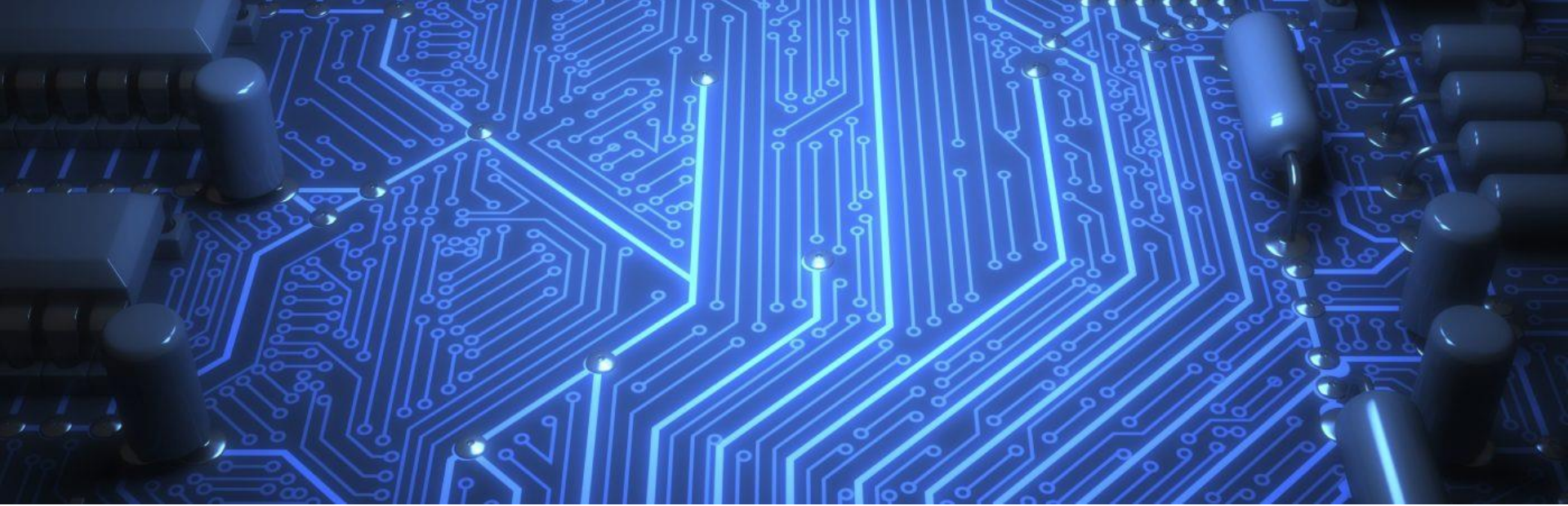


LITERACY IN COMPUTATIONAL RESEARCH

MIT MANAGEMENT SLOAN SCHOOL

OPERATIONS RESEARCH CENTER

IAP 15.S60 Session 1

Alex Schmid

Today’s Topics

Computing Tools

- Executing commands in Terminal
- Version control with Github/Git
- Distributed computing on the Engaging cluster

Writing and Organizational Tools

- Typesetting in LaTeX
- Managing citations with Zotero

Terminal

Adapted from slides by Galit Lukin, Jackie Baek

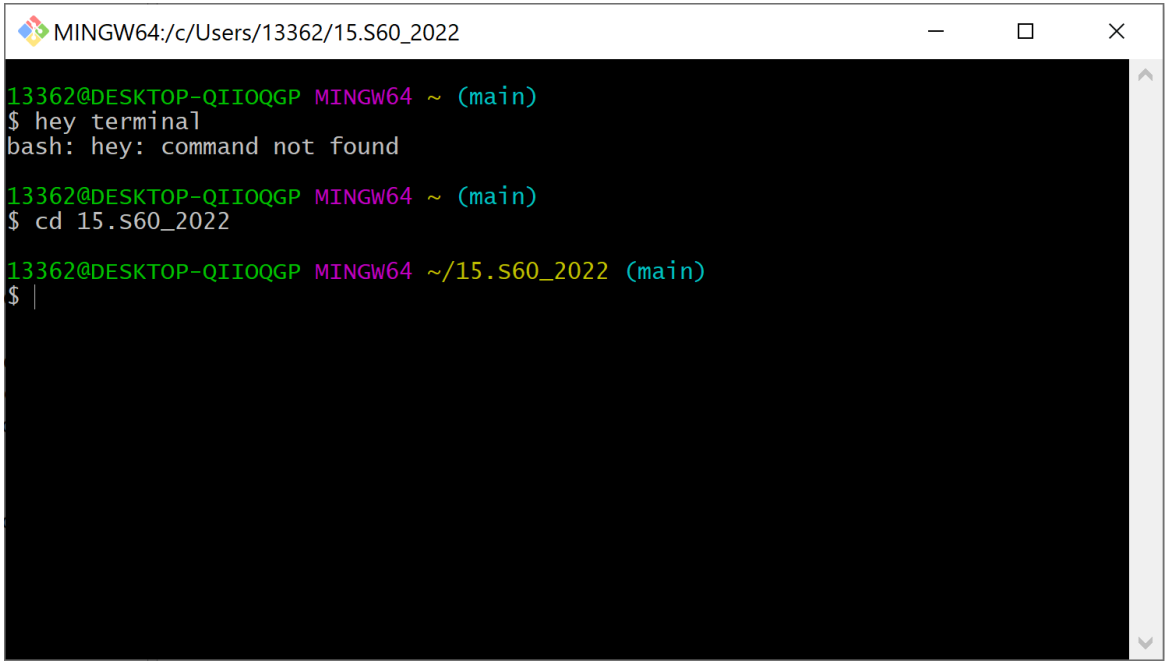
Learning Objectives - Terminal

At the end of the session, students will be able to…

- Navigate and manipulate files and directories using terminal commands
- Recognize scenarios when it is necessary or more efficient use the terminal rather than a graphical interface

What is the terminal?

The terminal/Unix shell is a text-based interface to interact with the computer.



Why use the terminal?

Repetitive tasks, like "delete all files in a directory ending in .csv"

python

Chain commands across programming languages sequentially

Athena

Engaging

Access to client servers and computing clusters with Secure Shell (SSH)

We must learn a few basic commands to interact!

File Basics

- A file is a container of data (0’s and 1’s)
- A file is contained in a directory. Files within the same directory have unique names.
- Every file and directory has a unique location in the file system, called a path.
  - Absolute path:

    /Users/Alex/Desktop/Fall2021/FallRegistration.pdf
  - Relative path (current working directory is Users/Alex/Desktop):

    Fall2021/FallRegistration.pdf
- In terminal, we are always in a **working directory**

Terminal Basics

- We are using a shell called **bash.** This program will interpret and process the commands you input into the terminal.
- A typical command looks like:

```
command <argument1> <argument2> …
```

To open:

- Mac users open Terminal
- Windows users open Git Bash (installed in the pre-assignment)

Navigating – Commands

Print working directory	`pwd`
List contents of working directory	`ls`
List contents of specified directory	`ls <directory_name>`
Change to a new directory	`cd <directory_name>`
Return to “home” directory	`cd`
Open a file (analogous to double clicking)	`open <filename>`

Navigating – Commands 2

Reference current working directory	.
Reference parent of current working directory	..
Reference home directory	~

- Navigate history of previous commands using up and down arrow keys
- Use tab to autocomplete commands and paths

Poll Question

If our current working directory is `Users/Alex/Stuff`, which command would take us back to the home directory, `Users/Alex`?

A.) `cd ..`

B.) `cd ../..`

C.) `cd ~`

D.) `cd .`

E.) Both A and C

Poll Question

If our current working directory is `Users/Alex/Stuff`, which command would take us back to the home directory, `Users/Alex`?

A.) `cd ..`

B.) `cd ../..`

C.) `cd ~`

D.) `cd .`

E.) Both A and C

File Manipulation – Commands

Create a new directory	`mkdir <directory_name>`
Create a new file	`touch <filename>`
Delete a file **(cannot be undone!)**	`rm <filename>`
Edit file contents (Nano is a text editor)	`nano <filename>`
Print contents of a file	`cat <filename>`
Move or rename a file	`mv <source_filename> <target_filename>`

File Manipulation – Try it out

We want to create a new directory called `my_directory`, navigate to it, add a new file called `myfile.txt`, then add a line of text to the file.

1. Create and navigate to a new directory

```
mkdir my_directory
cd my_directory
```

2. Add and open a new file for editing

```
touch myfile.txt
nano myfile.txt
```

3. Add some text, save (Ctrl+O), and exit Nano (Ctrl+X)

Redirecting Outputs

Run a script

Send the output log to a file

(rather than the terminal)

```
python processStuff.py > outputfile.txt
```

Redirecting Outputs

python

Run a script

Send the output log to a file

(rather than the terminal)

Overwrite / create file	`<command> > outputfile.txt`
Append to file	`<command> >> outputfile.txt`

Redirecting Outputs - Try it out

Create another text file in `my_directory` called `newfile.txt`. Print the contents of `my_directory` and direct the list to a file called `outputfile.txt`.

1. Create new text file

```
touch newfile.txt
```

2. Print the contents of the directory and redirect the output

```
ls > outputfile.txt
```

Check the output:

```
cat outputfile.txt
```

More Terminal

There are many more things you can do in Terminal.

Simple pattern matching to find and sort files

`readdata.py` `makeplots.R`

Use multiple programming languages to run sequential processes

Shell scripts to automate chains of commands you use often

If you're interested, check out the tutorial here: https://swcarpentry.github.io/shell-novice/

Git & Github

Adapted from slides by Galit Lukin and Jackie Baek
with activites from Turing Way and Software Carpentry

Learning Objectives – Git/Github

At the end of the session, students will be able to contribute to a project with shared code, maintain a version control history with appropriate documentation, and easily share code with the broader community.

Git and Github

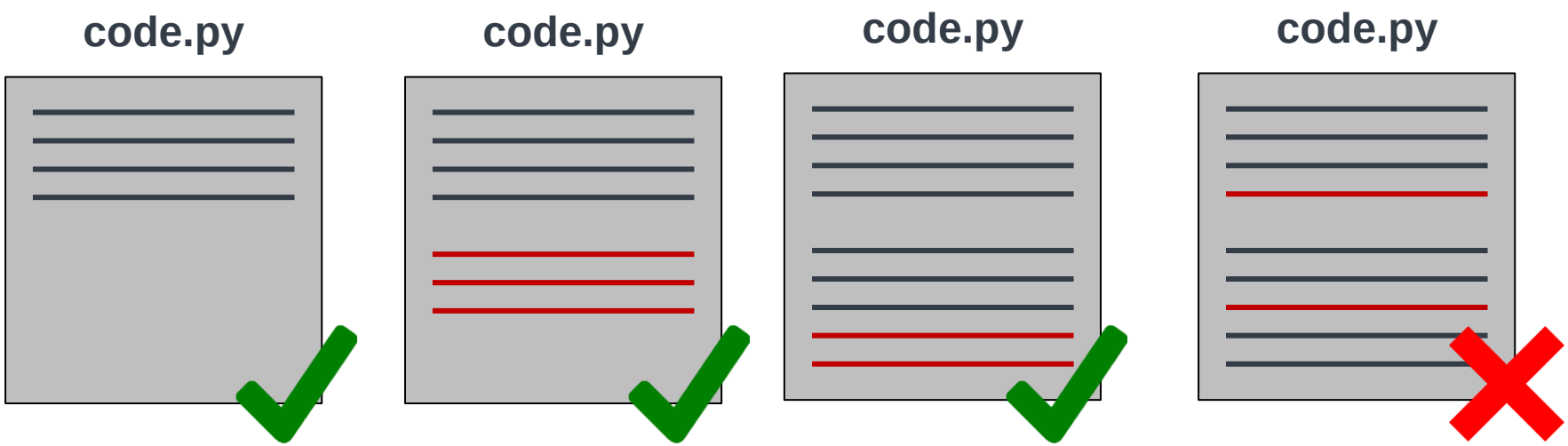
Github - Hosting platform for version control and collaboration

- Widely used by developers to store their projects
- Easily share code and data, privately and publicly

Git - The version control system itself

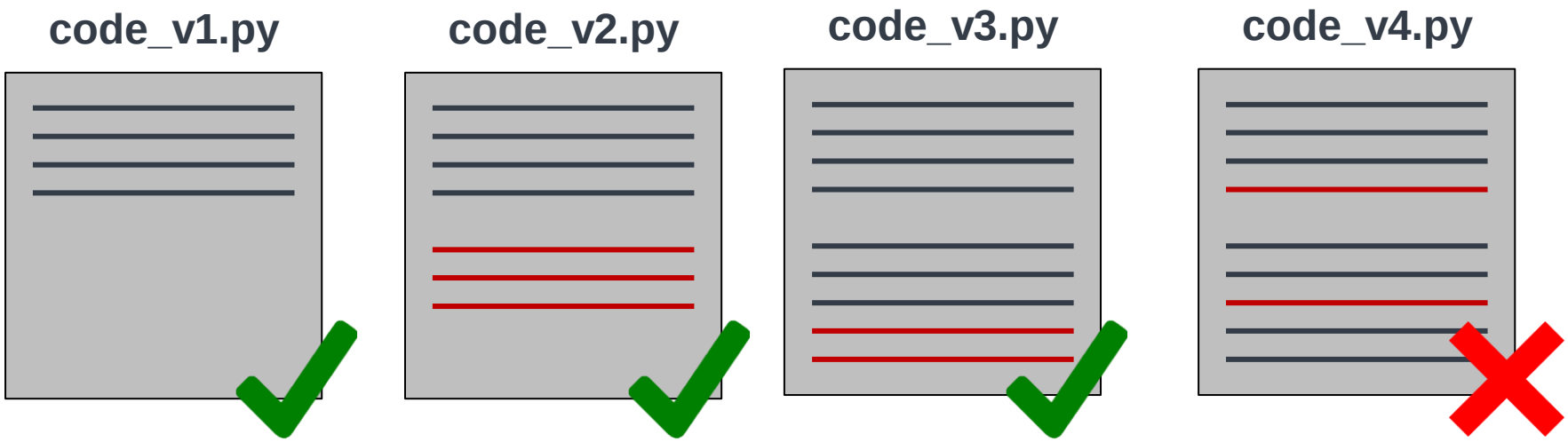
- Command line tool
- Used to make local code changes and communicate those changes with Github

Why use Git?



If only I could go back to a version that was working…

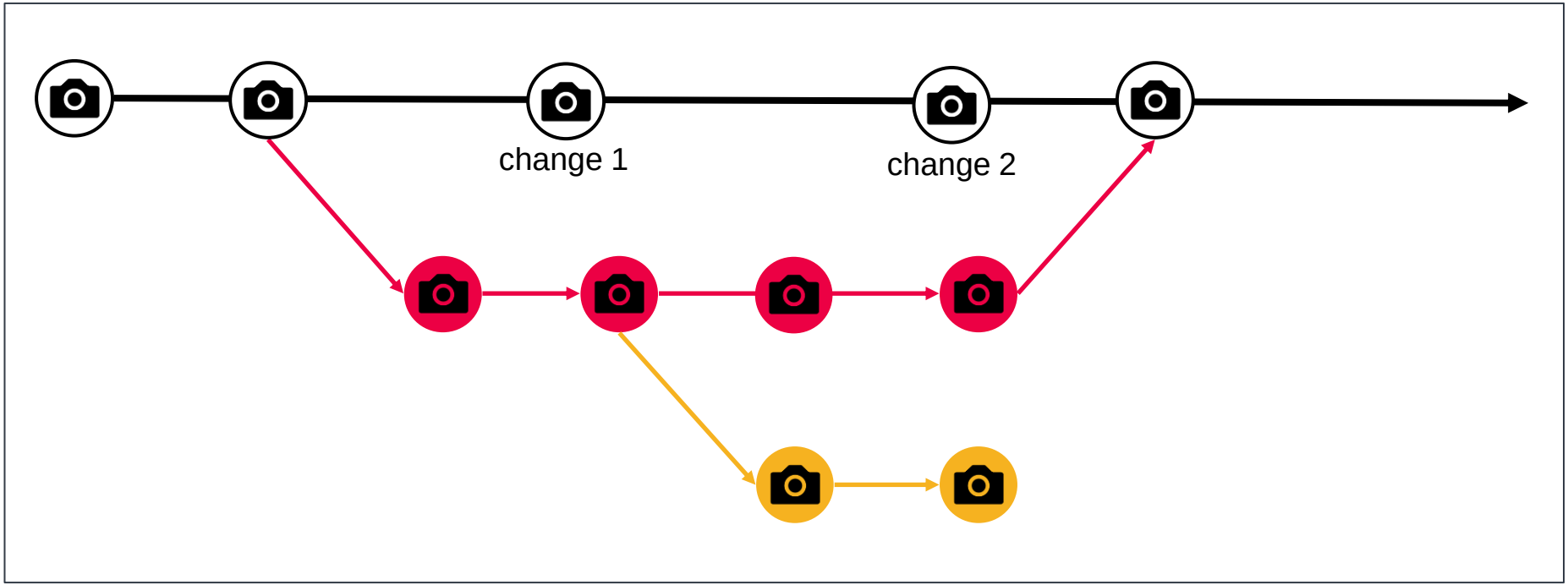
Why use Git?



Which version to go back to? How often do I save a new version? How do I name the versions meaningfully? What changed between versions?

High-Level Idea

repository (aka repo)



Create a repository - Commands

1. Create a new directory for your repo

```
mkdir <directory_name>
```

---

2. Navigate to the directory

```
cd <directory_name>
```

---

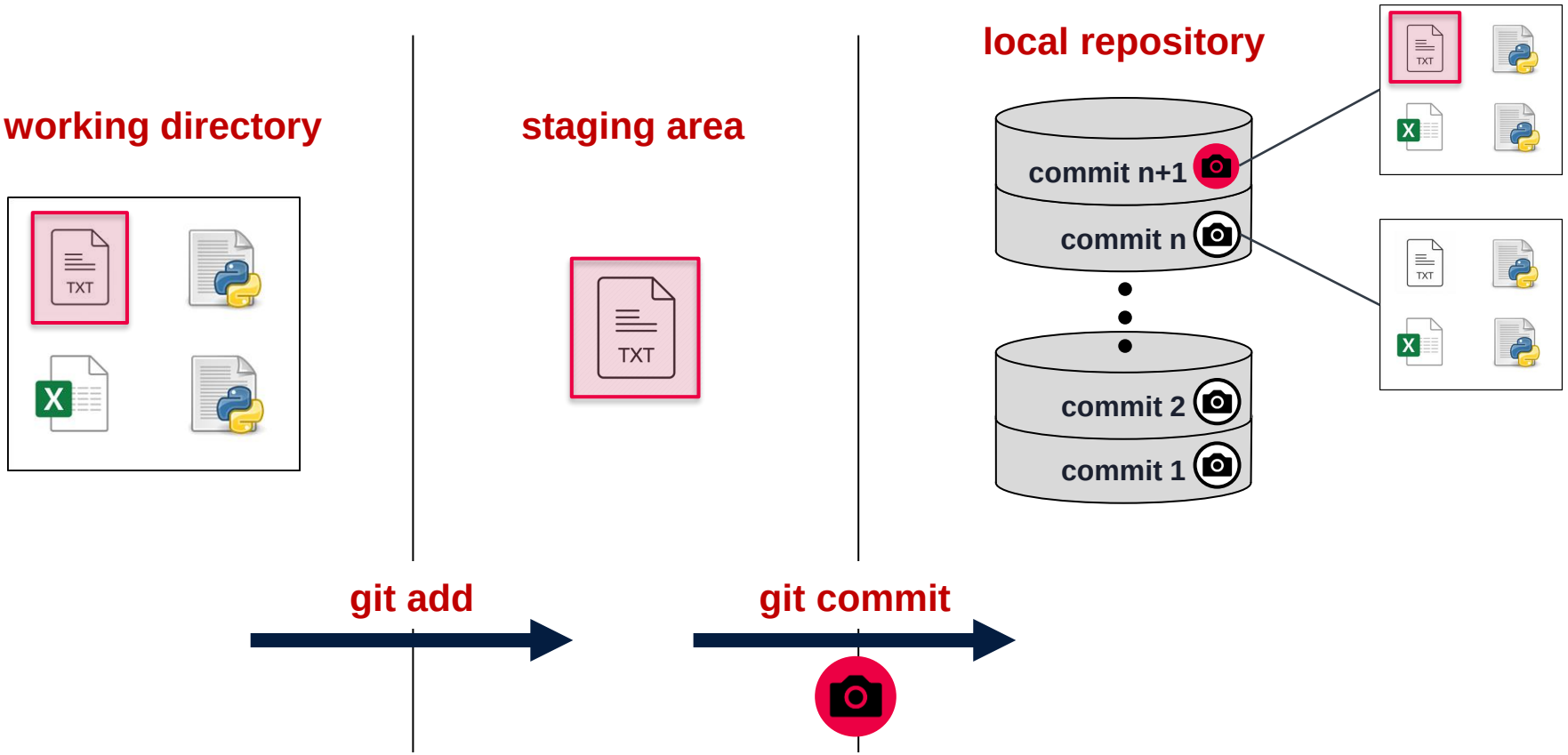
3. Initialize repository (you only need to do this once for each repo you create)

```
git init
```

Tell Github who you are

```
git config user.email "<github_email>"
git config user.name "Name"
```

Add and commit



Add and commit - Commands

1. Make changes in working directory

---

2. Check the files in the staging area and any “untracked” changes (not yet added)

```
git status
```

---

3. As you change files:
working directory → staging area

```
git add <filename>
```

```
git add .
```
(adds all files)

---

4. Once you have added a new functionality/completed an update:
staging area → local repo

```
git commit -m “comment”
```

Add a short, intuitive comment describing the change in functionality
→ For reference later!

Add and commit - Try it out

We want to create a new repository called “myrepo”, create a text file called “myfile.txt”, and add and commit it to the repository.

1. Create and navigate to a new directory

```
mkdir myrepo
cd myrepo
```

2. Clone the Github repository to your machine

```
git init
```

3. Create an empty text file in myrepo

```
touch myfile.txt
```

4. Push the text file to your new repository

```
git add myfile.txt
git commit -m “Adding first file”
```

Add and commit - Try it out

Add some text in the myfile.txt, then add and commit the changes to the repo.

1. Open the file to edit

```
nano myfile.txt
```

2. Add some text and save

3. Add and commit the changes to the repository

```
git add myfile.txt
git commit -m “Add text to myfile”
```

Poll Question

In your opinion, which would be the most helpful commit message one month from now?

A.) `“fixed bug”`

B.) `“added functions: cleanData, processData, runPredictions”`

C.) `“added function to plot results”`

D.) `“set n=100, t_max=500, s=10”`

Poll Question

In your opinion, which would be the most helpful commit message one month from now?

A.) `“fixed bug”`

B.) `“added functions: cleanData, processData, runPredictions”`

C.) `“added function to plot results”`

D.) `“set n=100, t_max=500, s=10”`

Poll Question

What does coolwords.txt look like after I run this?

```
$ touch coolwords.txt
$ nano coolwords.txt (Add line of text: “Line one, having fun”)
$ git add coolwords.txt
$ nano coolwords.txt (Add line of text: “Line two, whoop-de-doo”)
$ git commit -m “Commit message”
```

A.)

```
Line one, having fun
```

B.)

```
Line one, having fun
Line two, whoop-de-do
```

Poll Question

What does `coolwords.txt` look like after I run this?

```
$ touch coolwords.txt
$ nano coolwords.txt (Add line of text: “Line one, having fun”)
$ git add coolwords.txt
$ nano coolwords.txt (Add line of text: “Line two, whoop-de-doo”)
$ git commit -m “Commit message”
```

A.)

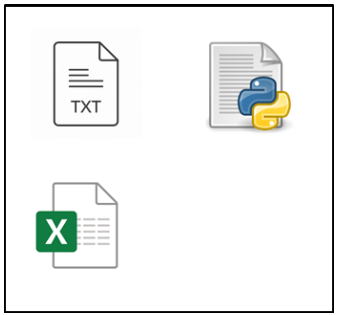
```
Line one, having fun
```

B.)

```
Line one, having fun
Line two, whoop-de-do
```

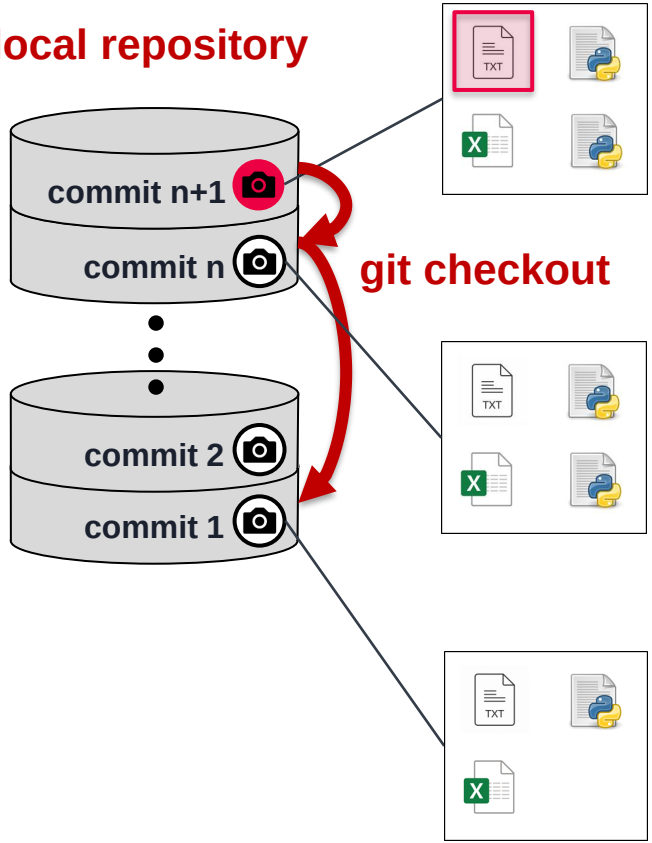
Reverting Changes

working directory



staging area

local repository



Reverting Changes - Commands

See log of commits, including SHA ID for each version

```
git log
```

Return to a previous version of a file from the commit history

```
git checkout <version_SHA> <filename>
```

See differences between two commits or differences in a specific file from two commits

```
git diff <version1_SHA> <version2_SHA>
```

```
git diff <version1_SHA:filename>
<version2_SHA:filename>
```

Reverting Changes - Try it out

Check the log of commits, look at the differences between our two commits so far, and then check out the first commit.

1. View the log to see our commit history

```
git log
```

2. Find the differences between the two commits (The first 7 digits of the SHA id should be sufficient)

```
git diff 1093106 976686d
```

or

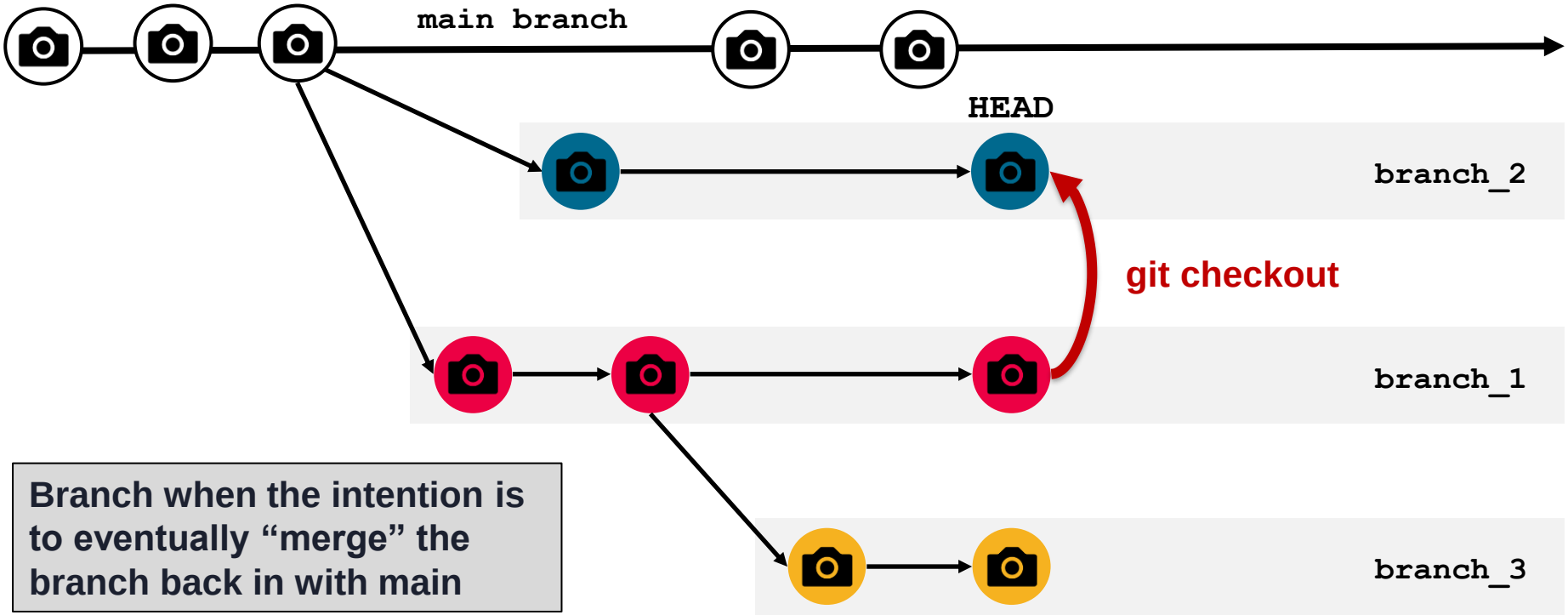
```
git diff 976686d
```

3. Check out the previous commit

```
git checkout 976686d myfile.txt
```

Branching

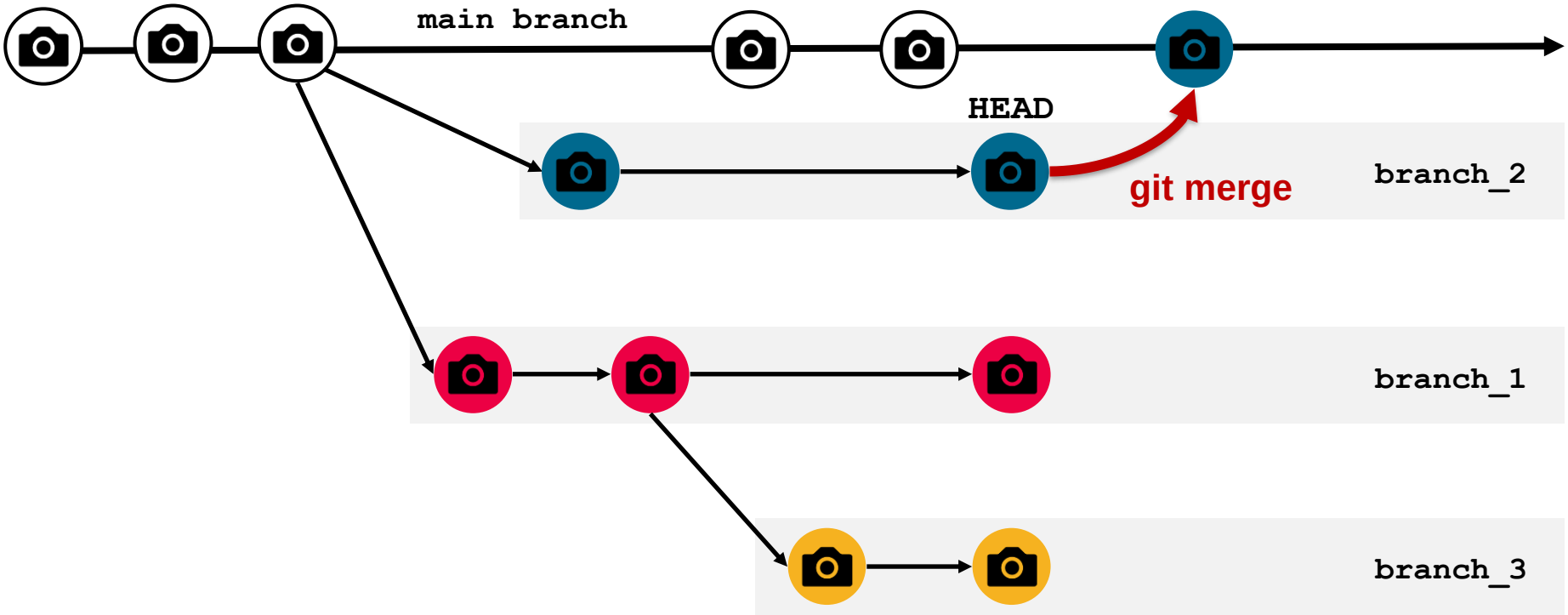
HEAD – the most recent commit on the current branch



Branch when the intention is to eventually "merge" the branch back in with main

Merging

HEAD – the most recent commit on the current branch



Branching and Merging - Commands

See all branches	`git branch`
Create and navigate to a new branch	`git checkout -b new_branch_name`
Switch between branches	`git checkout branch_name`
Merge two branches	`git checkout branch_receiving_changes` `git merge branch_giving_changes`
Delete a branch	`git branch -d branch_name`

Branching - Try it out

Create a new branch called `my_branch`, add a file called `mynewfile.txt` to `my_branch` and commit, and look at the list of branches and log of commits.

1. Create and check out a new branch (one step)

```
git checkout -b new_branch_name
```

2. Add a new file to the new branch

```
touch mynewfile.txt
```

3. Add the new file to staging area and commit

```
git add mynewfile.txt
git commit -m "Adding a second text file"
```

4. Display the branches, then look at the log of commits

```
git branch
git log
```

Merging - Try it out

Navigate to the `main` branch, merge `my_branch` into main, and delete `my_branch`.

1. Navigate to the main branch

```
git checkout main
```

2. Merge `my_branch` into the main branch

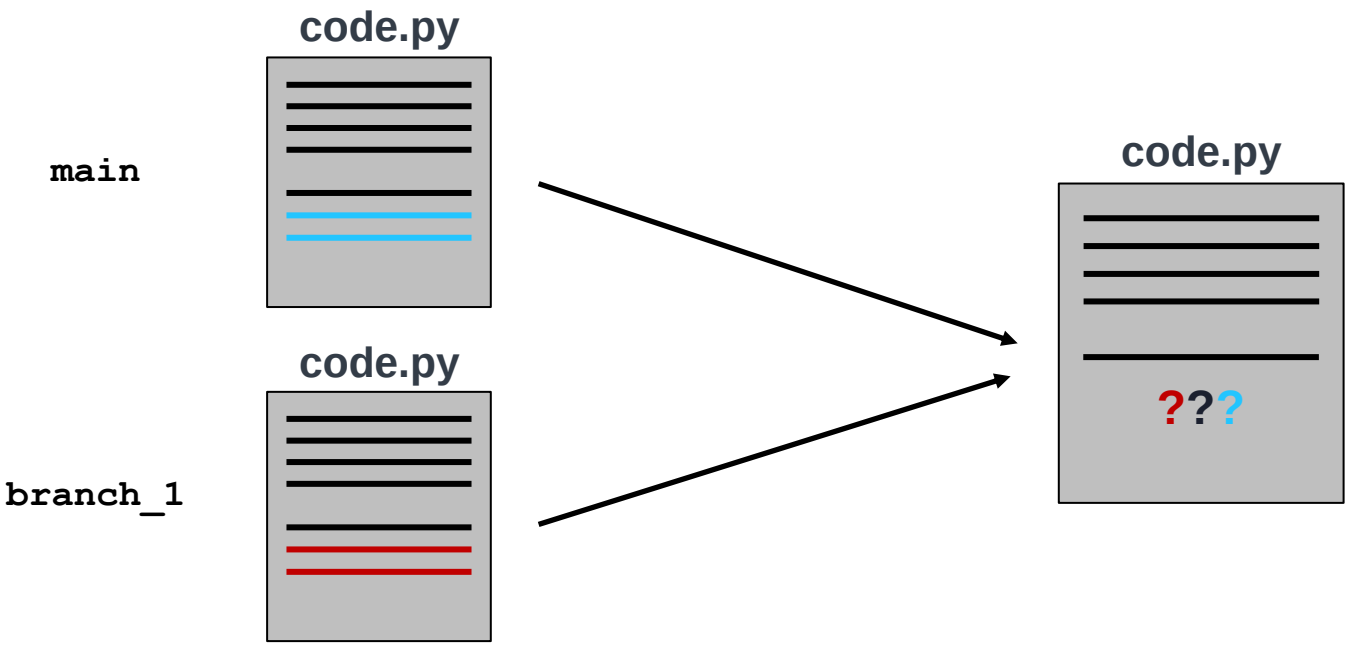
```
git merge my_branch
```

3. Delete `my_branch`

```
git branch -D my_branch
```

Merge Conflicts

When merging changes, we may sometimes find that our branches conflict.



Merge Conflicts

HEAD

```
The first line of the file is the same
The second line is different
```

Commit `dabb4c8c`

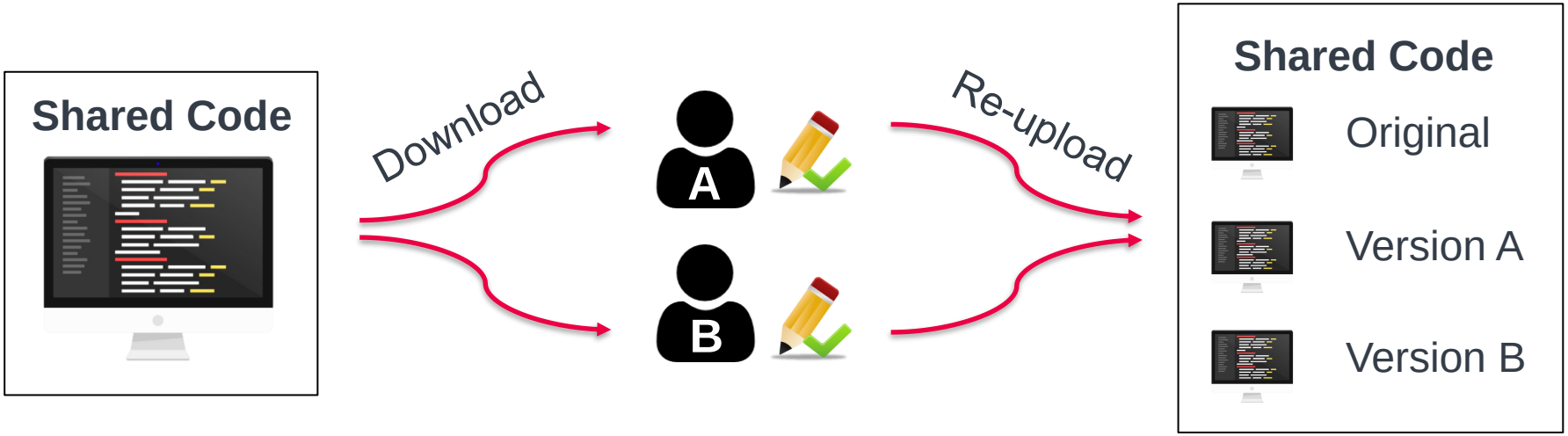
```
The first line of the file is the same
See? It's different
```

```
The first line of the file is the same
<<<<<<< HEAD
The second line is different
=======
See? It's different
>>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d
```

When this happens, Git shows you exactly what the conflicts are and where they are located. It is up to you to reconcile the conflicts and commit them.

Why use Github?

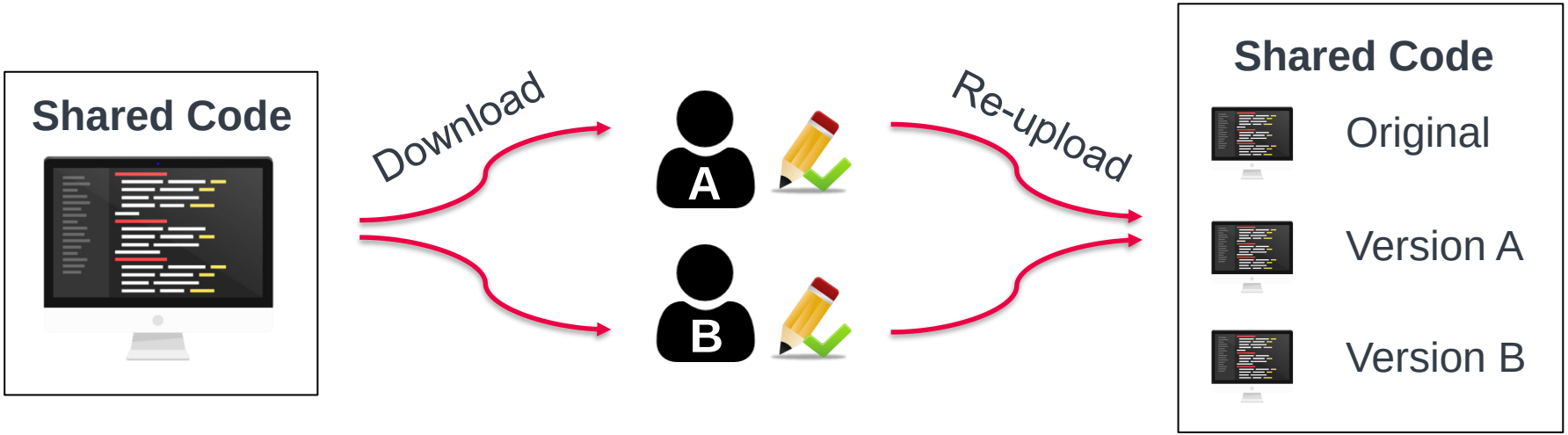
You're working on a research project with another graduate student and both have access to the code in a shared space, e.g. Dropbox.



What are some potential problems that could arise using this process?

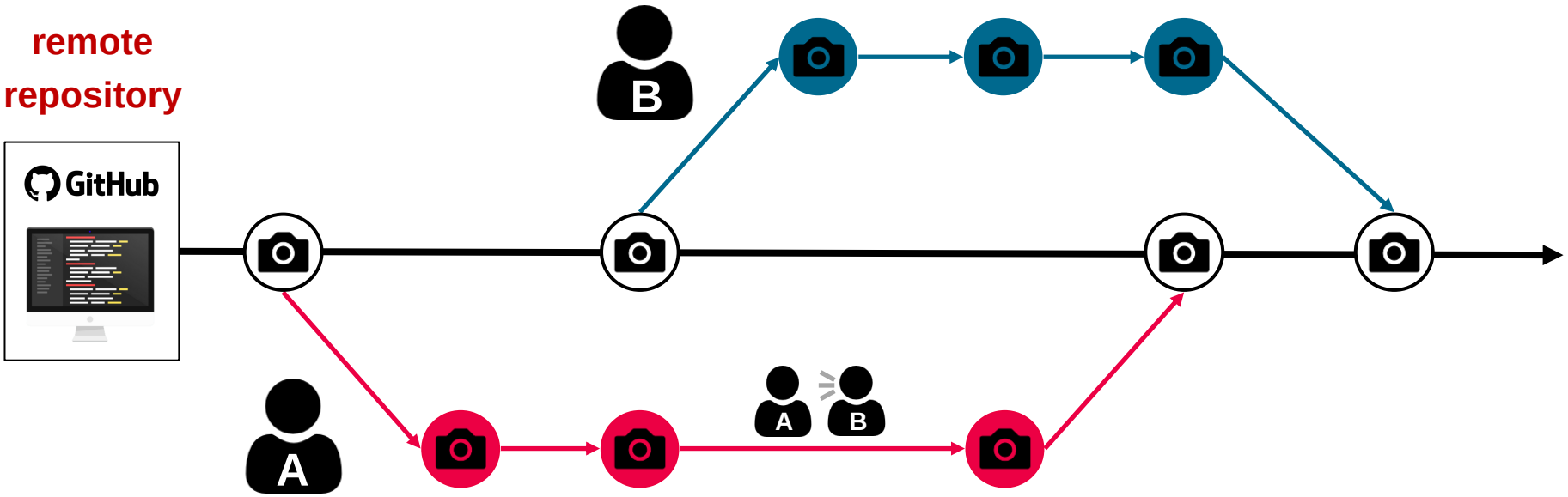
Why use Github?

You're working on a research project with another graduate student and both have access to the code in a shared space, e.g. Dropbox.



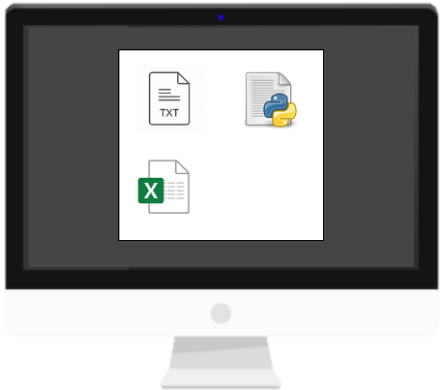
Potential issues: Documenting all changes, staying up to date with changes, overwriting others' work, conflicts between versions, etc.

High-Level Idea



Local and Remote Repositories

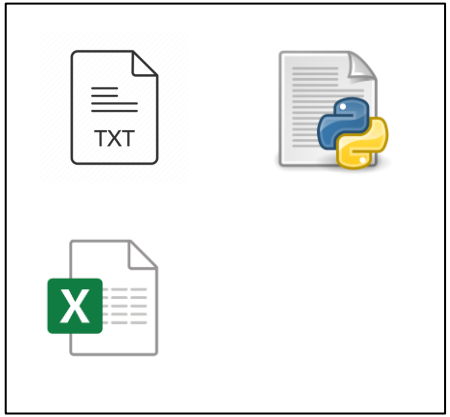
local repository



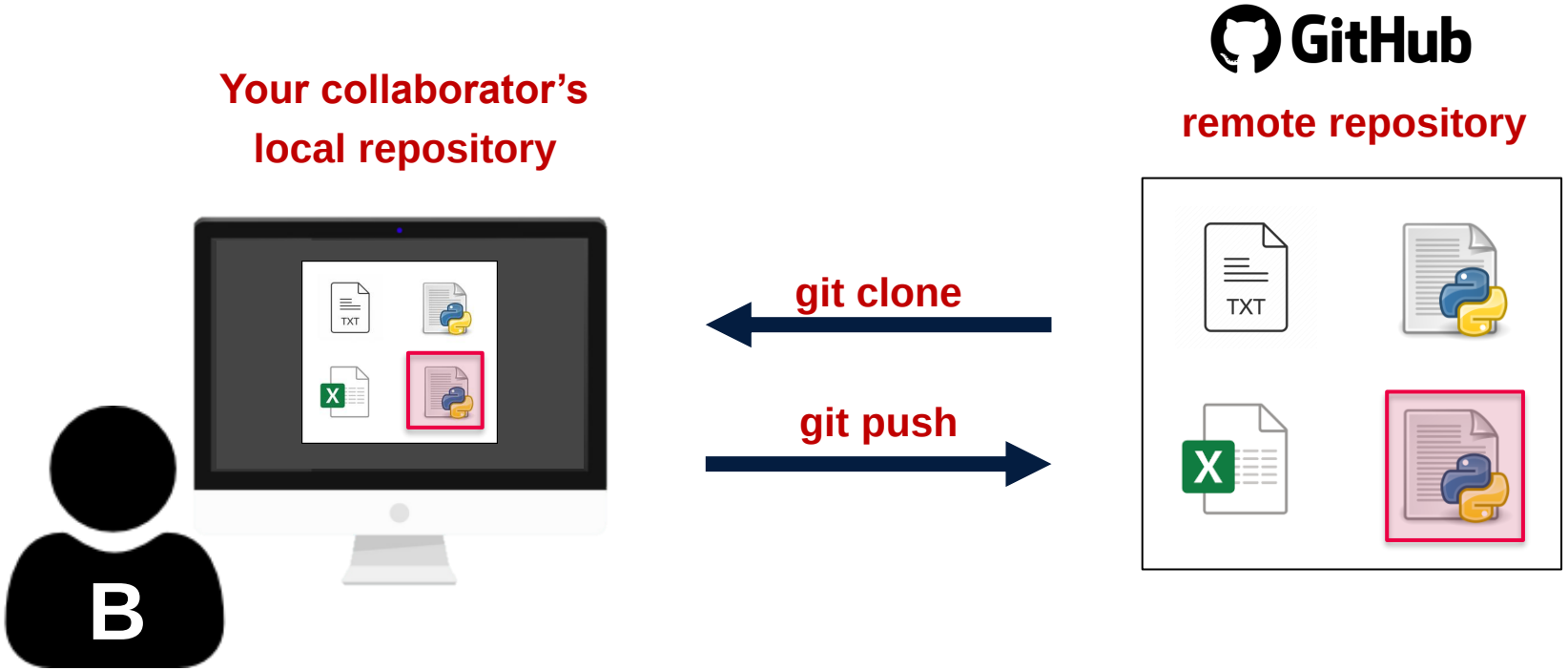
git remote add

GitHub

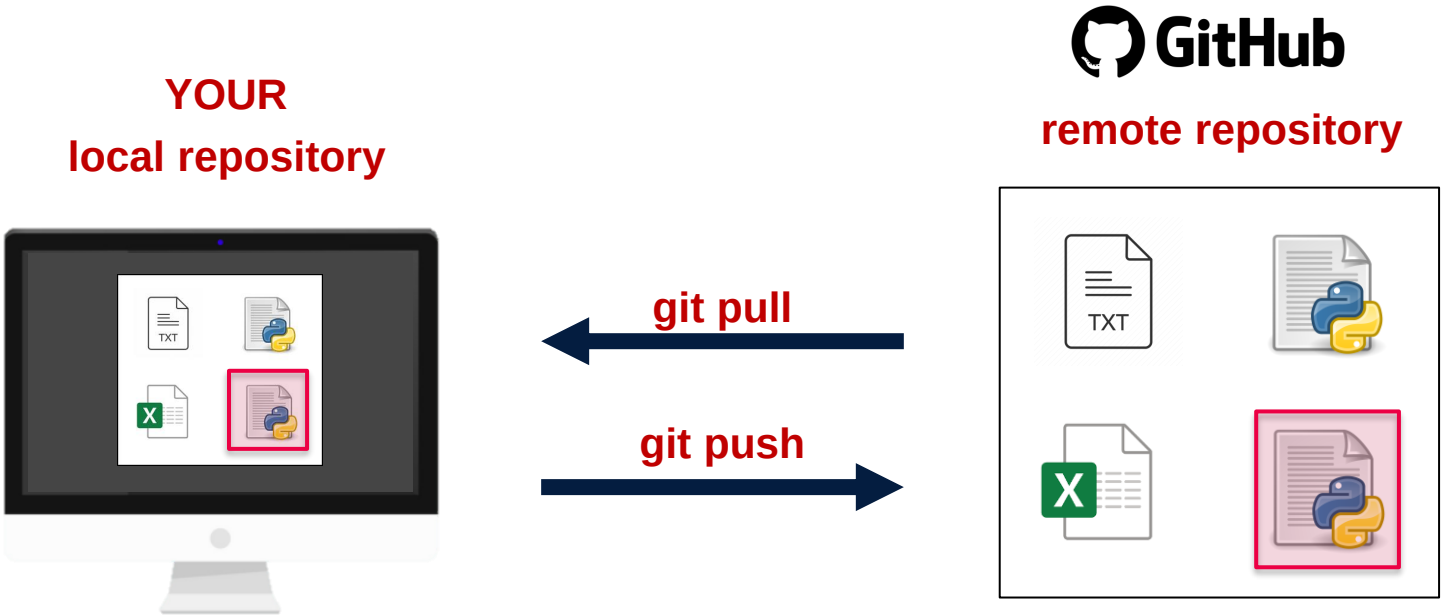
remote repository



Local and Remote Repositories



Local and Remote Repositories



Local and Remote - Commands

Add a remote connection to a Github repo for an existing local repo

```
git remote add origin <Github SSH>
```

Clone existing Github repo

```
git clone <Github URL>
```

Pull from Github repo to local repo

```
git pull
```

Push from local repo to Github repo

```
git push origin <branch_name>
```

Local machine's name for the remote repo (`origin` is usual convention)

Cloning a Github repo - Try it out

Clone the class repo into the location of your choice.

1. Navigate to wherever you want to clone the repo

```
cd ~/courses/
```

2. Clone the Github repository to your machine

```
git clone https://github.com/kscummings/15.S60_2022.git
```

3. Find it on your computer and check it out! You should see the slides for this session.

Cloning a Github repo - Try it out

Create your own repository in Github called `testrepo` (we'll walk through this together). Clone the repo to your machine.

1. Navigate to wherever you want to clone the repo

```
cd
```

2. Clone the Github repository to your machine

```
git clone https://github.com/<username>/testrepo.git
```

Putting it all together – Try it out

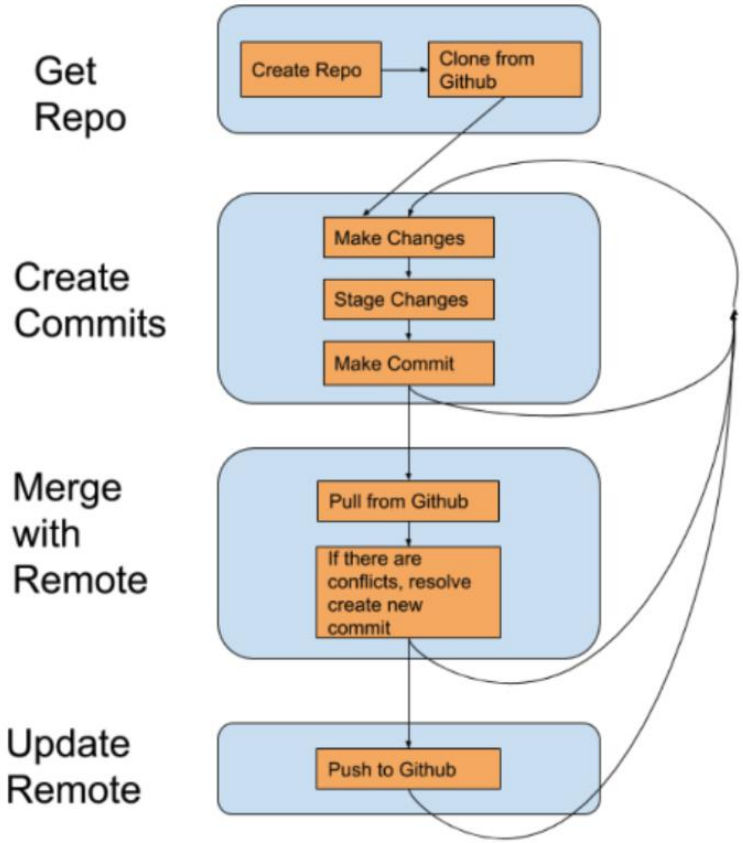
Create a new local branch called `new_branch`, add some text to `mynewfile.txt` on `new_branch` and commit, merge the changes into `main`, and push to Github.

```
git checkout -b new_branch
nano mynewfile.txt
```

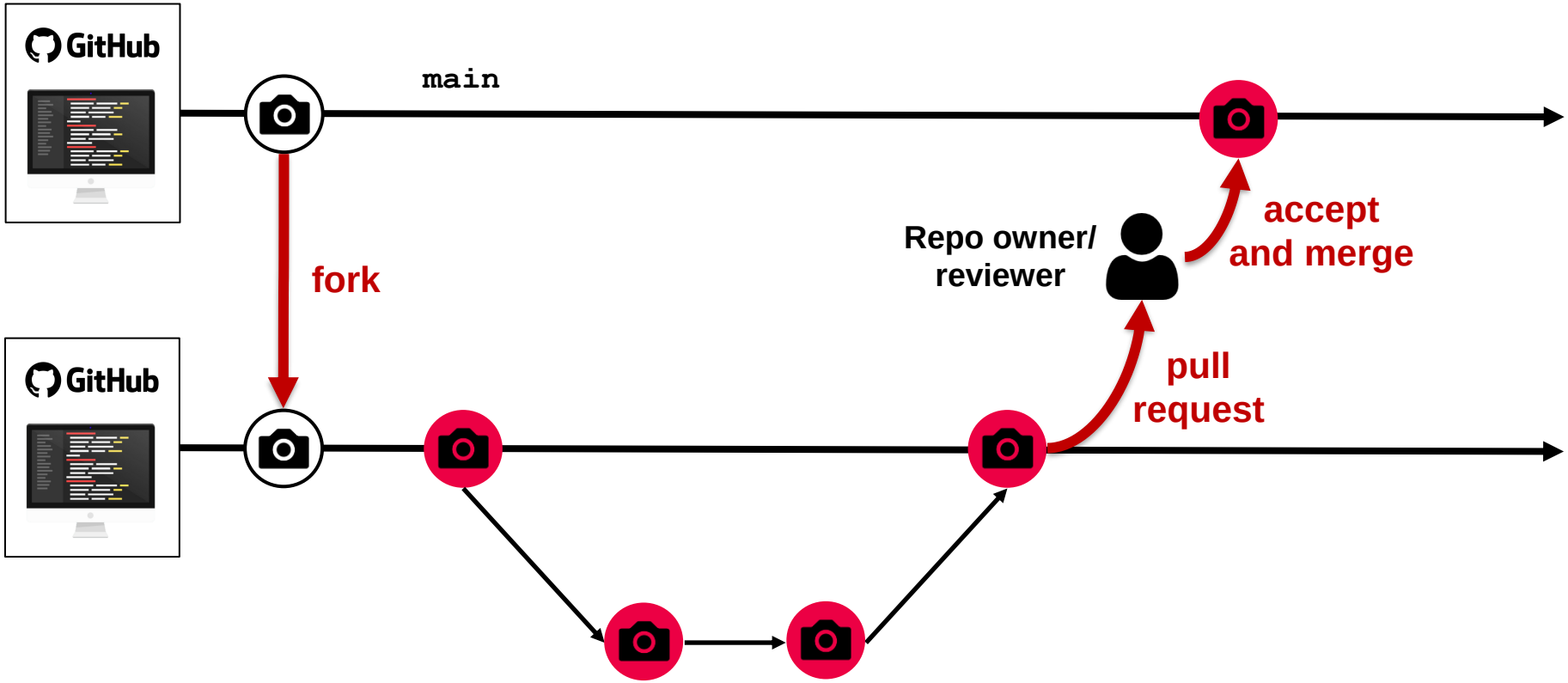
Add a line of text in the editor, save, and exit

```
git add mynewfile.txt
git commit -m “Adding a text file”
git checkout main
git merge new_branch
git push origin main
```

Review of Basic Workflow



Another Workflow – Forking / Pull Requests



Final Tips

- Google is your friend (e.g. “How to undo merge in Git”.)
- Almost anything can be undone, as long as it is committed.
- Commit often, pull often.
- Might take a while to get used to, but is useful knowledge that will improve productivity and collaboration.

Break

Resume in 10 minutes