

Intermediate Programming

Day 3

Announcement

- Lecture slides can be found at:
Website -> Course Materials -> Additional Resources

Outline

- Emacs basics
- Working with `git`
- Zipping files
- Transferring files
- Review questions

Emacs basics

- In the command line prompt, type:

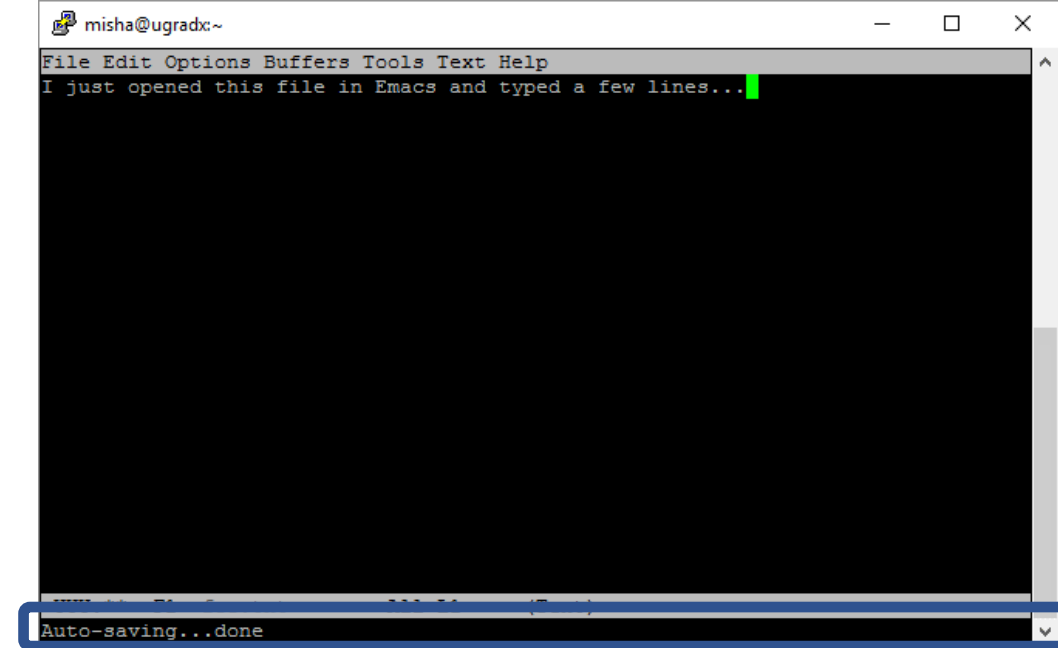
```
>> emacs <filename>
```

to start editing.

- If the file exists already, you will be editing that.
- Otherwise, a new file with the specified filename will be created first.

Emacs basics

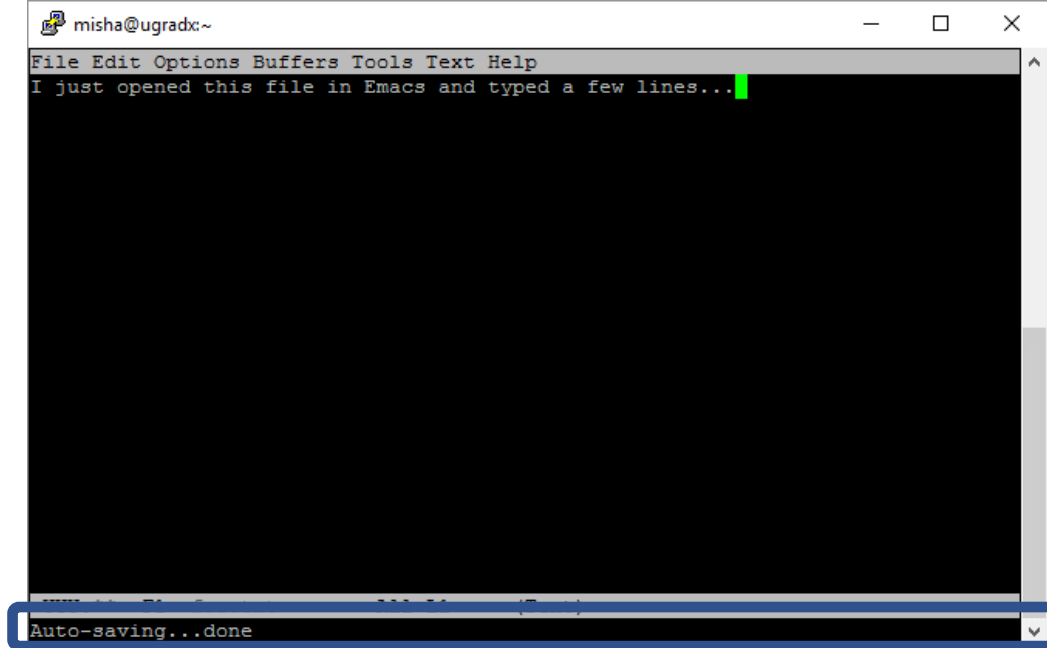
- Can type directly
- Arrows move you around
- Basics
 - [CTRL]-x + [CTRL]-c: quit
 - [CTRL]-x + [CTRL]-s: save
 - [CTRL]-x + [CTRL]-f: open a file*
 - [CTRL]-x + u: undo
 - [CTRL]-_: undo
 - [CTRL]-g: cancel



*prompts show up at the bottom

Emacs basics

- Cut / Paste:
 - [CTRL]-k: kill the rest of the line
 - [CTRL]-[SPACE]: set (start) mark
 - [CTRL]-w: cut from the mark to the cursor
 - [ESCAPE]+w: copy from the mark to the cursor
 - [CTRL]-y: copy the previously cut/copied buffer
- Search / Replace:
 - [CTRL]-s: search forward*
 - [ESCAPE]-%: query replace*
- And much, much more:
 - [ESCAPE]+<command name>: ...



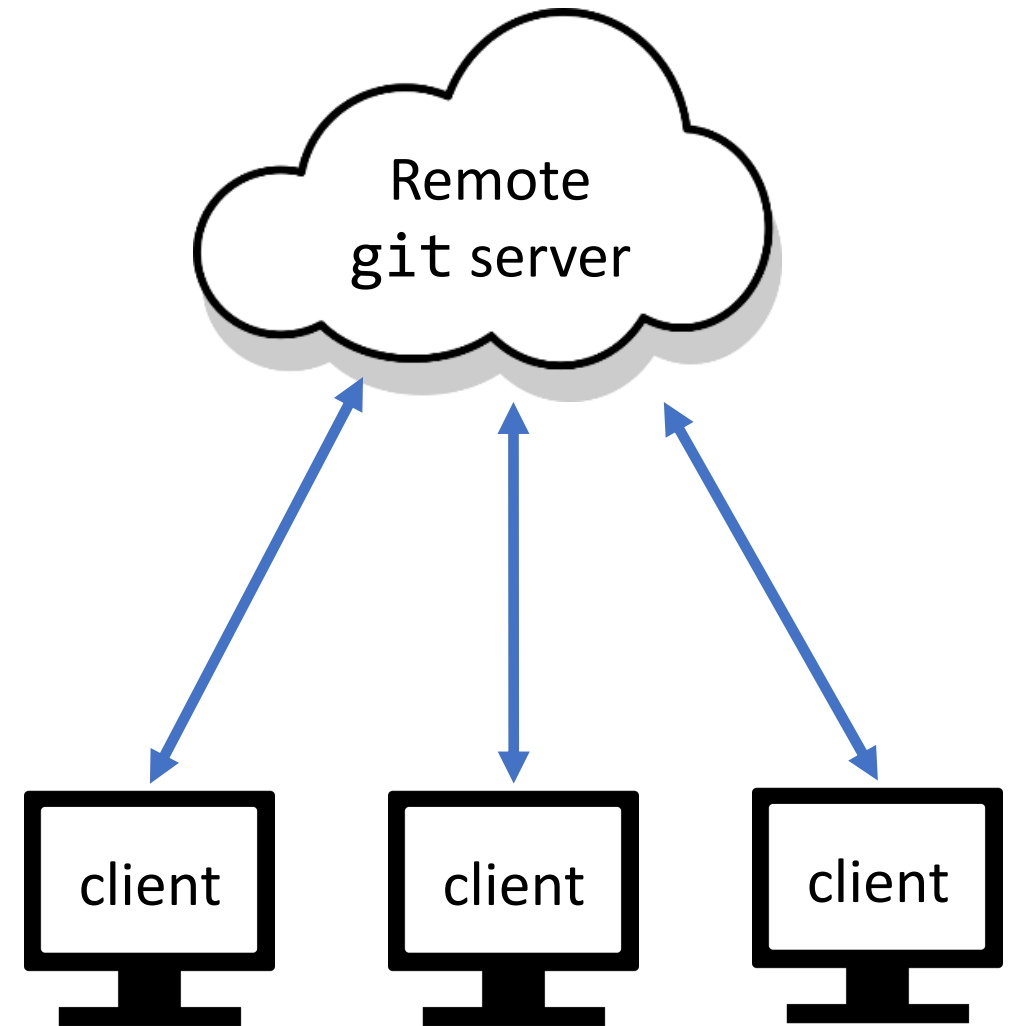
Outline

- Emacs basics
- Working with `git`
- Zipping files
- Transferring files
- Review questions

Working with `git`

Repo (why)?

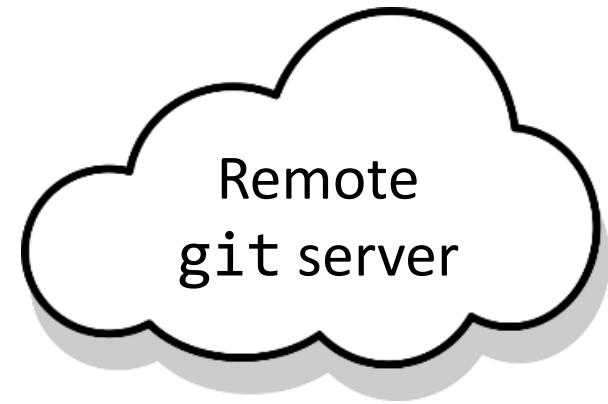
- Supports collaboration / sharing
- Version control
 - Stores the history as a collection of concise, semi-automatically annotated, “snapshots” of the repo



Working with `git`

Repo (what)?

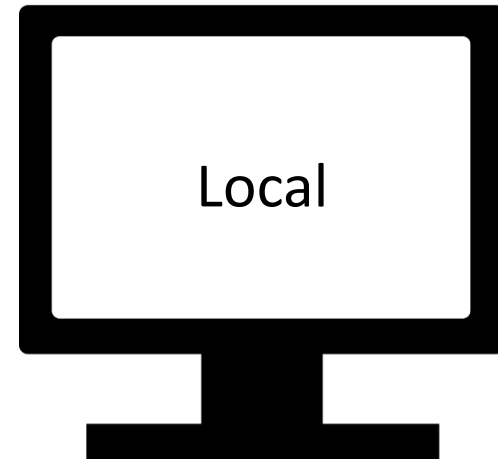
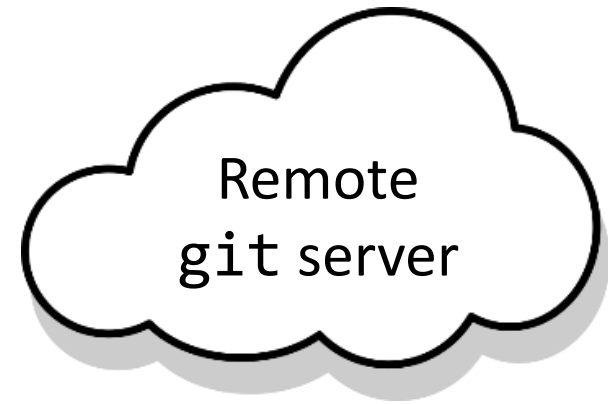
- A copy of the entire history of the files/project
 - A *local* copy is stored on your (client) machine.
 - A *remote* copy is stored on the `git` server (e.g. `github.com`)



Working with `git`

Repo (how)?

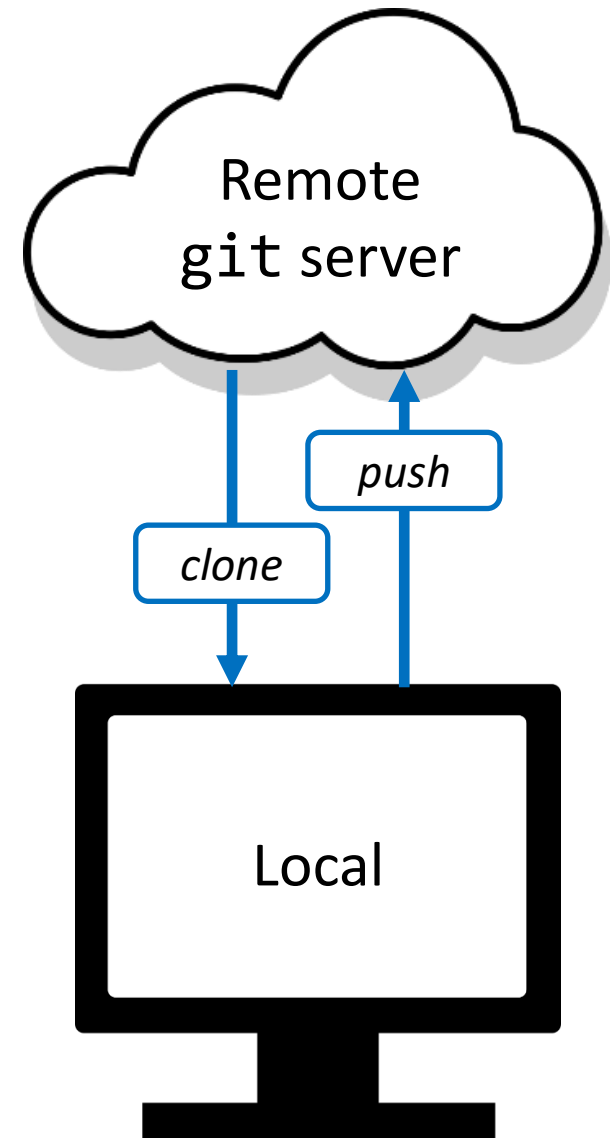
- The repo directory contains:
 - Local Repo: A copy of the entire repository (in the `.git` subdirectory)
 - Working copy: A replica of all the “latest” files.



Working with `git`

Getting a remote repo to our local machine:

- Already have an empty repo on `github.com`
- Clone the repository to `ugradx`
- Add / modify local files
- Push the edited files in the repo.



Working with `git`

Before we start:

- We need to tell the `git` system (on `ugradx`) a bit about ourselves:
 - What name do we want the system to know us by:

```
git config --global user.name "<your name>"
```

(quotes are required if there is a space in `<your name>`)
 - What email do we want the system to know us by:

```
git config --global user.email <your email>
```

Already have an empty repo on github.com

The screenshot shows a web browser window with the URL `github.com/jhu-ip/2021-instructor-misha`. The page features a dark navigation bar with the GitHub logo, a search bar, and links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. A large green banner with the text 'Learn Git and GitHub without any code!' and a 'Read the guide' button is prominent. Below the banner, the repository name 'jhu-ip / 2021-instructor-misha' is displayed as 'Private'. The repository has 2 watchers, 0 stars, and 0 forks. A navigation menu includes 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', 'Insights', and 'Settings'. The main content area shows the 'master' branch with 1 branch and 0 tags. A commit history table lists a commit by 'mkazhdan' that updated the README. The README content is visible, including the author's name 'Misha Kazhdan' and a brief description. On the right, the 'About' section is empty, and the 'Releases' and 'Packages' sections indicate no published items.

github.com/jhu-ip/2021-instructor-misha

Search or jump to... Pull requests Issues Marketplace Explore

Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

jhu-ip / 2021-instructor-misha Private

Unwatch 2 Star 0 Fork 0

Code Issues Pull requests Actions Projects Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

Commit	Author	Message	Time	Commits
e015e98	mkazhdan	updated README	13 hours ago	2

README updated README 13 hours ago

README

Misha Kazhdan
XXXXXX
1997
Mathematics
This is my personal repository for ...

About

No description, website, or topics provided.

Readme

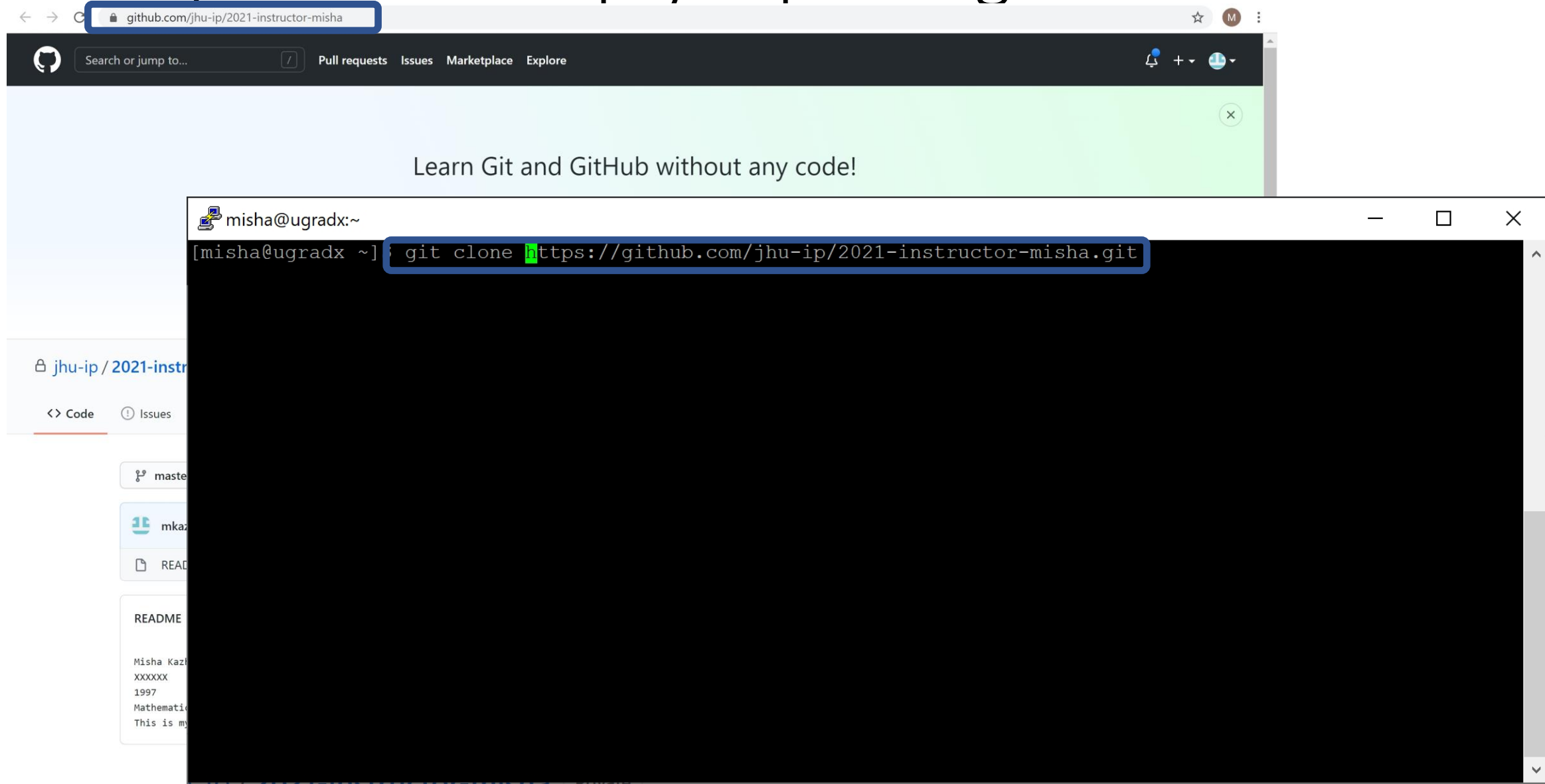
Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Already have an empty repo on github.com



The image shows a web browser window displaying the GitHub repository page for `jhu-ip/2021-instructor-misha`. The browser's address bar shows the URL `github.com/jhu-ip/2021-instructor-misha`. The page features a navigation bar with the GitHub logo, a search bar, and links for "Pull requests", "Issues", "Marketplace", and "Explore". A prominent green banner across the top of the repository page reads "Learn Git and GitHub without any code!". Below the banner, the repository name `jhu-ip / 2021-instr` is visible, along with tabs for "Code" and "Issues".

Overlaid on the bottom right of the browser window is a terminal window. The terminal prompt is `misha@ugradx:~`. The command being executed is `git clone https://github.com/jhu-ip/2021-instructor-misha.git`, which is highlighted with a blue selection box.

Working with `git`

Once you have a repository, you can:

- commit changes you've made locally (to create a restore point)
- push those back up to the `git` server (to synchronize your local repo)

Note:

- `git` does not know what changes you want to commit.
- You will have to let it know explicitly.

Working with `git`

To run a `git` command you type “`git`”, followed by the command, followed by the parameters:

- `git clone <address>`:
 - clones a repository from the remote server into the local directory
- `git pull`:
 - fetches the most recent version of the repo from the server and merges it with the repo (and working copy) on the local machine
- `git status`:
 - display the current status of the repository
- `git log`:
 - display the history of changes (commits)

Working with `git`

To run a `git` command you type “`git`”, followed by the command, followed by the parameters:

- `git add <file name>`:
 - add the specified file to the list of files that you will be committing (do this to add a file to the repo or if you’ve changed an existing file)
- `git commit -m “<commit message>”`:
 - commit the changes **locally**, including a brief message describing the changes.
- `git commit -am “<commit message>”`:
 - like `git commit -m “<commit message>”` but automatically adds any files that have been modified (not added) since the last commit
- `git push`:
 - push your local repo back to the server

Working with `git`

Note that you should not modify a repo directly using standard `mv` or `rm` commands. All interactions should be via the `git` command:

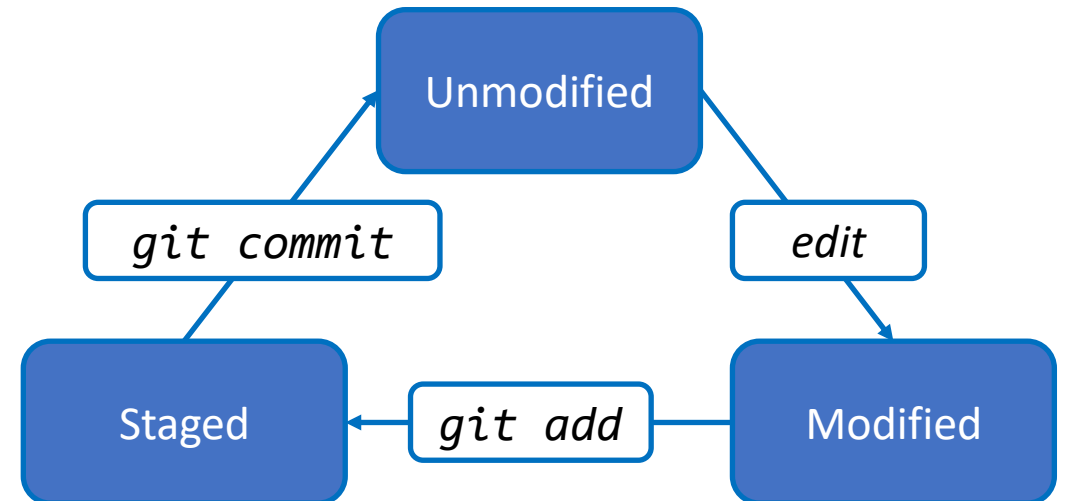
- `git mv <source file> <target file>`:
 - rename a file
- `git rm <file>`:
 - remove a file (delete it)

For a more complete list, see:

<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

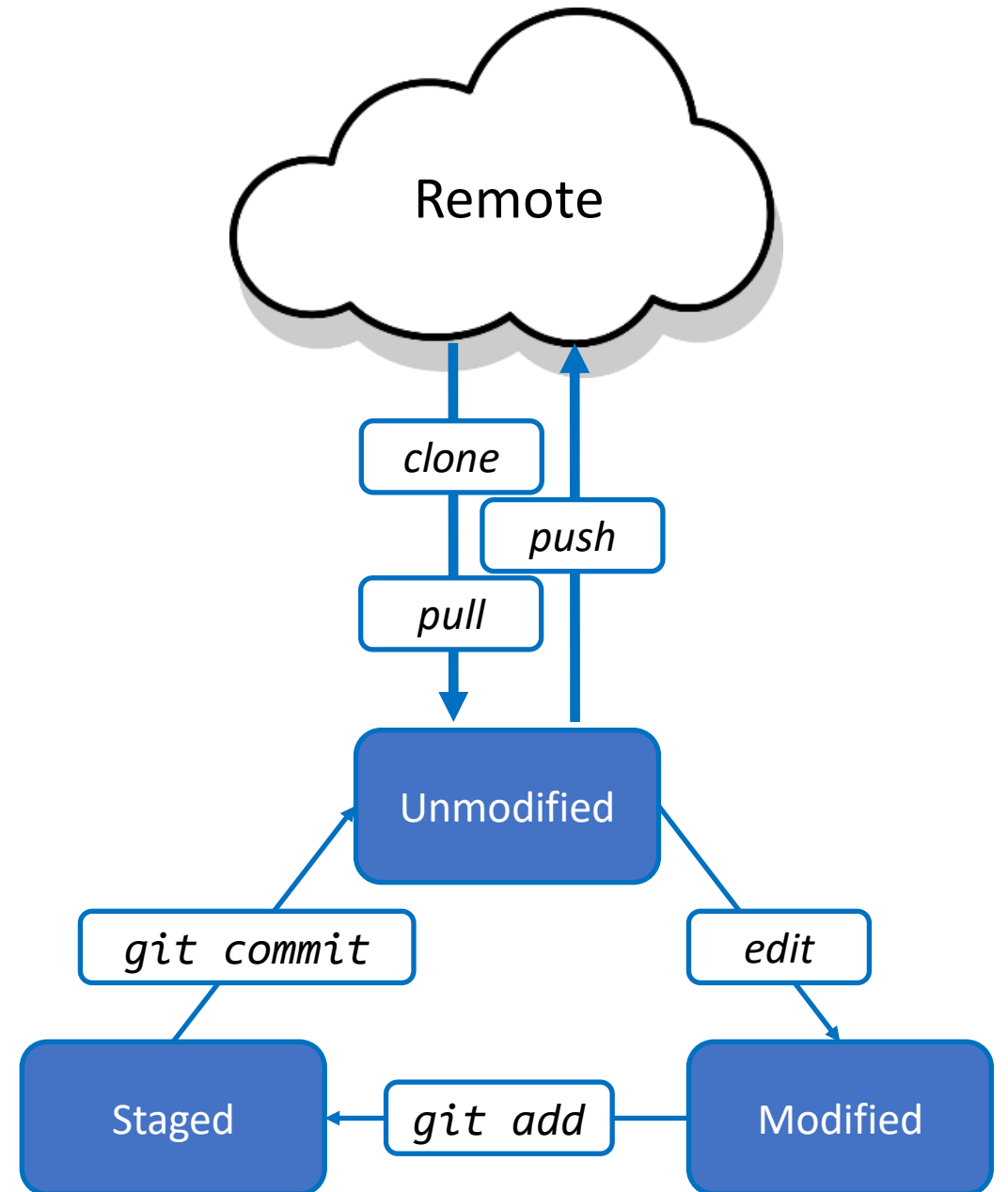
Working with `git`

- Files in your working copy (you `git add`'ed them) are *tracked* and can be in one of several states:
 - *Unmodified* (same as copy in local repo)
 - *Modified* (different from copy in local repo but not yet staged)
 - *Staged* (next `git commit` will update the repo)



Working with `git`

- The `git push` / `pull` / `clone` commands synchronize the local repository with the remote one.
- These commands synchronize the local/global repos, so be sure that your working copy matches the local repo before calling them (i.e. files are in “unmodified” state)



Working with `git`

- Files that are *not* yet part of your project are *untracked*
 - When you create a new file; it's *unstaged* until you `git add` it
 - This is true even if the file is in the directory containing your working copy
 - However, `git` will notice it, and it will appear as unstaged if you `git status`
- Some untracked files are files that we want `git` to ignore because we'll never want to include them in the remote repo
 - Tell `git` to ignore a file by adding it to `.gitignore` file
 - Good candidates for ignoring might be `a.out`, `gitlog.txt`

Working with `git`

Workflow Suggestions:

- Start each session with `git pull`, to ensure your local copy is up-to-date
- After you complete work on a small task, `git commit` it
- Include a message with every commit to explain what changes you committed (use `-m`, or you might be forced into `vim` editor to create one!)
- Make sure you `git commit` and `git push` at the end of each work session

Outline

- Emacs basics
- Working with `git`
- **Zippping files**
- Transferring files
- Review questions

Ziping files

The `zip` command lets us package multiple files (and even directories) into a single (compressed) file.

Why?

- It is easier to move a single file
- The compression may reduce the size of the contents

Zippping files

Basic operations:

- Create a zip file:

```
zip <zip file name>.zip <file name 1> <file name 2> ...
```

- Extract the contents of a zip file:

```
unzip <zip file name>.zip
```

- List the contents of a zip file:

```
unzip -l <zip file name>.zip
```

Outline

- Emacs basics
- Working with `git`
- Zipping files
- **Transferring files**
- Review questions

Transferring files

While we will do our work on the ugradx machines, we will sometimes want to copy our files to other machines.

- Audio
- Video
- Printing
- Uploading to GradeScope
- etc.

Or we may want to move content from our machines to ugradx...

Since the content on our ugradx is ours, we need to do this securely (using authentication to validate that the files are ours).

Transferring files

To perform local/local copying, we use the copy command, `cp`.

To perform local/remote or remote/local copying we use the secure copy command, `scp` (linux) or `pscp` (windows)

`[p]scp misha@ugradx.cs.jhu.edu:<source> <target>`
moves from `ugradx` to local machine.

`[p]scp <source> misha@ugradx.cs.jhu.edu:<target>`
moves to `ugradx` from local machine.

Transferring files

Example:

If the source file is `cs220/foo.c` on `ugradx` and we want to copy it to the current directory on our local machine:

```
scp misha@ugradx.cs.jhu.edu:cs120/foo.c .  
pscp misha@ugradx.cs.jhu.edu:cs120/foo.c .
```

If we want to copy the source file to directory `foo` and call it `bar.c`:

```
scp misha@ugradx.cs.jhu.edu:cs120/foo.c foo/bar.c  
pscp misha@ugradx.cs.jhu.edu:cs120/foo.c foo\bar.c
```

Transferring files

Example:

If the source file is `cs220/foo.c` on `ugradx` and we want to copy it to the current directory on our local machine:

```
scp misha@ugradx.cs.jhu.edu:cs120/foo.c .  
pscp misha@ugradx.cs.jhu.edu:cs120/foo.c .
```

These commands have to be run from your **local** machine:

- The local machine knows about `ugradx`.
- `ugradx` does not know about your local machine.

Outline

- Emacs basics
- Working with `git`
- Zipping files
- Transferring files
- Review questions

Review questions

1. Why do we use a version control system like `git`?

- Version control
- Supports collaboration / sharing

Review questions

2. Name six common `git` commands.

- `clone`
- `pull`
- `status`
- `log`
- `add`
- `commit`
- `push`

Review questions

3. Where are the files that must be included in your submission?

Your submission must include your source code `.c` files and your `gitlog.txt` bundled in a single `.zip` file. This file should be downloaded to your personal computer from ugrad using `[p]scp` and then submitted through Gradescope.

Review questions

4. How do you save and quit on the emacs editor?

- [CTRL]-x + [CTRL]-s: save
- [CTRL]-x + [CTRL]-c: quit

Review questions

5. How do you search and replace on emacs?

- [CTRL]-s: search forward
- [ESCAPE]-%: query replace

Exercise 3a & 3b

- Website -> Course Materials -> 3a
- Website -> Course Materials -> 3b