

Lab 07

GitHub Collaboration

Today's plan

- | | | |
|----|------------------------------|-----------------------------|
| 01 | Cloning & Forking | Differences and preferences |
| 02 | Conflict resolution | For git, not for life |
| 03 | Issues & PRs | |
| 04 | Dotfiles | Config files |

Clone vs. Fork

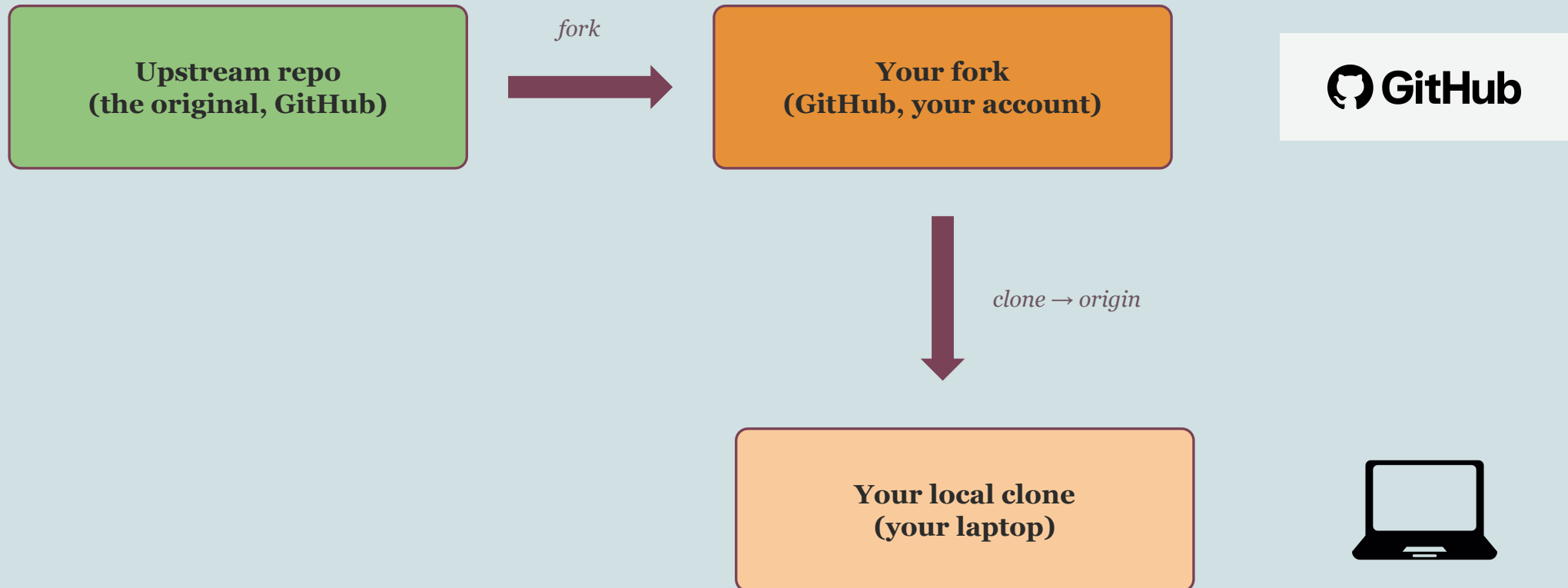
Clone

- Downloads a full copy of a repo you already have access to
- Nothing new is created on GitHub
- Establishes a connection between local and remote repos
- Use for: your own repos, your team's shared repo

Fork

- Creates your own copy of someone else's repo, under your account
- You can then clone your fork and work freely
- Use for: contributing to a repo you don't own or can't write to

From their repo to your laptop



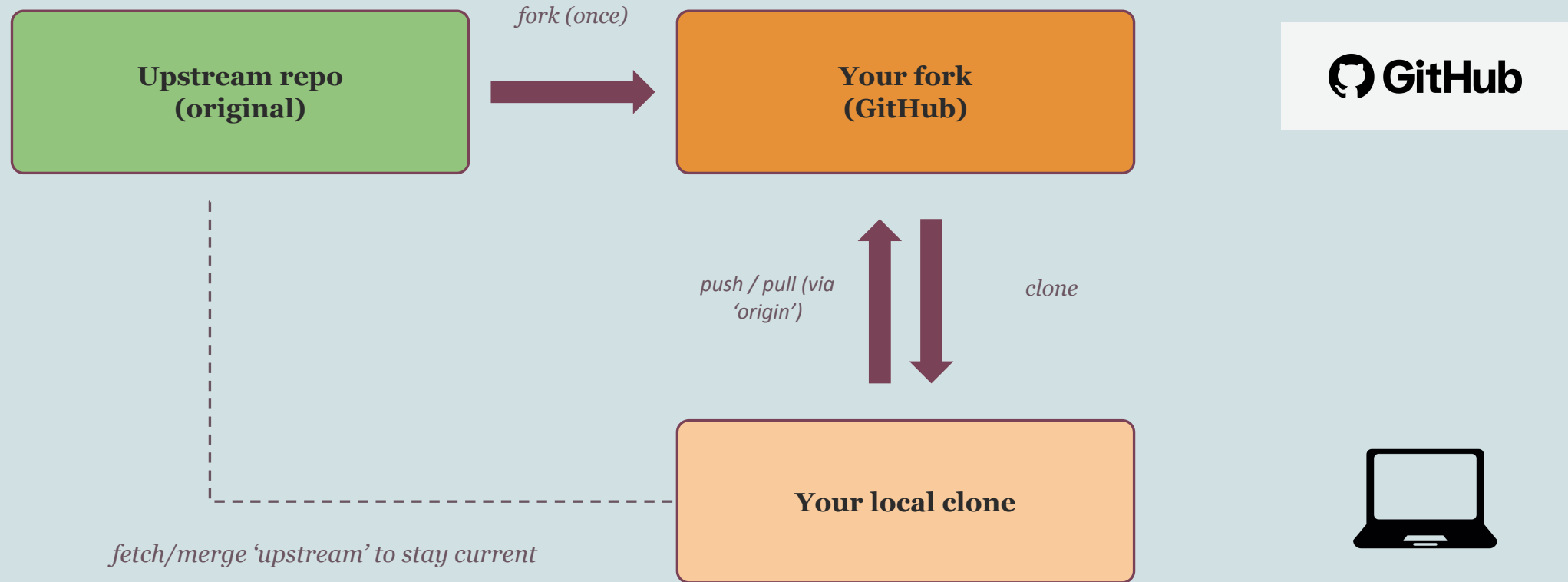
Which one do I want?

- **Do you have write access to the repo?**
 - Yes → clone it directly.
 - No, but you want to propose a change → fork it, clone your fork.

Rule of thumb

- Clone = “I’m part of it already.”
- Fork = “I’m making my own copy to work from.”

A remote mental model (for forks)



- Two labels to remember:
origin (your fork) and **upstream** (the original)

Keeping your fork in sync with the original repo

One-time setup

- `git remote add upstream <original-repo-url>`

Every time you want the latest changes

- `git fetch upstream + git merge upstream/main` (a more conservative approach)

OR

- `git pull upstream main` (does all at once)

Another option would be to update/sync the fork directly on github then pull the changes from origin

Conflict resolution

What we know

Happens when the **automatic merging of git** doesn't work, because there are different changes in the same parts.

Git is great for a lot of things, but it can't make decisions on what to keep. That's where the human-part comes in.

Under what circumstances can we find them

- When pulling or merging when the *remote* and *your local* branch both changed the same line
- Or if your own Github repo you changed something directly and locally without syncing the changes

VSCode for conflicts

```
13
14  /**
15   * Prints the welcome message
16   */
17  Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
18  <<<<<< HEAD (Current Change)
19  function printMessage(showUsage, message) {
20      console.log(message);
21  }
22  =====
23  function printMessage(showUsage, showVersion) {
24      console.log("Welcome To Line Counter");
25      if (showVersion) {
26          console.log("Version: 1.0.0");
27      }
28  }
29  >>>>>> theirs (Incoming Change)
30  if (showUsage) {
31      console.log("Usage: node base.js <file1> <file2> ... ");
32  }
33  }
```

Resolve in Merge Editor

🔍 You, 20 seconds ago Ln 11, Col 26 Spaces: 4 UTF-8 CRLF {} JavaScript

Text editors are also an option

Merging: index.html — code

index.html ! • Merging: index.html ! • JS app.js 3, ! mytext.txt !

Incoming ◊ ad547d9 origin/new-feature, new-feature

Current ◊ 999403d main, origin/main

file from different branches

13

14

Accept Incoming | Accept Combination (Incoming First) | Ignore

15

16

17

18

19

20

21

22

file from different branches

13

14

Accept Current | Accept Combination (Current First) | Ignore

15

16

17

18

19

20

21

22

Result index.html

13

14

No Changes Accepted

15

16

17

18

19

20

21

22

1 Conflict Remaining

Resolving a (git) conflict workflow

1. Open the affected file, decide what it should actually say:
 - a. keep one side, the other, or something new that combines both
2. Delete all three marker lines completely.
3. Stage, commit, done: a conflict resolution is just a regular commit.

Conflicts in Jupyter Notebooks

The screenshot shows a Git GUI interface for the repository `~/repos/github/amazon-sagemaker-examples` on the `master` branch. The left sidebar displays a commit history with the following entries:

- 1 unstaged file
- Commit Changes
- Merge branch 'master' into remove-long-running (Ajay Karpur)
- remove examples that run longer than 2 hours (Ajay Karpur)
- Update index.rst (#2143) (Paul B)
- removed geolocator (#2139) (Hongshan Li)
- Customer churn (#2071) (Ali Arsanjani)
- Remove external dependencies for XGBoost Abalone NBs (#2071) (Julia Kroll)
- update studio notebooks for neo mxnet v1.8 (#2129) (Pratyush Bagaria)
- New feature store notebook on how to encrypt your feature (#2129) (Paul B)
- update neo mxnet notebooks for v1.8 (#2125) (Pratyush Bagaria)
- Adding Tensorflow Unet notebook (#2123) (Prateek Chauhan)
- [Debugger] add a boto3 version of TF profiling multi-GPU-multi (Miyong)
- Update feature_store_introduction.ipynb (#2127) (Paul B)
- Removing unused lr variable (Talia)
- Fixing learning rate declaration. (#2122) (Talia)
- Fixing learning rate declaration. (Talia)
- Training job with Encrypted Static Assets (#2119) (Hongshan Li)
- Add Notebook for using Elastic Inference with compiled model on (Jiacheng Guo)

The main area shows a conflict in the file `prep_data/image_data_guide/03c_pytorch_preprocessing.ipynb`. The conflict is between the `origin/master` branch and the `origin/remove-long-running` branch. The code is displayed in a diff view, with the `origin/master` version in green and the `origin/remove-long-running` version in red. The conflict is highlighted in a yellow box. The code is a Jupyter notebook cell with the following content:

```
"cell_type": "code",
"execution_count": 5,
"metadata": {},
"source": [
    "import tqdm"
],
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [
    {
      "data": {
        "image/png": "iVBORw0KgAAAAANSUHEUgAAAsIAAAFPcAYAAACWU5pAAAABHNCVSQICAgI fAhk1AAAAA lwsF l zAAAEgAACxIB0t1+ /AAAADh0RVh0U29mdHdhc",
        "text/plain": [
          "<Figure size 720x360 with 2 Axes>"
        ]
      }
    ]
  }
}
```

Try for yourself Lab Notebook #7 - Part 1 & 2

Issues & PRs

Issues as a way to track work, not only mistakes

- An Issue is a tracked conversation about something to do: a bug, a question, a feature idea, a task.
- Anatomy: title, description, (optional) labels and assignees, a comment thread.
- Anyone with access to a repo can open one, it doesn't require write access to the code. - Like on your peer reviews.
- Closing an issue doesn't require a PR, but PRs can close issues automatically (example: writing "Closes #12" in a PR description).

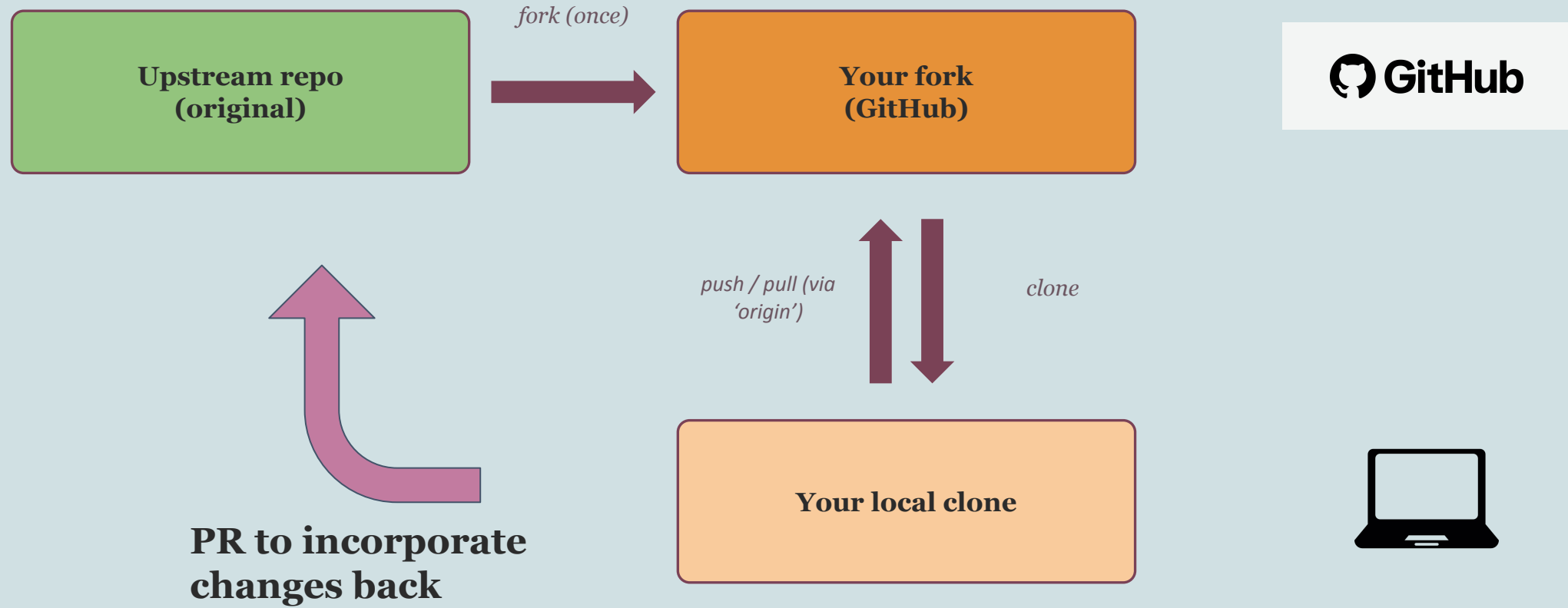
Pull Requests (PRs) to propose changes

- A PR is a request to merge changes from one branch (often on your fork) into another (often the original repo's main branch).
- It's a git diff + a conversation: reviewers can comment on specific lines, request changes, approve.
- Nothing merges until someone with write access to the target repo clicks merge.

A full fork circle

- fork → clone → branch/edit → push to your fork → open a PR back to the original → review → merge.
- This is like the other side of the the loop that "Keeping your fork in sync" started:
 - syncing pulls changes **in**, a PR pushes your changes **back out**.

A full fork circle



Try for yourself Lab Notebook #7 - Part 3 & 4

Break

Mid-semester check-in



<https://forms.gle/tAzDyzjStMc1Dpks6>

Reading Checkpoint #6

Dotfiles

What are dotfiles?

- Config files for the tools you use every day:
 - **.gitconfig, .vimrc, shell settings.** Named with a leading dot so they're hidden by default.
- They live in your home directory (\$HOME), not inside any one project.
- Without version control, your setup lives only on one machine and you can lose it.

We can't version our home directory (\$HOME)

- You can't just git init your home directory, it's not a project folder
- The common fix: a "bare" git repository that lives somewhere else, but tracks files in \$HOME as its working tree.

Bare repos

- A bare repo stores git's history but has no working directory of its own. You point it at one separately.
- `git clone --bare <repo_url> $HOME/.dotfiles` clones history into `.dotfiles`, with `$HOME` set as the working tree.
- A shell alias makes this usable day to day:
 - `alias gdot='git --git-dir=$HOME/.dotfiles --work-tree=$HOME'.`
 - From here, `gdot status`, `gdot add`, `gdot commit`, `gdot push` all work like normal git, just scoped to the dotfiles you're tracking, not your whole home directory.
- `gdot config --local status.showUntrackedFiles no`, so `gdot status` doesn't list every unrelated file in your home directory.

Let's try it for our class

- A small, class-maintained dotfiles repo inspired by this [.dotfiles](https://github.com/UCB-stat159-f26/dotfiles) repo. Just not the full personal setup it's inspired by.
- You'll fork it, set it up with the bare-repo pattern, and make it yours.
 - <https://github.com/UCB-stat159-f26/dotfiles/tree/main>

Let's do this part together