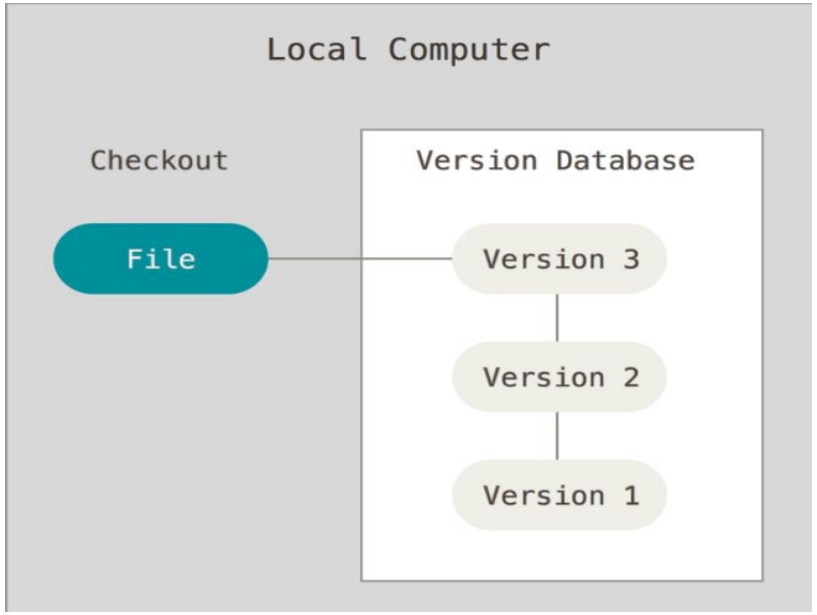
GIT, Version Control System, Branch, GIT Hub

Version Control System

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use.
- It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

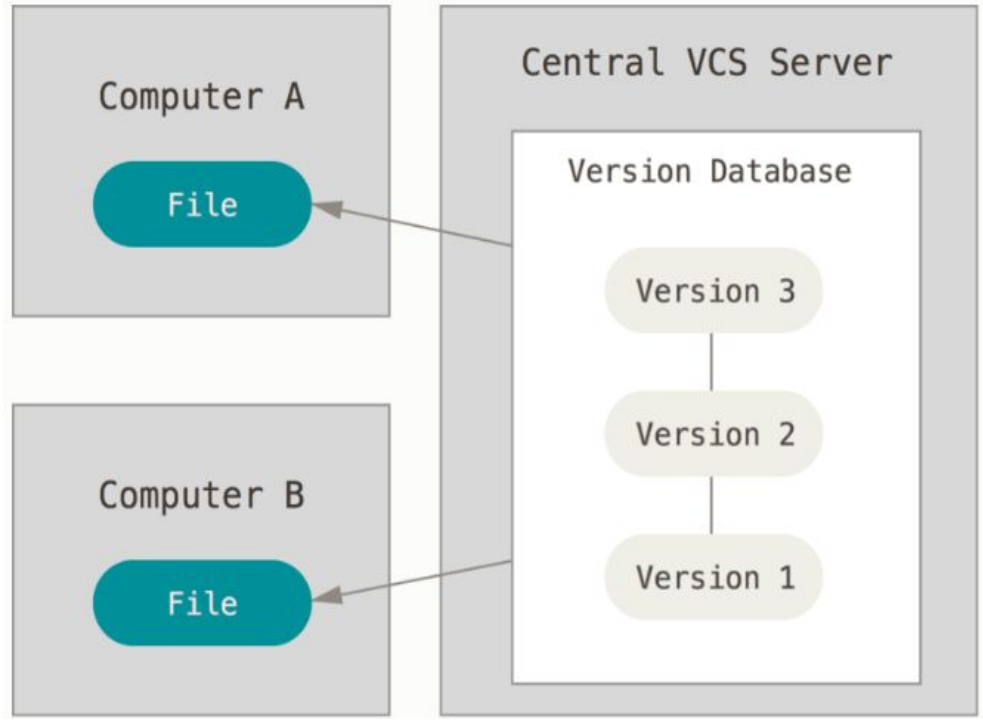
Local Version Control Systems

- Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone.



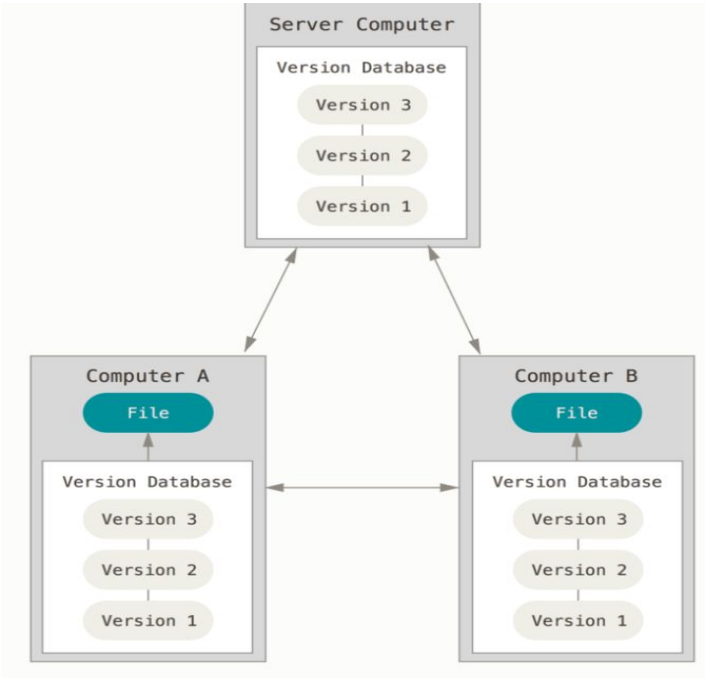
Centralized Version Control Systems

- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.



Distributed Version Control Systems

- In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.



- Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

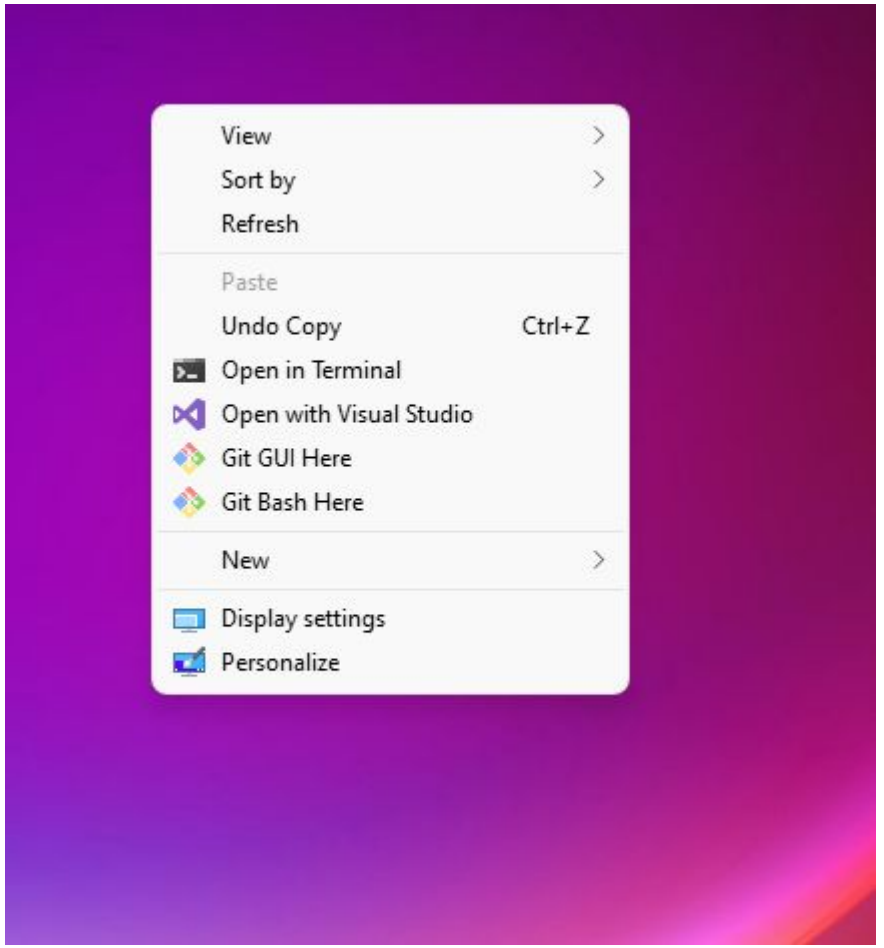
- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project

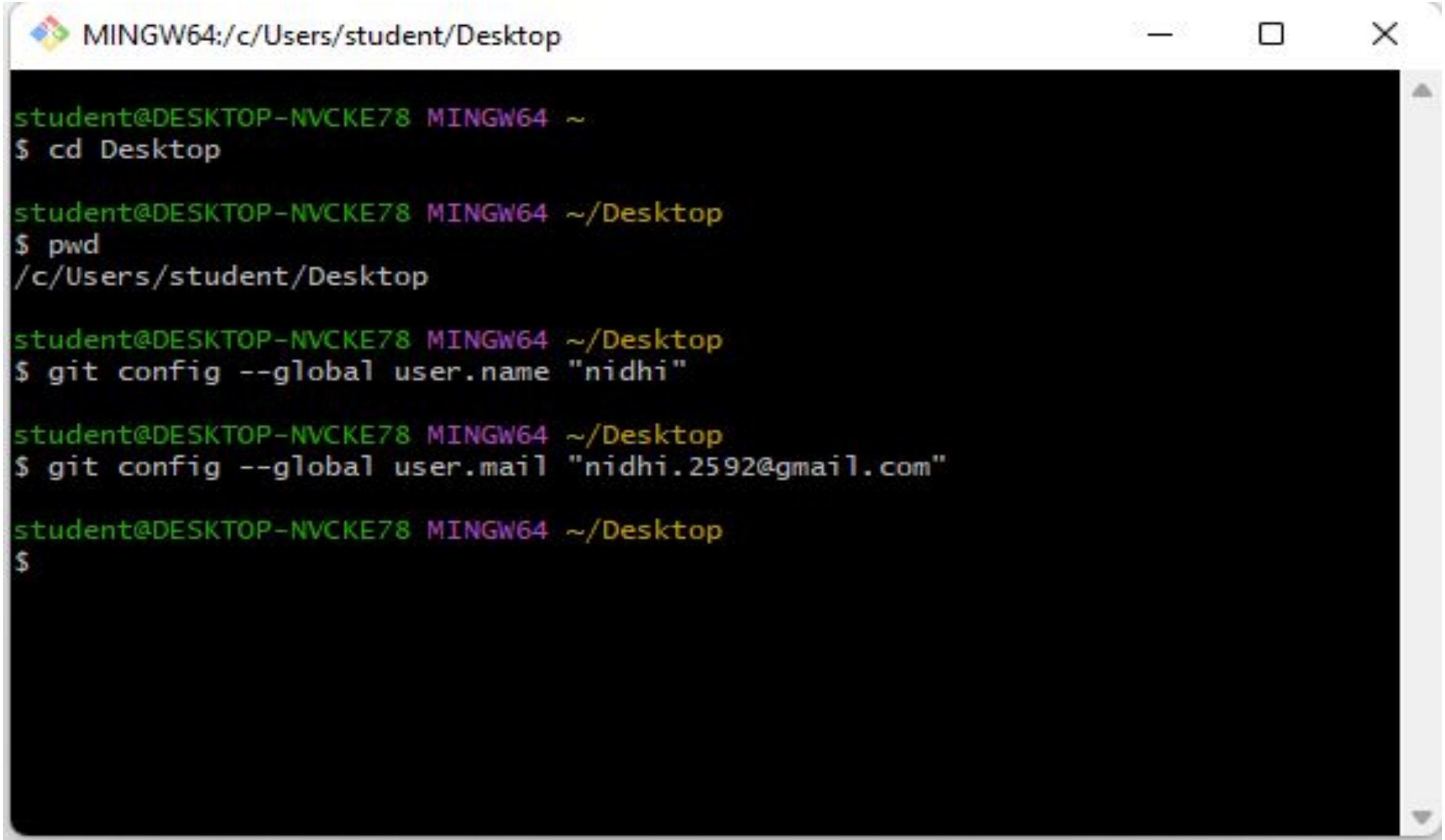
Why Git?

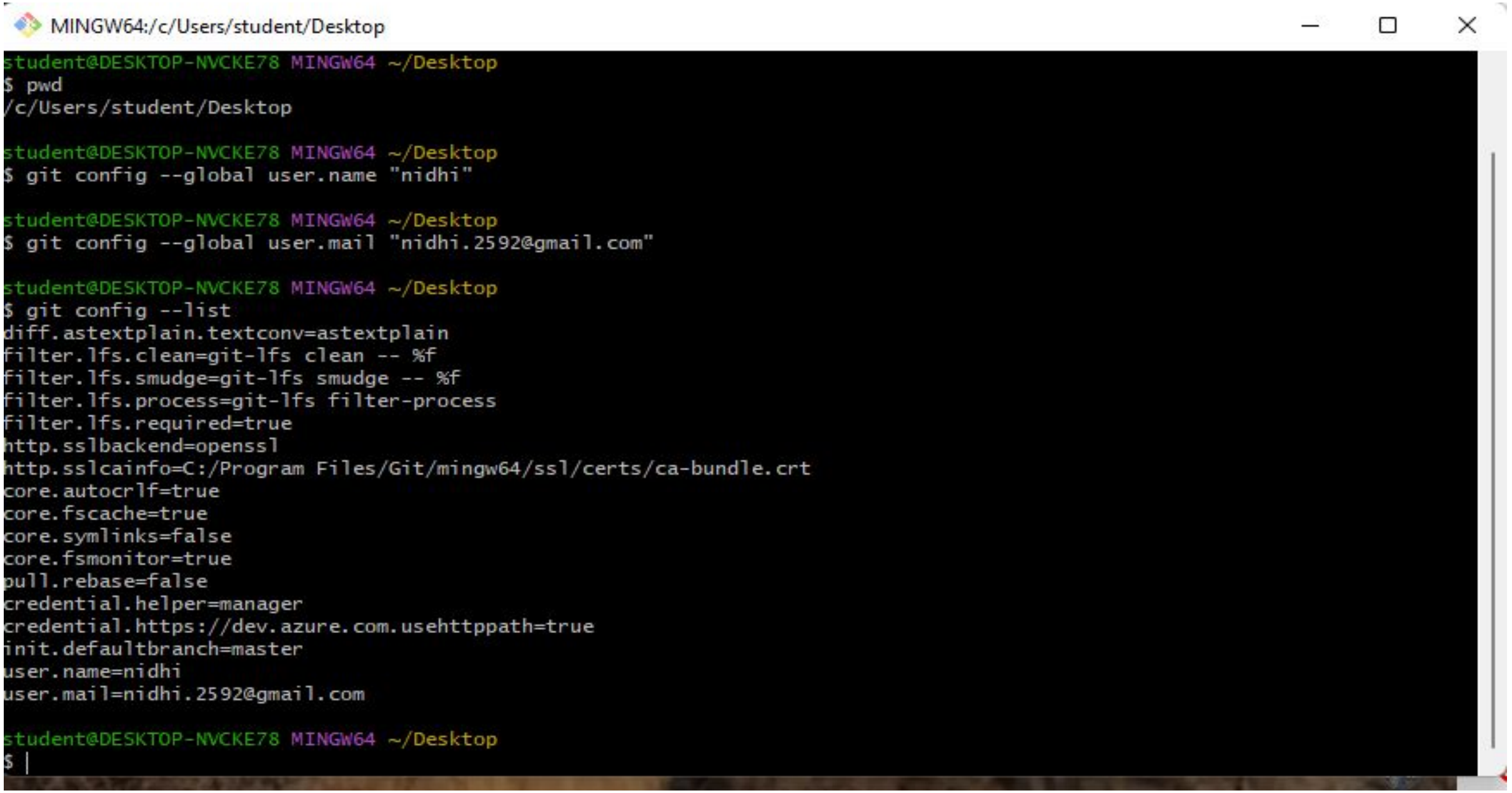
- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

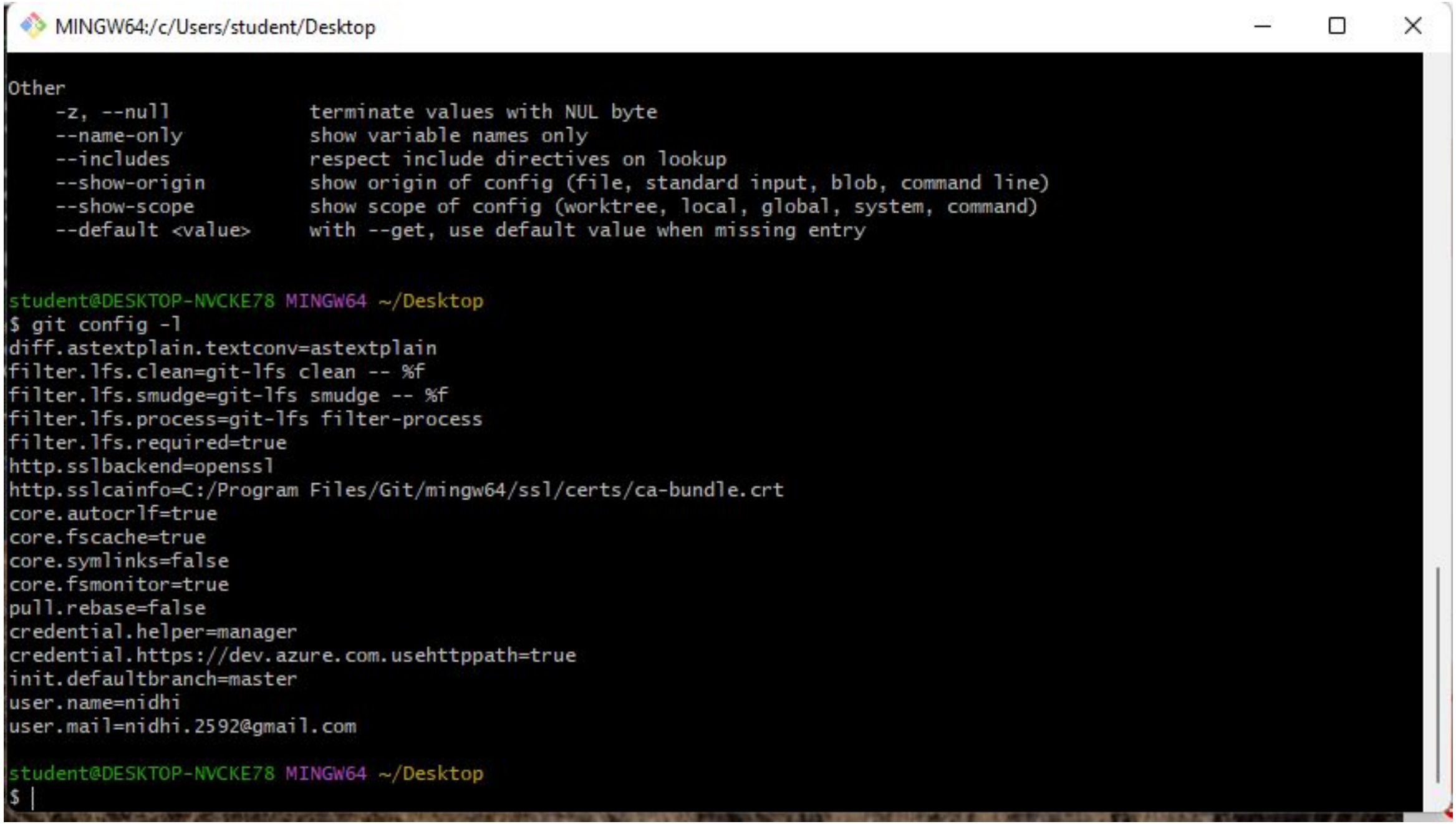
GIT Installation

- https://git-scm.com/

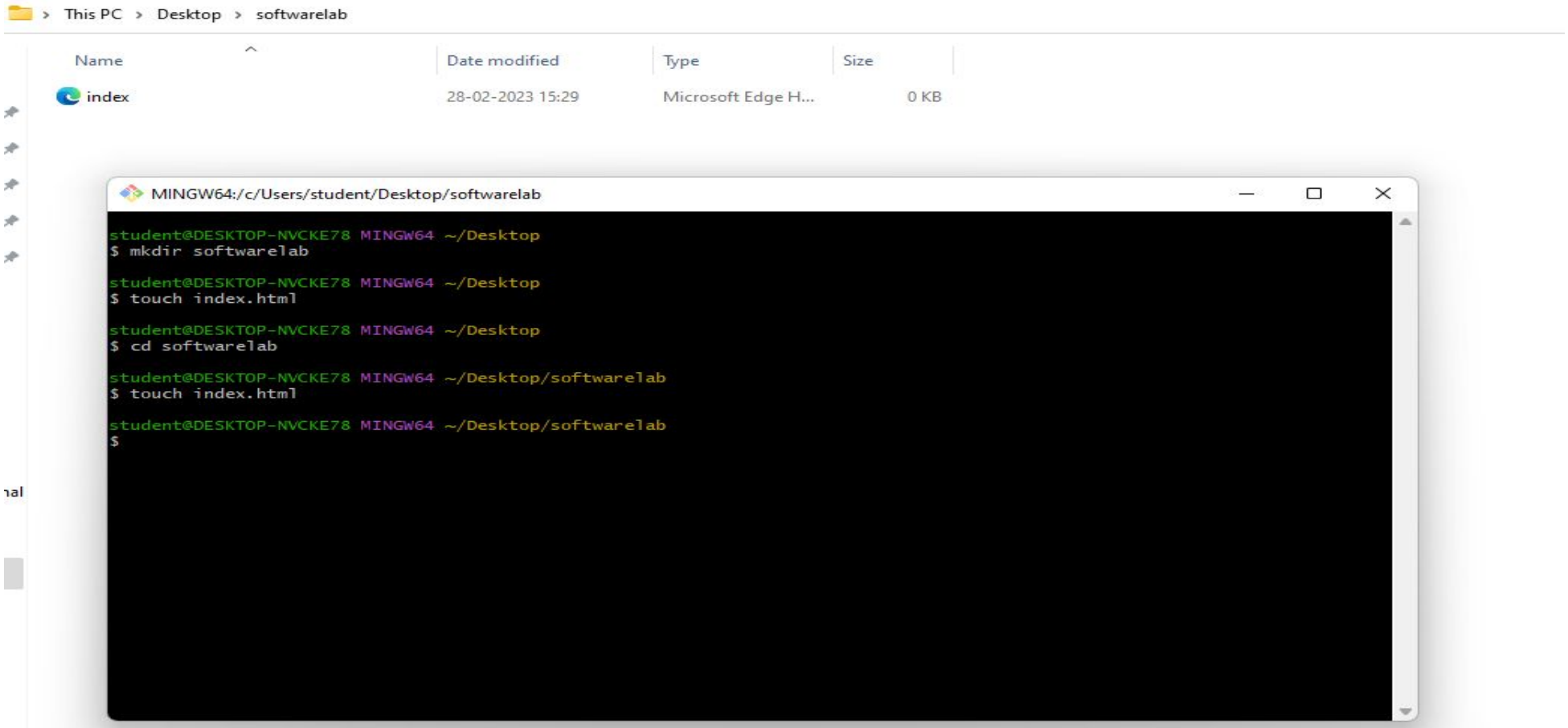


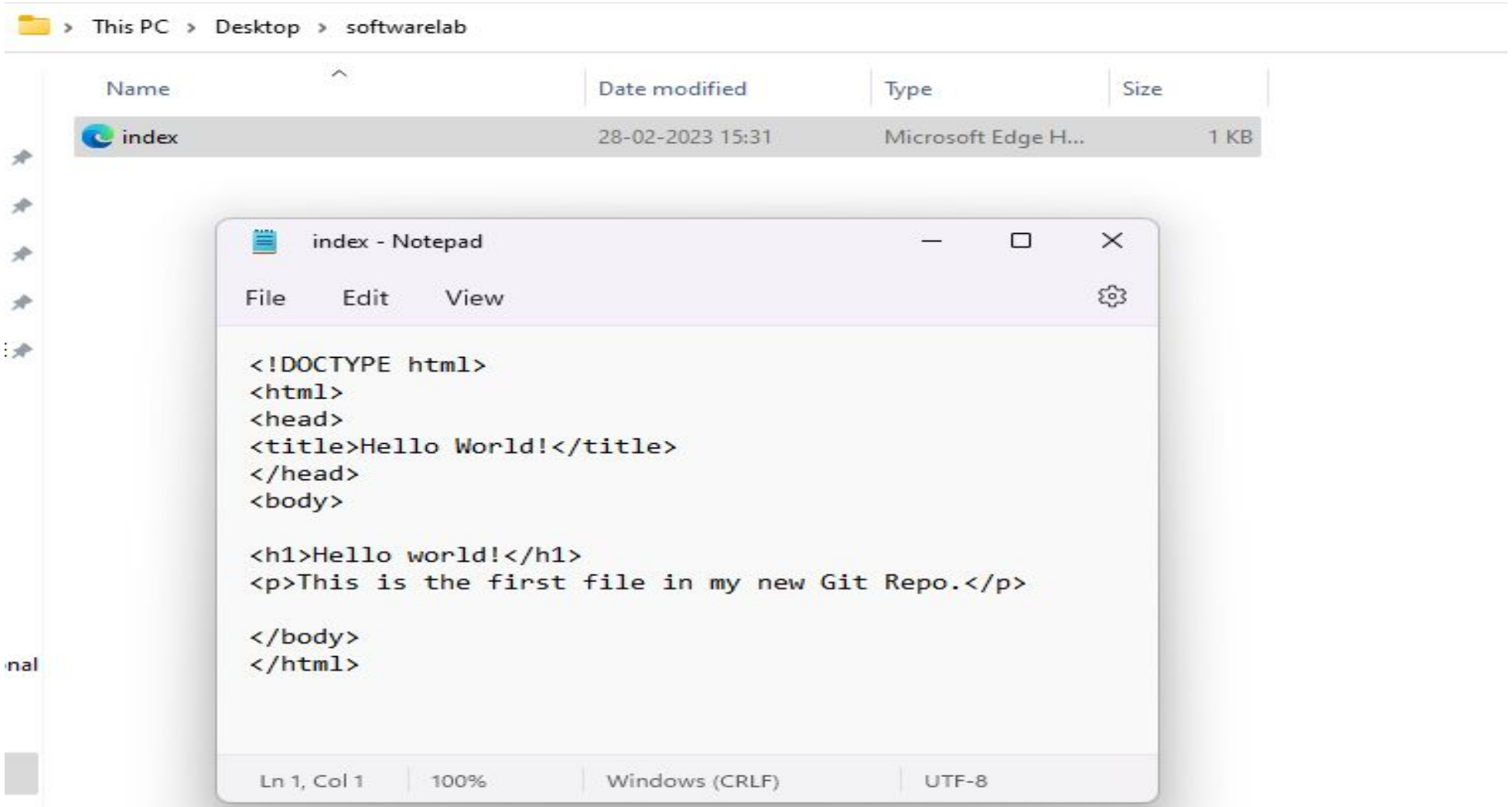

MINGW64:/c/Users/student/Desktop
student@DESKTOP-NVCKE78 MINGW64 ~
$ cd Desktop
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ pwd
/c/Users/student/Desktop
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config --global user.name "nidhi"
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config --global user.mail "nidhi.2592@gmail.com"
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$


MINGW64:/c/Users/student/Desktop
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ pwd
/c/Users/student/Desktop
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config --global user.name "nidhi"
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config --global user.mail "nidhi.2592@gmail.com"
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.fsmonitor=true
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=nidhi
user.mail=nidhi.2592@gmail.com
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$

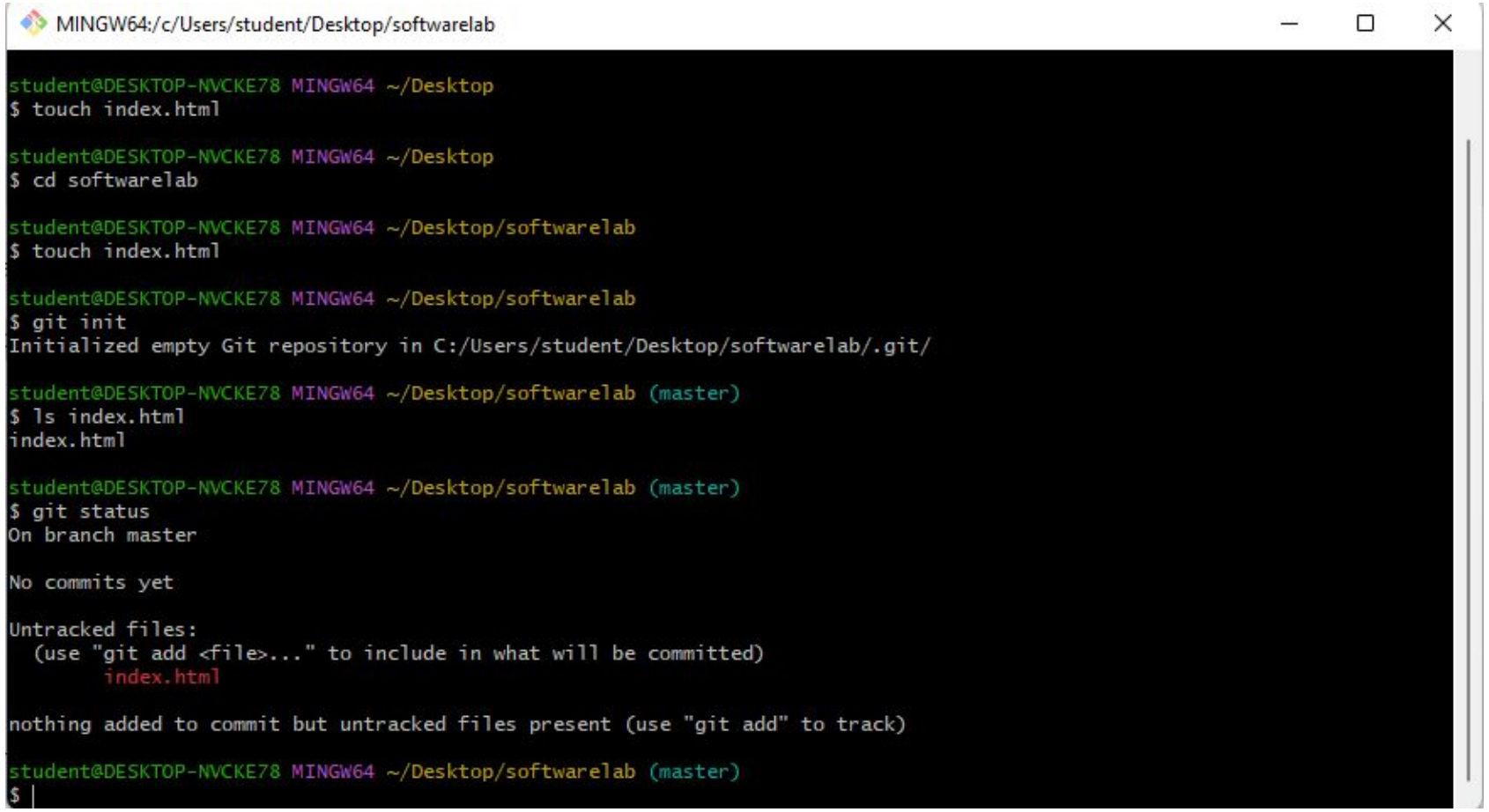

MINGW64:/c/Users/student/Desktop
Other
-z, --null terminate values with NUL byte
--name-only show variable names only
--includes respect include directives on lookup
--show-origin show origin of config (file, standard input, blob, command line)
--show-scope show scope of config (worktree, local, global, system, command)
--default <value> with --get, use default value when missing entry
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.fsmonitor=true
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=nidhi
user.mail=nidhi.2592@gmail.com
student@DESKTOP-NVCKE78 MINGW64 ~/Desktop
$

Git in Action (git - add, status, commit, checkout)




This PC
Desktop
softwarelab
Name
Date modified
Type
Size
index
28-02-2023 15:31
Microsoft Edge H...
1 KB
index - Notepad
File
Edit
View
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello world!</h1>
This is the first file in my new Git Repo.
</body>
</html>
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

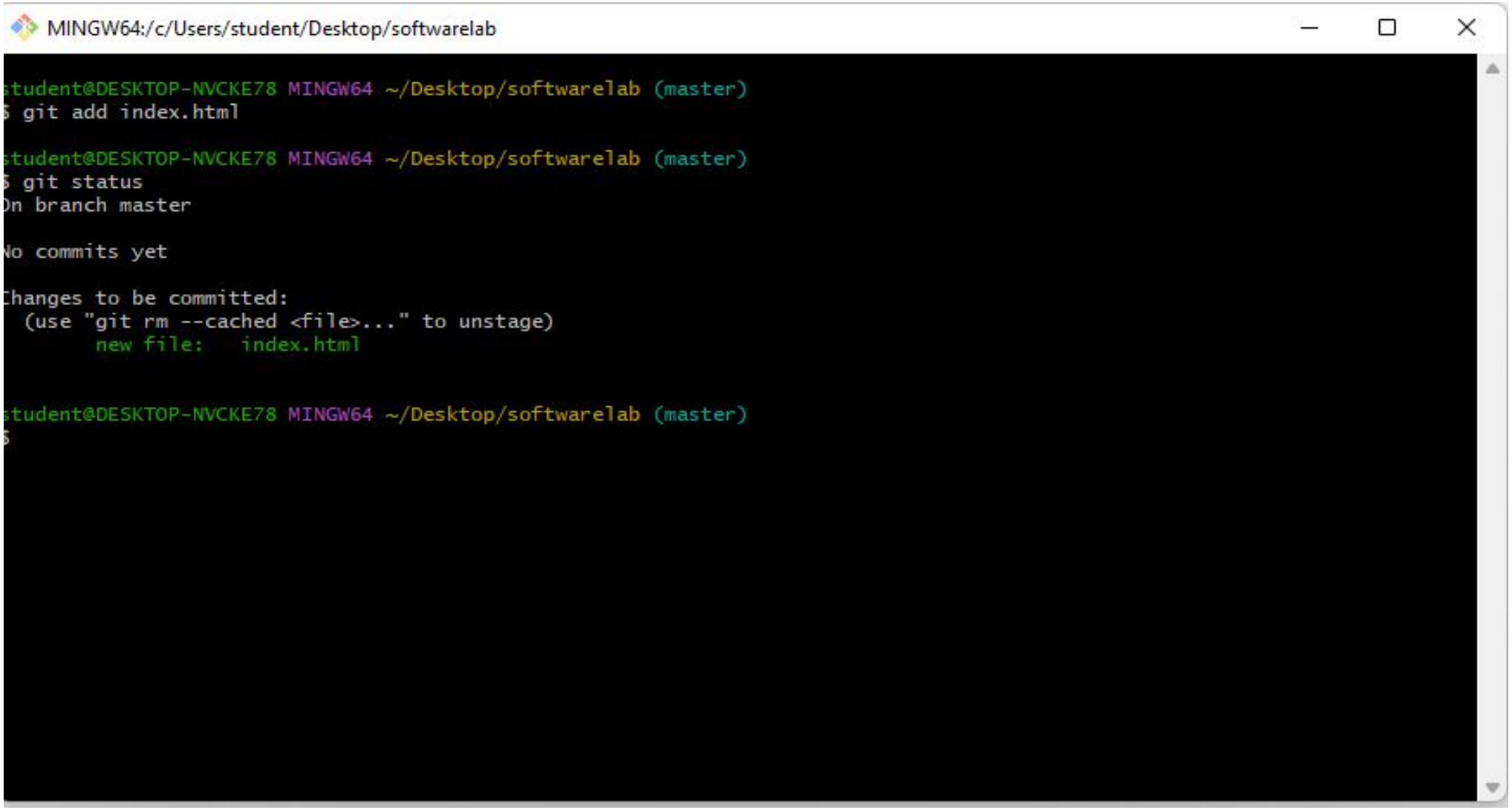
we check the Git status and see if it is a part of our repo:



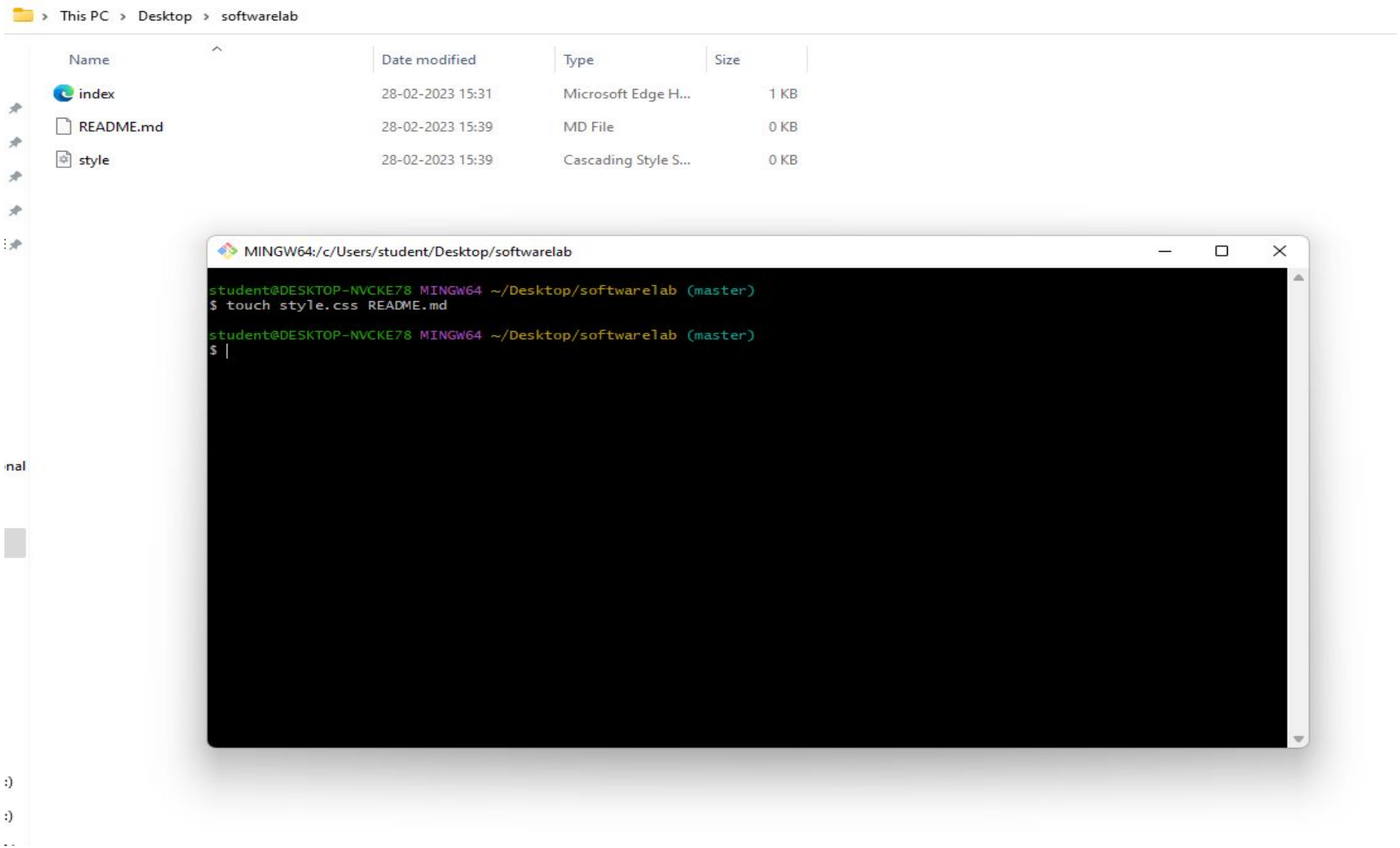
Git Staging Environment

- One of the core functions of Git is the concepts of the Staging Environment, and the Commit.
- As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.
- **Staged** files are files that are ready to be **committed** to the repository you are working on.

- `git add index.html`
- `git status`

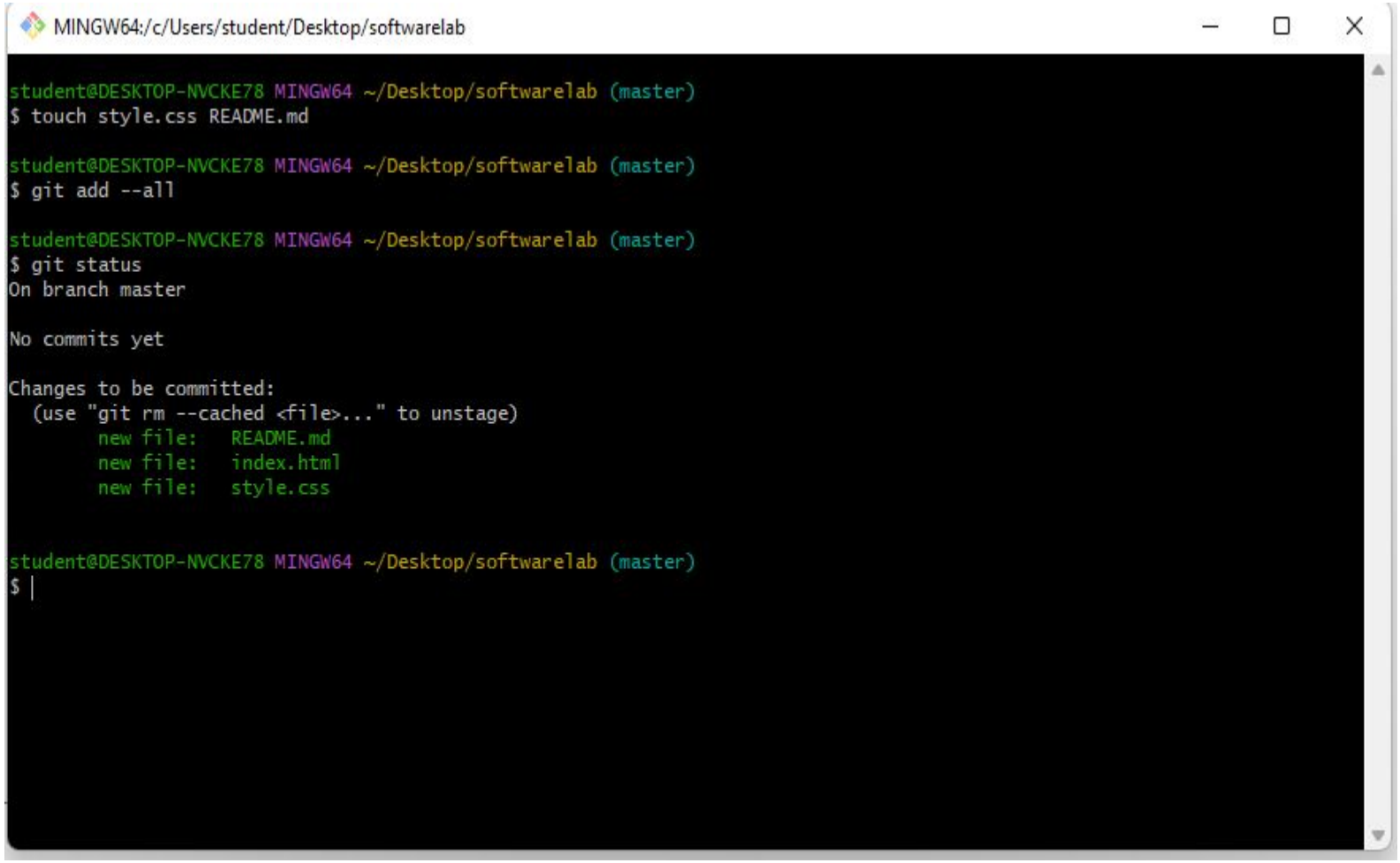


Git Add More than One File



Now add all files in the current directory to the Staging Environment:

- `git add –all`
- `git status`

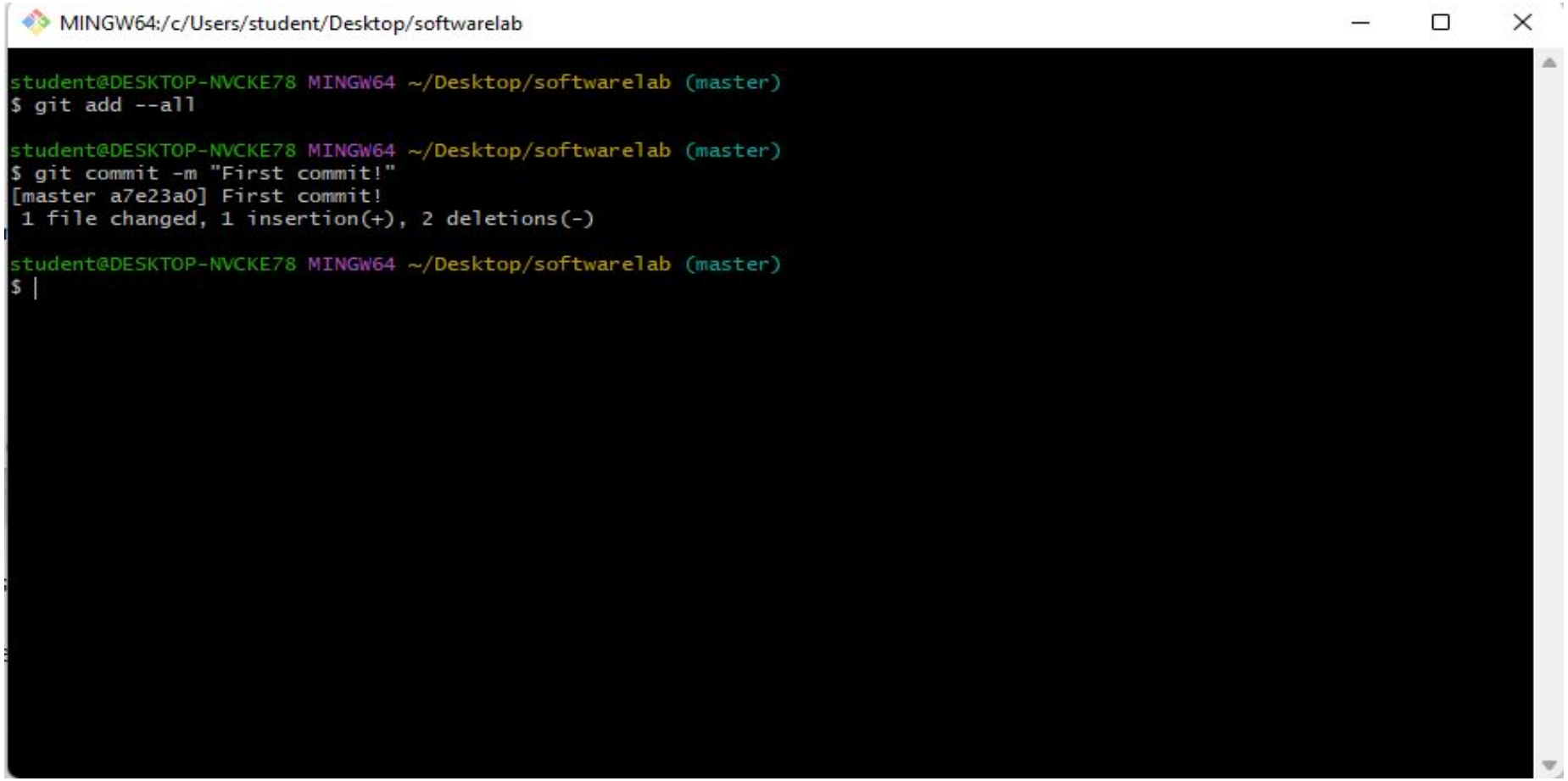


Git Commit

- Since we have finished our work, we are ready move from stage to commit for our repo.
- Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.
- When we commit, we should always include a message.
- By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Git Commit

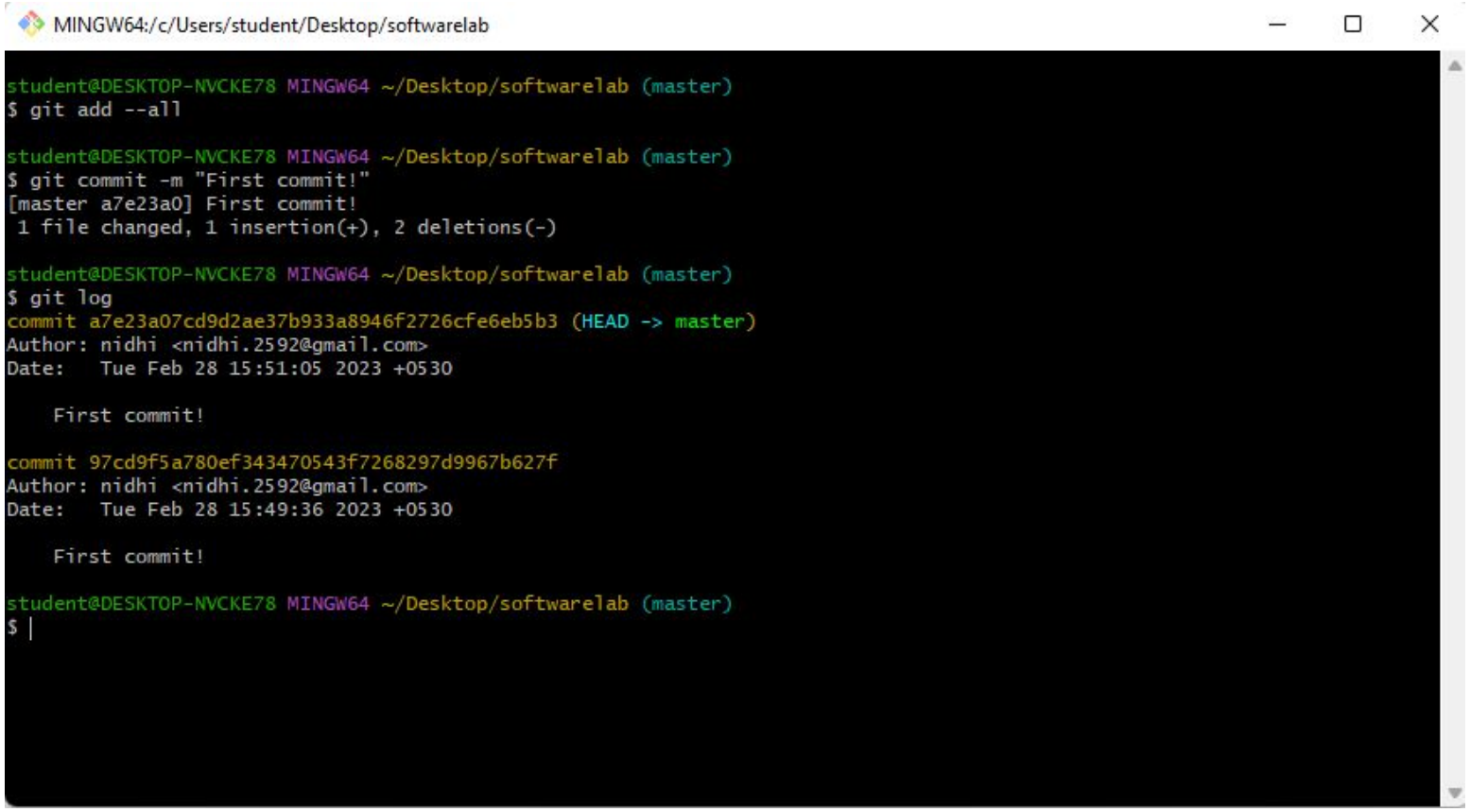
- `git commit -m "First commit!"`



Git Commit Log

- To view the history of commits for a repository, you can use the log command:

- `git log`



Git Branch

- Branches allow you to work on different parts of a project without impacting the main branch.
- When the work is complete, a branch can be merged with the main project.
- You can even switch between branches and work on different projects without them interfering with each other.
- Branching in Git is very lightweight and fast!

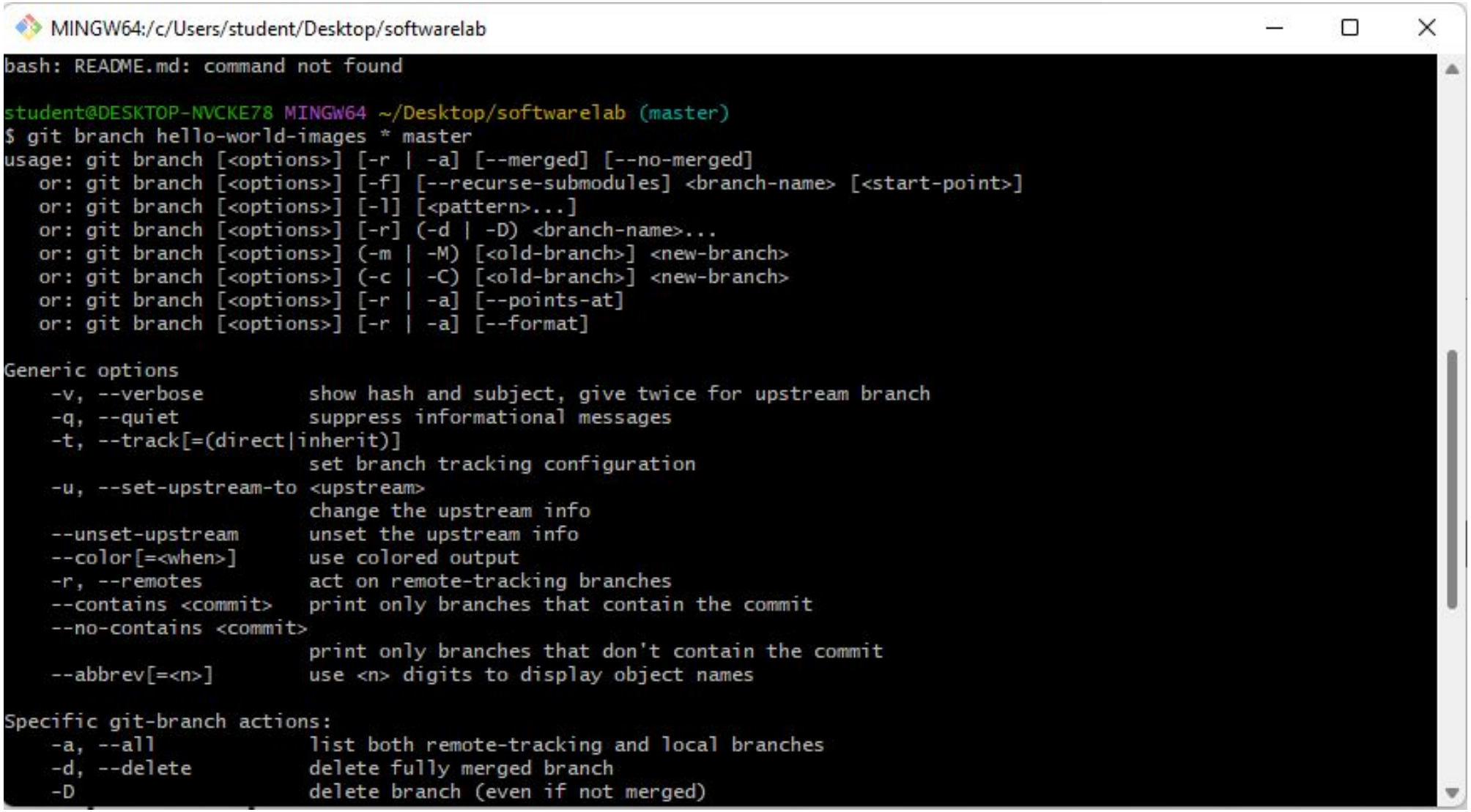
New Git Branch

- Let add some new features to our index.html page.

- We are working in our local repository, and we do not want to disturb or possibly in the main project.

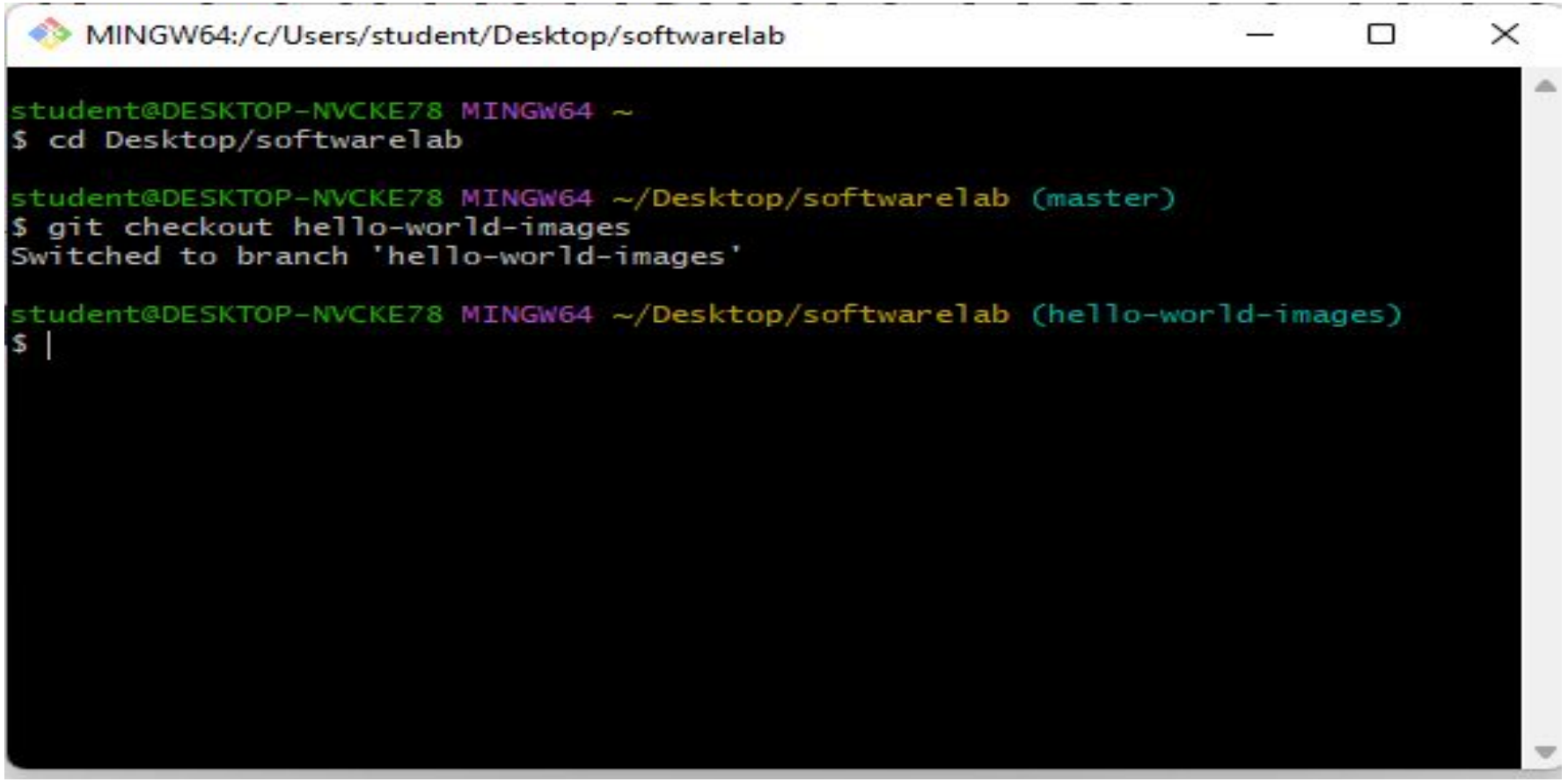
- So we create a new branch:
- `git branch hello-world-images`
- `git branch hello-world-images * master`

New Git Branch

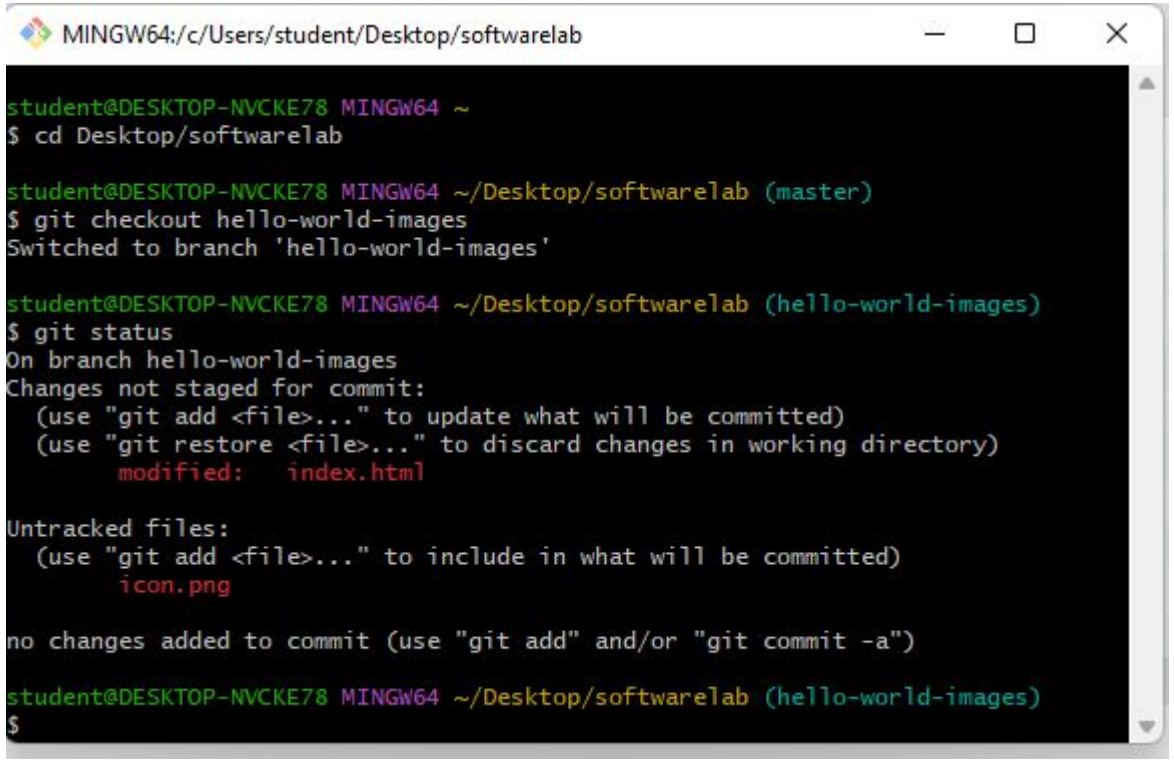


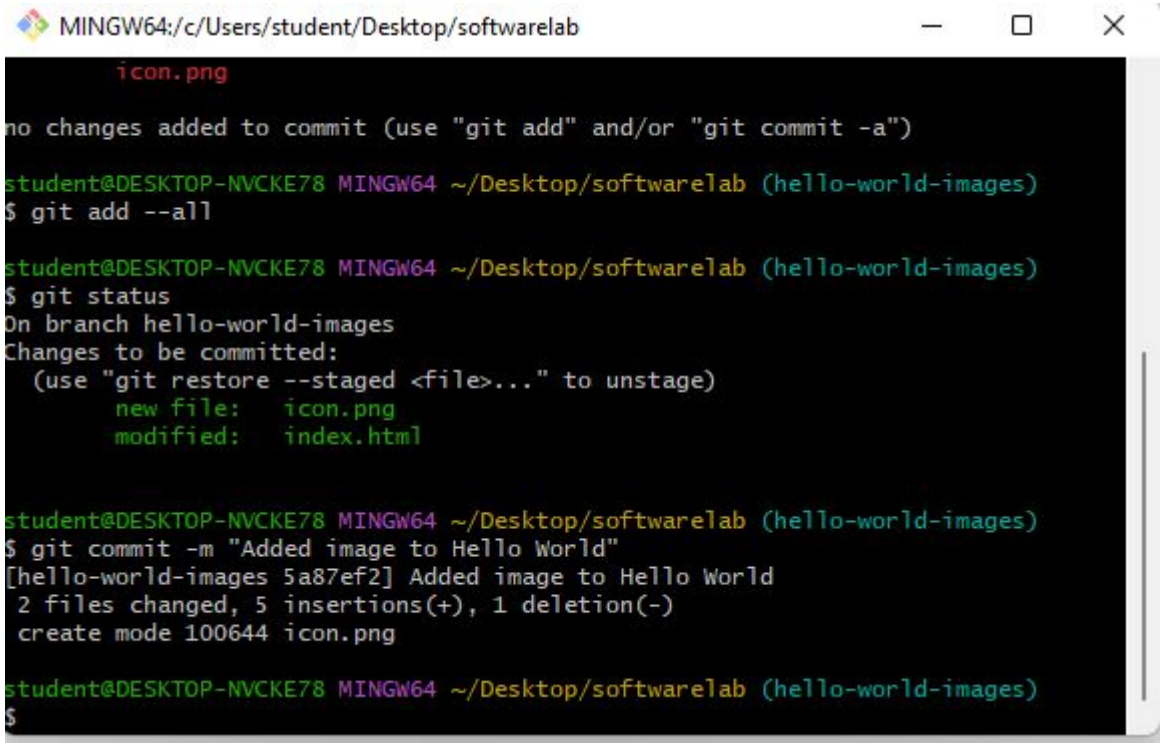
checkout is the command used to check out a branch

- `git checkout hello-world-images`

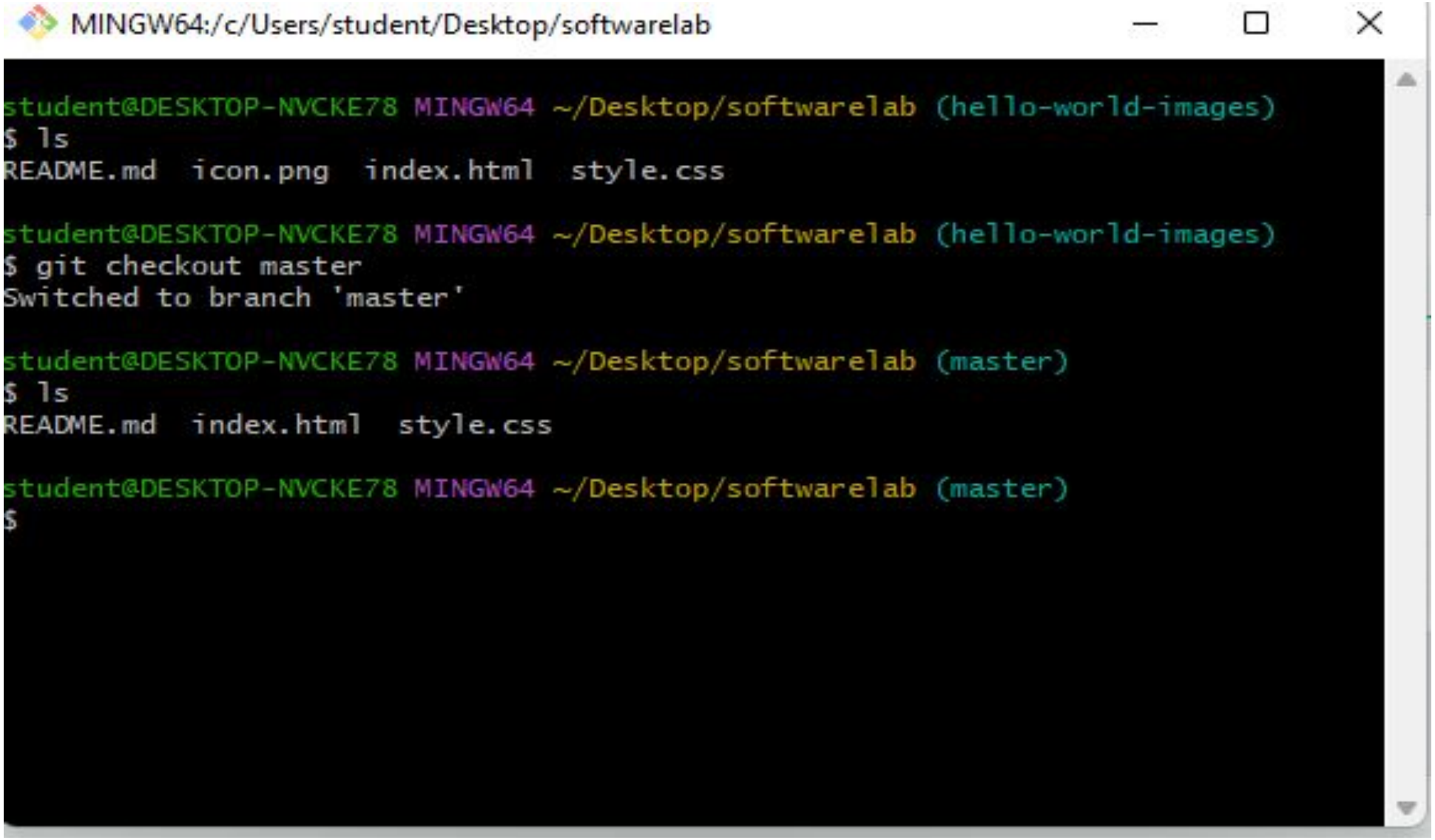


Adding file ïcon.png” is not tracked.



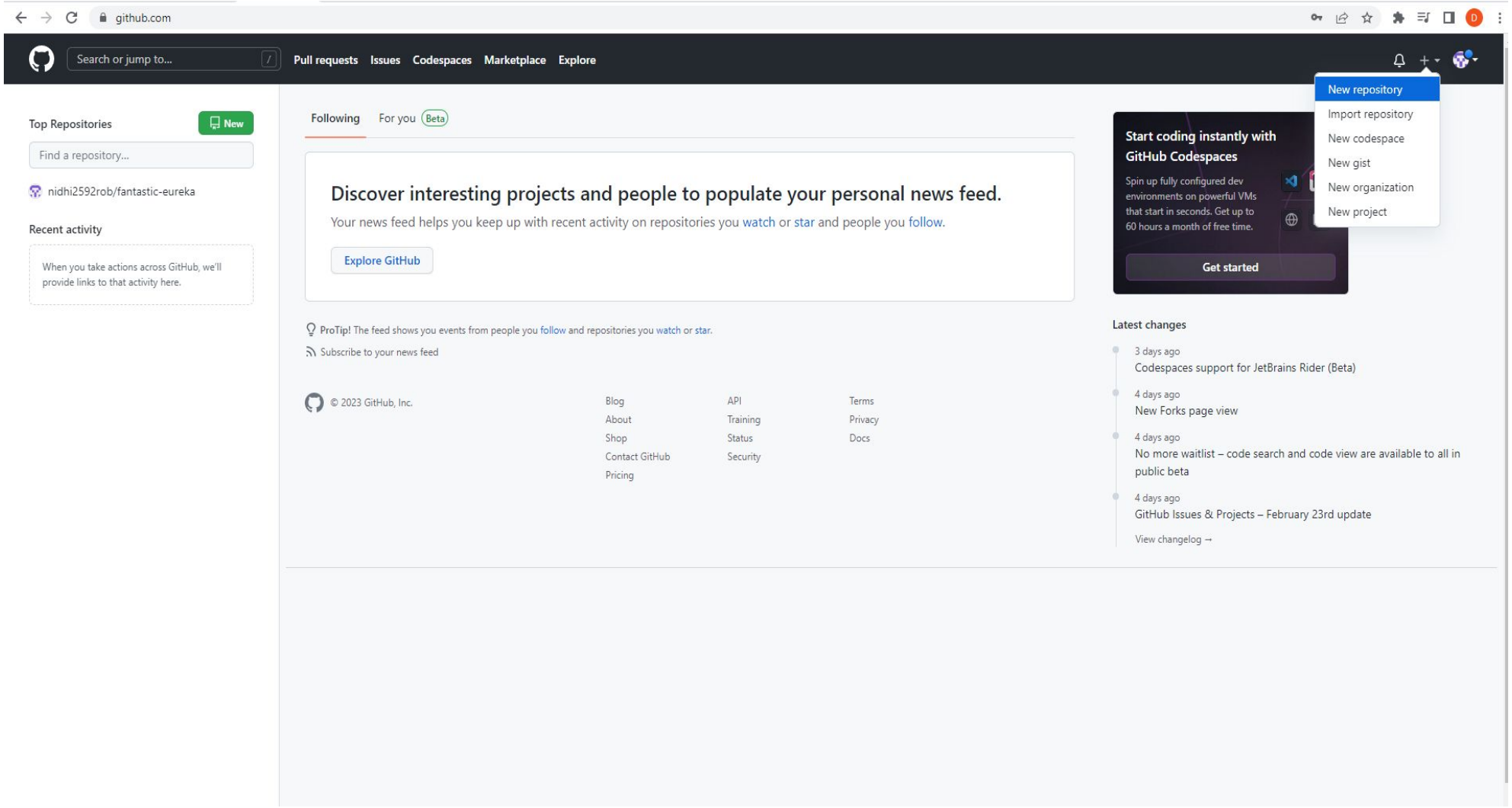


Switching Between Branches



Git and GitHub

- GitHub Account: Go to GitHub and sign up for an account:



nidhi2592rob / softwarelab Public

Pin | Unwatch 1 | Fork 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/nidhi2592rob/softwarelab.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# softwarelab" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/nidhi2592rob/softwarelab.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/nidhi2592rob/softwarelab.git
git branch -M main
git push -u origin main
```

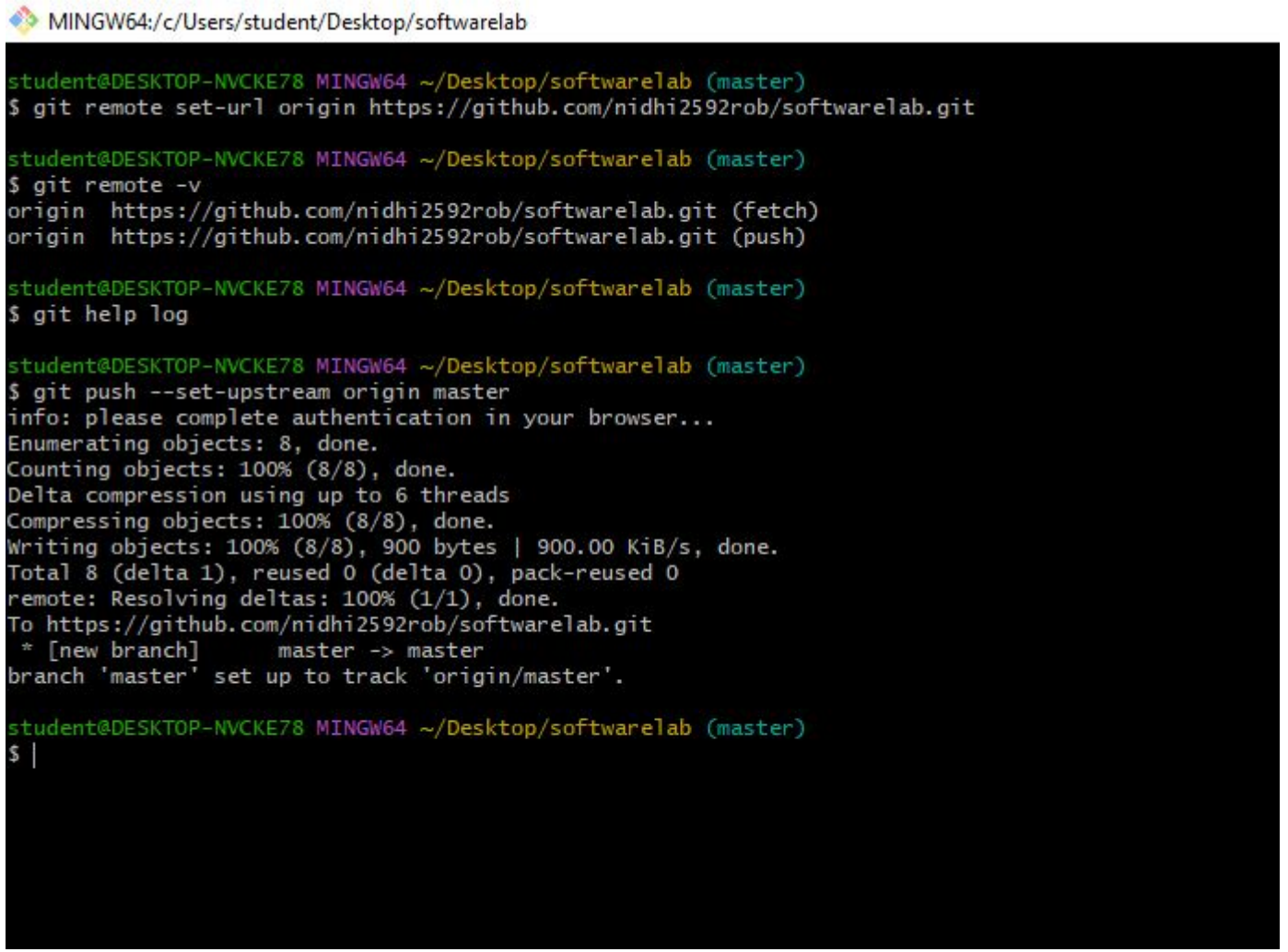
...or import code from another repository

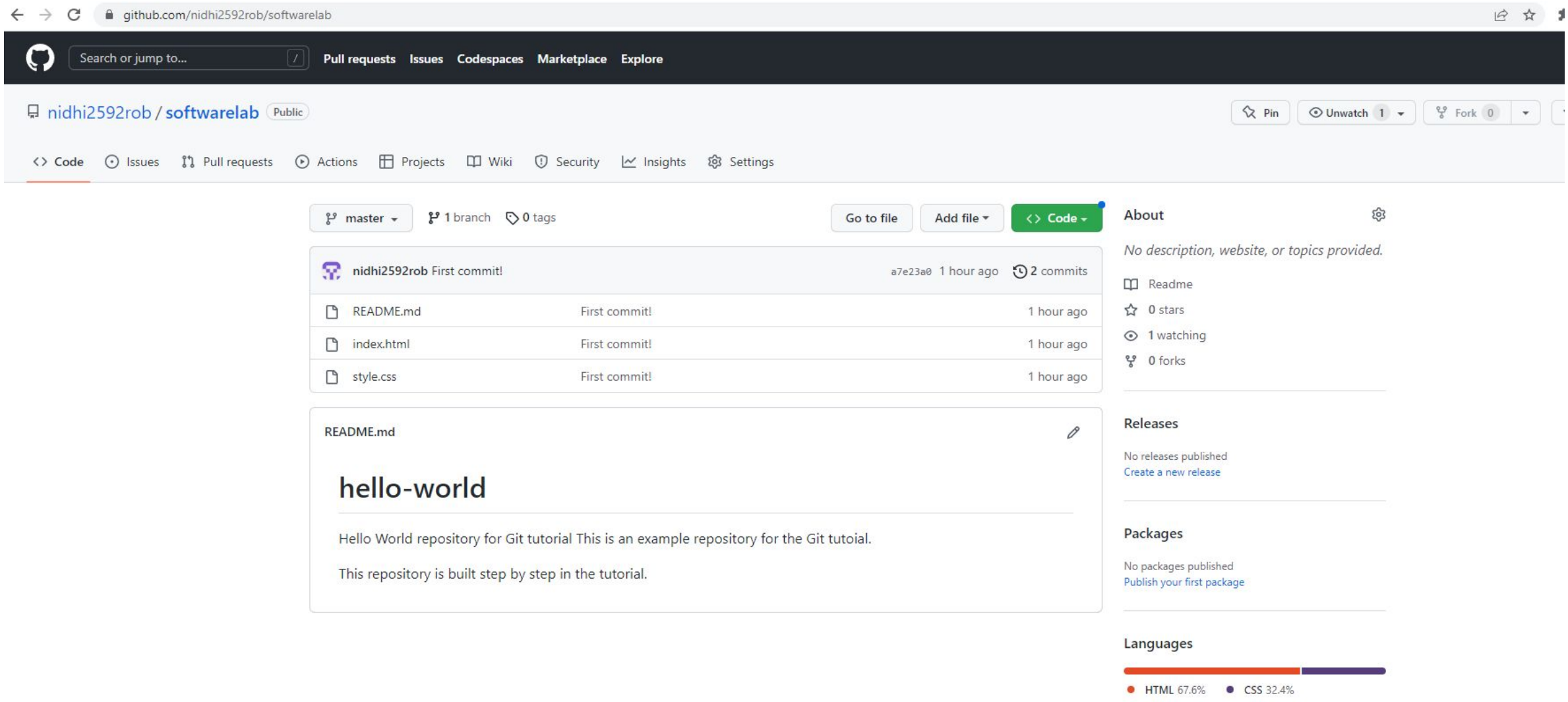
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

ProTip! Use the URL for this page when adding GitHub as a remote.

- git remote set-url origin URL specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.
- URL: https://github.com/nidhi2592rob/softwarelab.git




github.com/nidhi2592rob/softwarelab
Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
nidhi2592rob / softwarelab
Public
Pin
Unwatch 1
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
nidhi2592rob First commit!
a7e23a0 1 hour ago
2 commits
README.md First commit! 1 hour ago
index.html First commit! 1 hour ago
style.css First commit! 1 hour ago
README.md
hello-world
Hello World repository for Git tutorial This is an example repository for the Git tutoial.
This repository is built step by step in the tutorial.
About
No description, website, or topics provided.
Readme
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
HTML 67.6%
CSS 32.4%
© 2023 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About