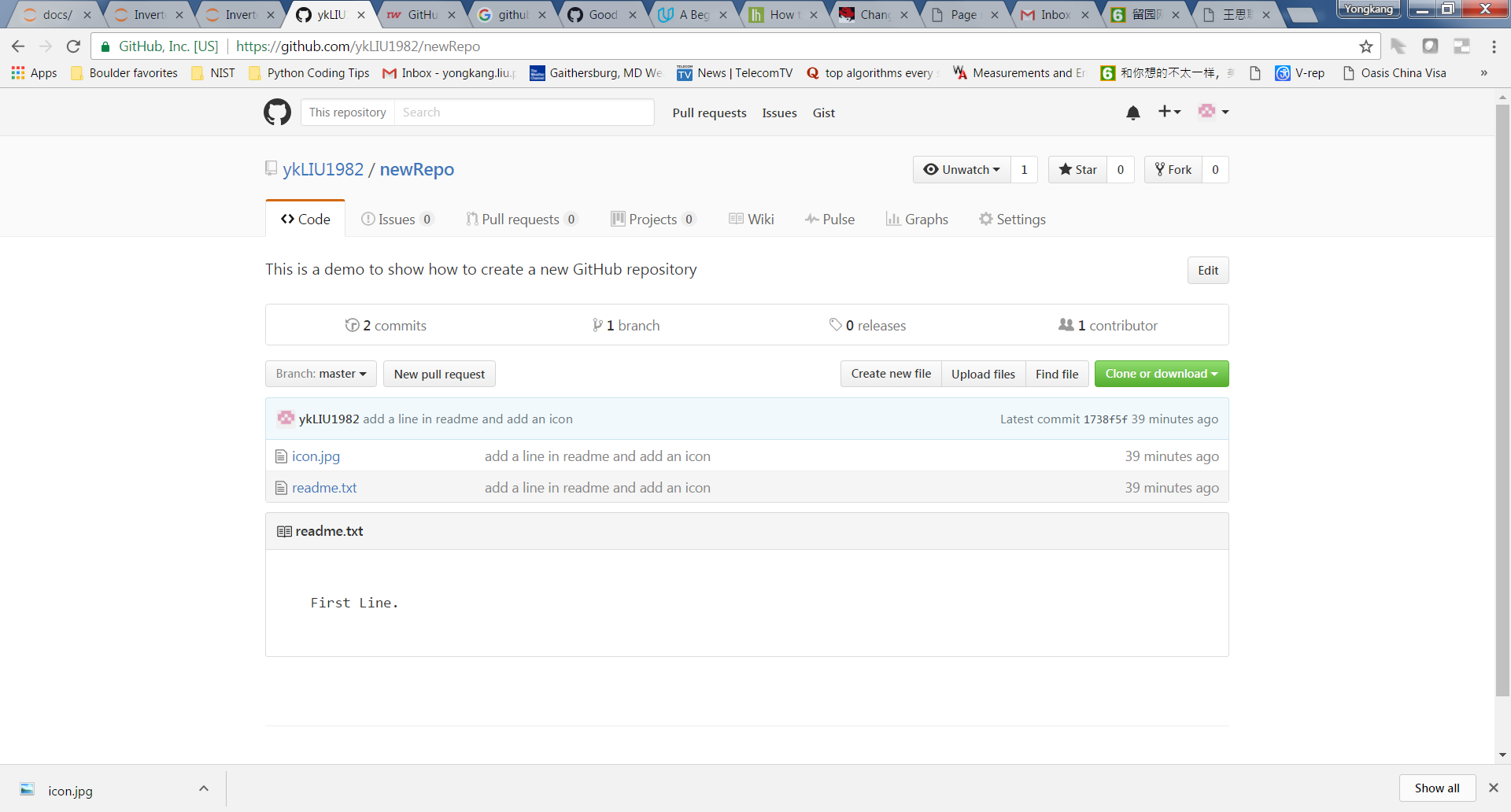
$ git remote -v

We can use this command to double check if the Git information stored in your local repository is the one expected.

$ git push -u origin master



We can then check the files online which are the same as the ones in the local computer.

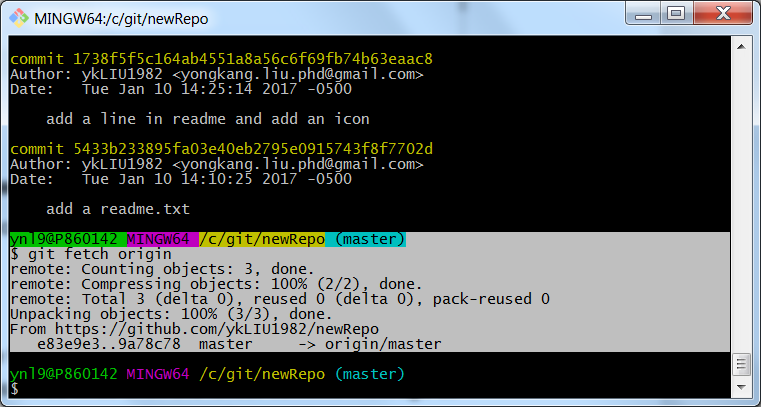


Use Scenario 4: Check the difference between the local directory and the online repository, and synchronize to the most up-to-date version

Steps:

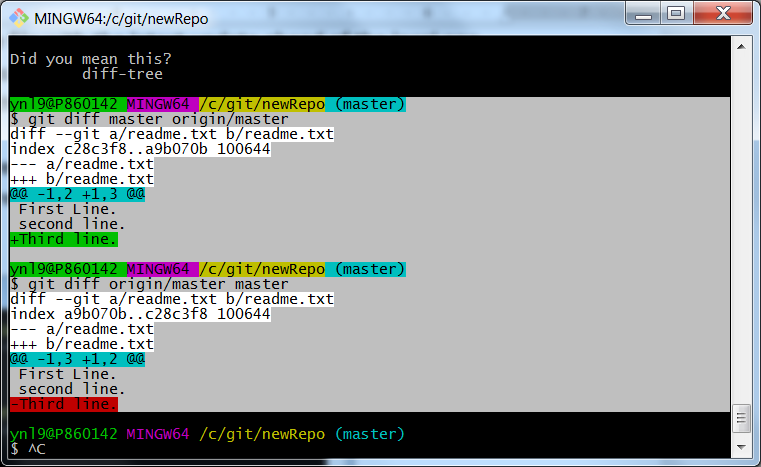
1. Suppose that the online repository has some file with the latest update ahead of the local one. E.g., the readme.txt in GitHub’s newRepo has three lines with the latest commit message as “Add third line” while in the local newRepo the readme.txt has two lines with the latest commit message as “Add second line”.
2. Before using git pull, we can check the difference between the local and online change history
   1. First get a remote called origin that refers to the GitHub repository, i.e., newRepo@GitHub

$ git fetch origin



* 1. Then see the difference between the local master and the one on GitHub

$ git diff master origin/master

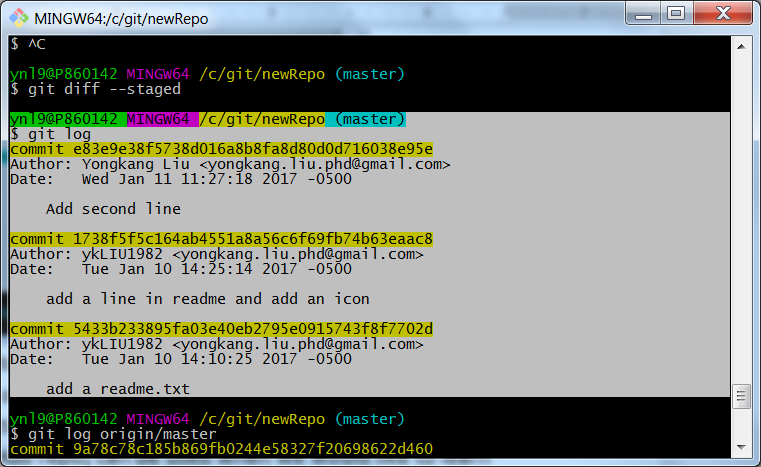


Remark: the order of the two repositories is critical. As shown above, $ git diff master origin/master gives the new changes (commits) on GitHub which are not in local; the reverse one gives the local changes/commits (the local tracked changes) that have not been pushed to GitHub.

Therefore, $ git diff [local repo] [GitHub repo] can be used when we would like to learn any new changes at GitHub that not merged into the local (the changes made by our coworkers after our last change); $ git diff [GitHub repo] [local repo] can be used when we make some changes in the local and would like to learn the difference with the GitHub repo (for verification purpose before we push the changes to GitHub)

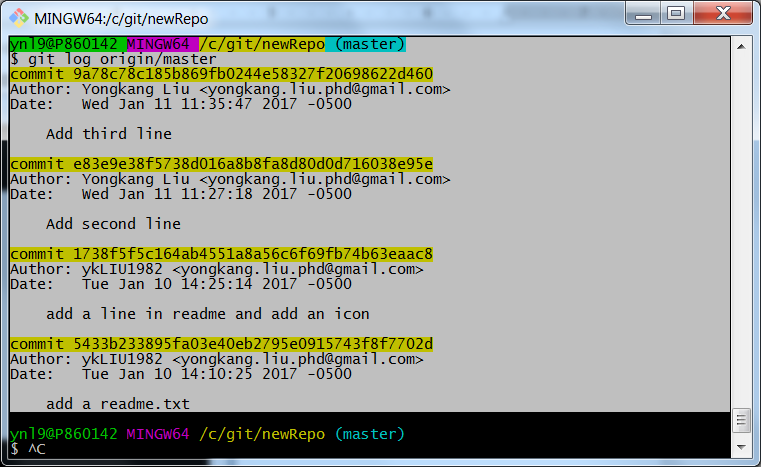
1. The second way to check the difference is to check the log

$ git log tells the commit message history in local repo



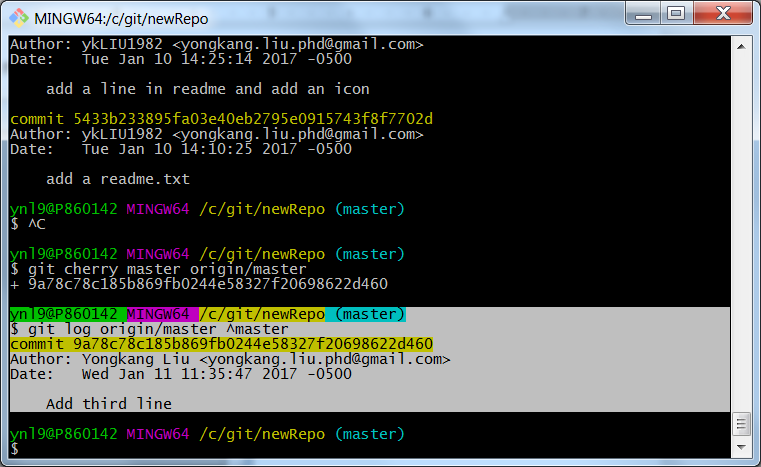
$ git fetch origin

$ git log origin/master tells the commit messages on GitHub

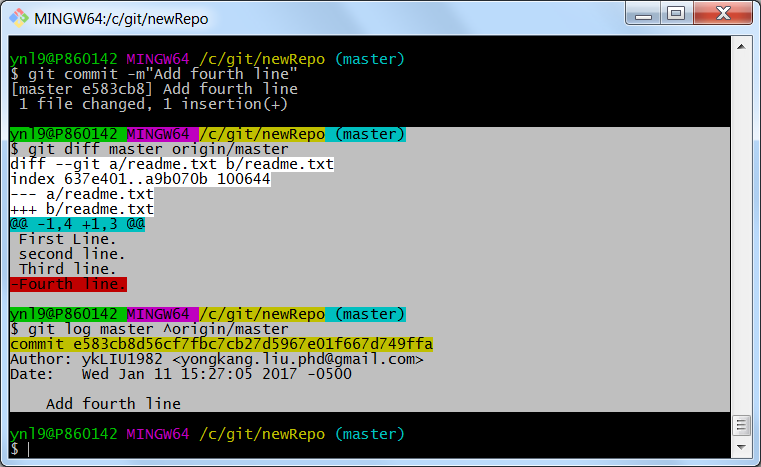


By comparing the last commit message, we can tell the difference

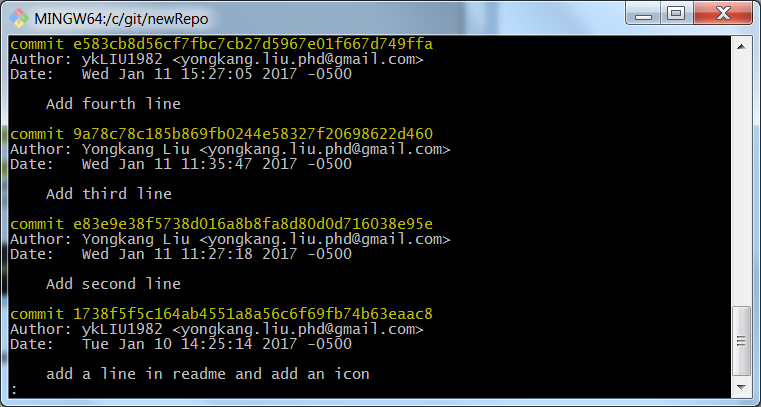
$ git log origin/master ^master tells the commits that are on GitHub but not in local



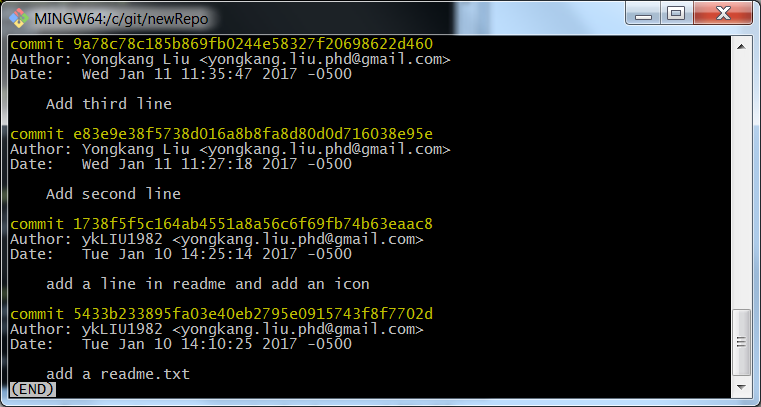
The opposite also works to see which commit has not be pushed to GitHub. E.g., we add the fourth line in readme.txt and run $ git diff master origin/master and $ git log master ^origin/master to see the difference.



If the git log is too long, the display will be scrollable like this



Press “space” key to show the next page of commit messages in the log until [END]



The online Q&A suggests that there are three ways to exist the screen: type “q”, “z”, or “Ctrl+z”. At least one would work in some case. To better control the display of the log history, it is able to add following alias in the .bashrc file

git --no-pager log --oneline -n 10

--no-pager will encounter the (END) word

-n 10 will show only the last 10 commits

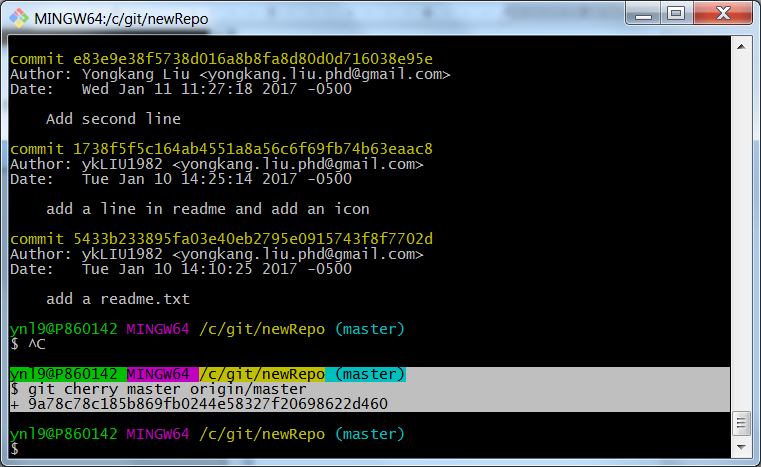
--oneline will show the commit message, ignore the author, date information

1. The third way is use $git cherry if we don’t care the real meaning of the commit message

$ git cherry master origin/master tells how many commit changes made on GitHub that are not in local. Each line means a commit on the GitHub that has not been merged into local.

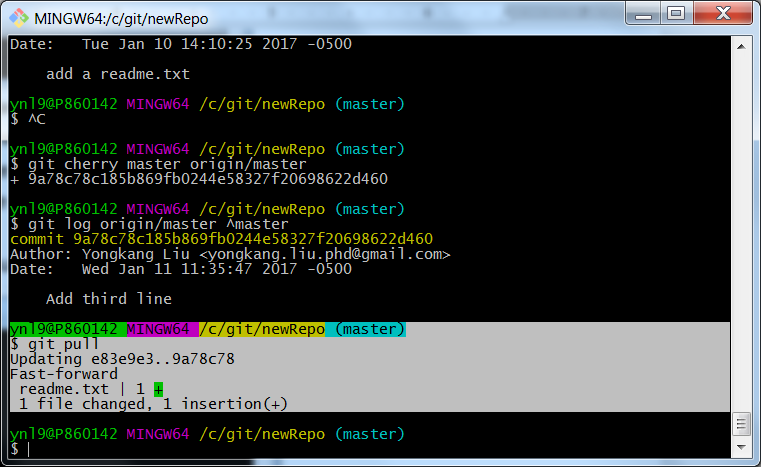
$ git cherry origin/master master tells the commits in the local that have not been pushed to GitHub.

cherry works in the similar way as diff without details in the commit



1. Once make sure that there are some commits on GitHub but not in local,

use $ git pull to get these new commits to local



Although $ git pull is equivalent to running $ git fetch and $ git merge, it is recommended online that fetch+merge is preferable than pull as fetch+merge can be used in some particular code development cases when different branches are created other than only master branch.

$ git fetch updates the so-called “remote-tracking branches” – typically these are ones that look like origin/master, github/experiment, etc. that it is seen with $ git branch -r. These are like a cache of the state of branches in the remote repository that are updated when $ git fetch is done (or a successful $ git push).

Suppose that you’ve got a remote called “origin” that refers to your GitHub repository, you would do: $ git fetch origin

And then do: $ git diff master origin/master

In order to see the difference between your “master”, and the one on GitHub. If you are happy with those differences, you can merge them with $ git merge origin/master, assuming “master” is your current branch.