LECTURE 41

GIT PART 3:

BRANCHING AND COLLABORATION

MCS 275 Spring 2023

Emily Dumas

LECTURE 41: BRANCHING AND COLLABORATION

Reminders and announcements:

- Project 4 due at 6pm today
- Don't forget your **course evaluation**
- I will post on Blackboard about what comes next (e.g. course material archive)

LAST TIME

We pushed our git repo to GitHub.

We experimented with changes on GitHub and changes locally (without truly "collaborating").

GITHUB COLLABORATION

We can mark other GitHub users as collaborators on a repo, allowing them to push to it.

(Private repos are invisible except to collaborators.)

Key consideration: Online repo can change with no action from us.

REVISED WORKFLOW

- `git pull` – get updates
- `git log` – see what's changed
- Make and test your changes
- `git add file1`
- `git add file2`
- `git commit`
- `git push` – make changes available to others

`git push`

Contact a remote repository and send it commits that are in our database but not theirs.

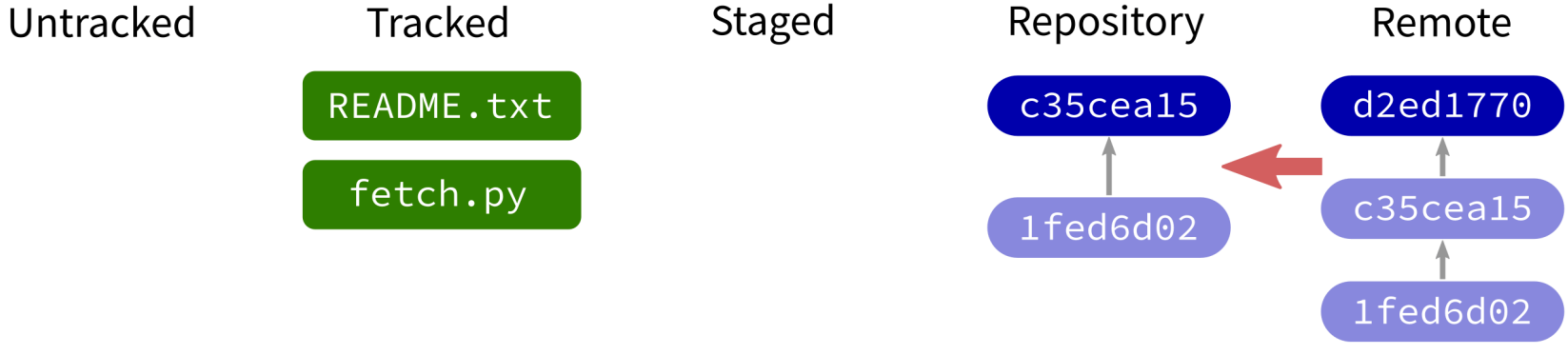
`git push`

Contact a remote repository and send it commits that are in our database but not theirs.

`git push`

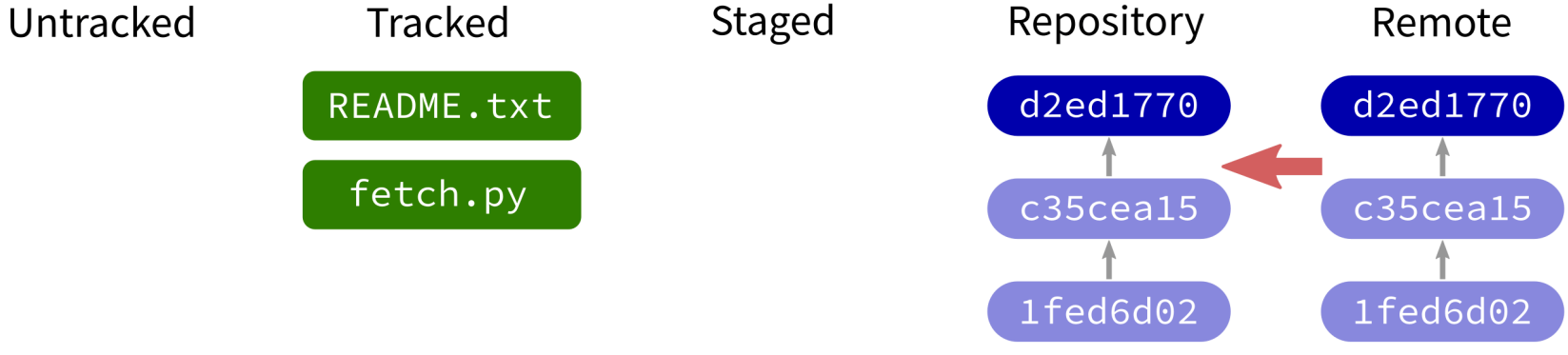
Contact a remote repository and send it commits that are in our database but not theirs.

`git pull`



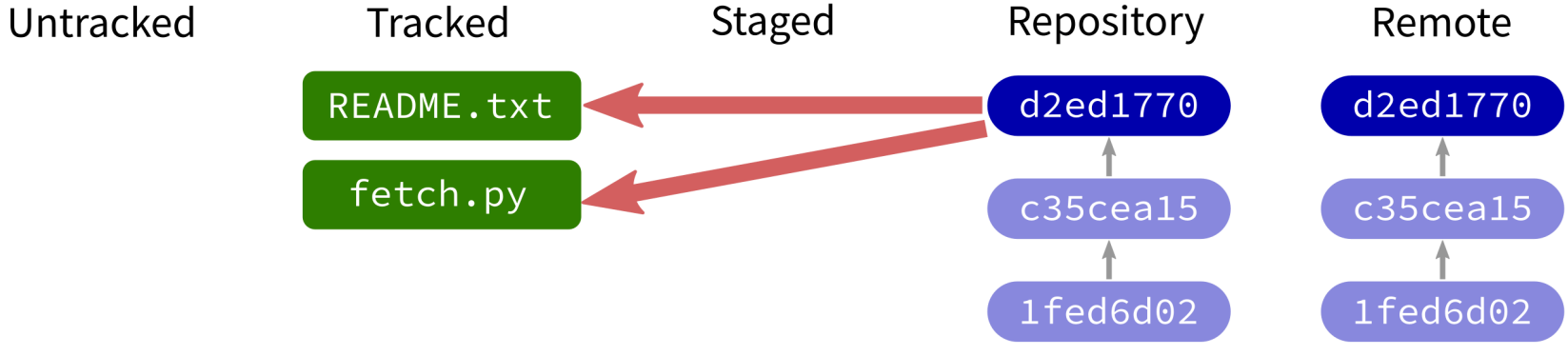
Contact a remote repository and get commits from its database that are not yet in ours.

`git pull`



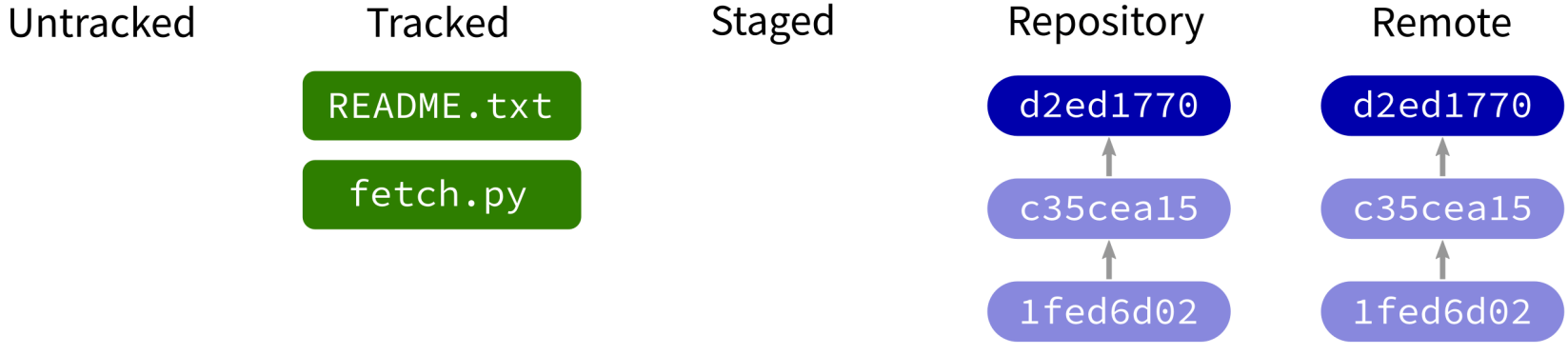
Contact a remote repository and get commits from its database that are not yet in ours.

`git pull`



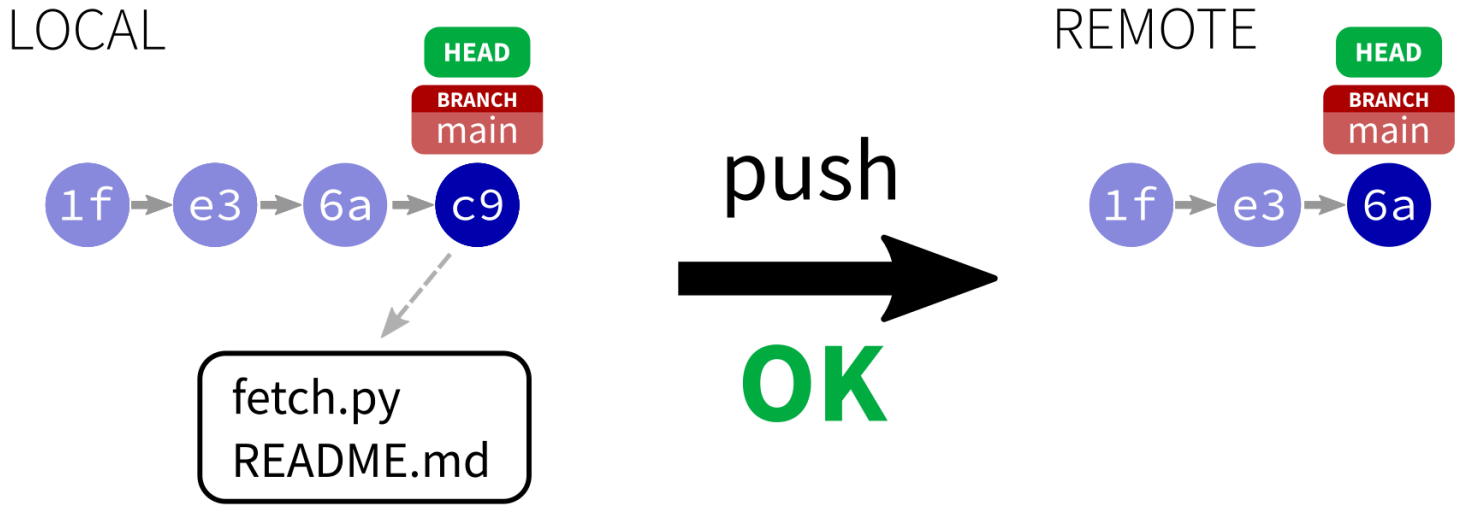
Contact a remote repository and get commits from its database that are not yet in ours.

`git pull`



Contact a remote repository and get commits from its database that are not yet in ours.

WHEN CAN YOU PUSH/PULL?

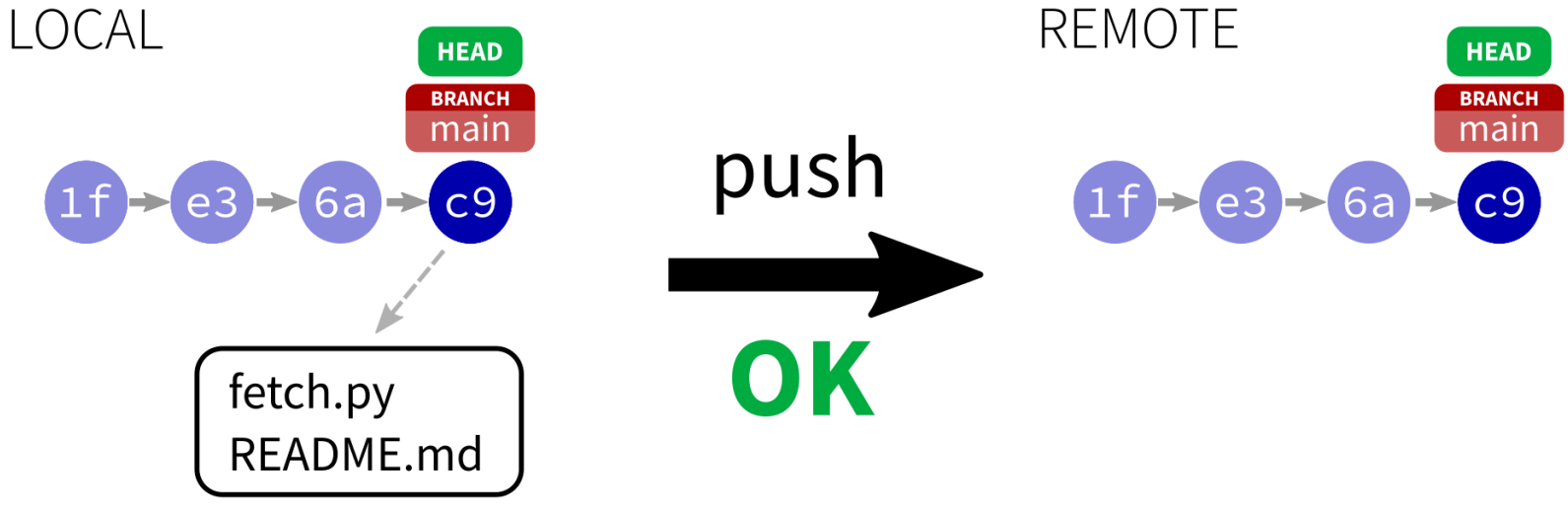


The local and remote repos have a commit labeled HEAD, the "latest".

You can **push** only if the local HEAD derives from the remote HEAD.

You can always **pull**, but it may trigger a merge.

WHEN CAN YOU PUSH/PULL?

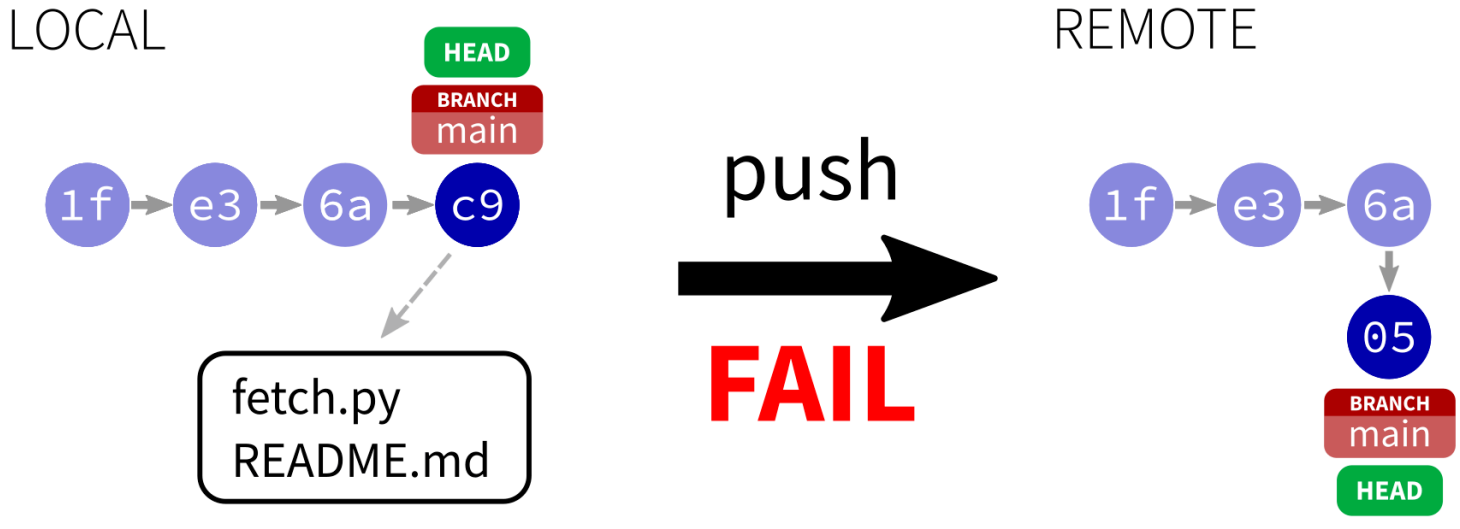


The local and remote repos have a commit labeled HEAD, the "latest".

You can **push** only if the local HEAD derives from the remote HEAD.

You can always **pull**, but it may trigger a merge.

WHEN CAN YOU PUSH/PULL?

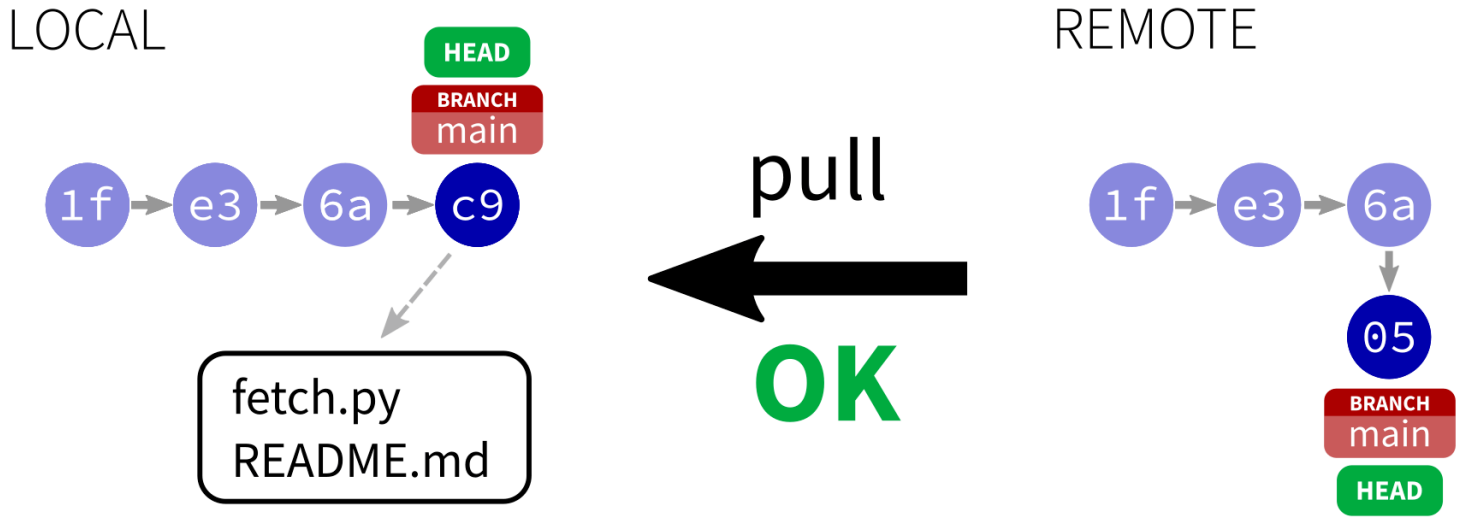


The local and remote repos have a commit labeled HEAD, the "latest".

You can **push** only if the local HEAD derives from the remote HEAD.

You can always **pull**, but it may trigger a merge.

WHEN CAN YOU PUSH/PULL?

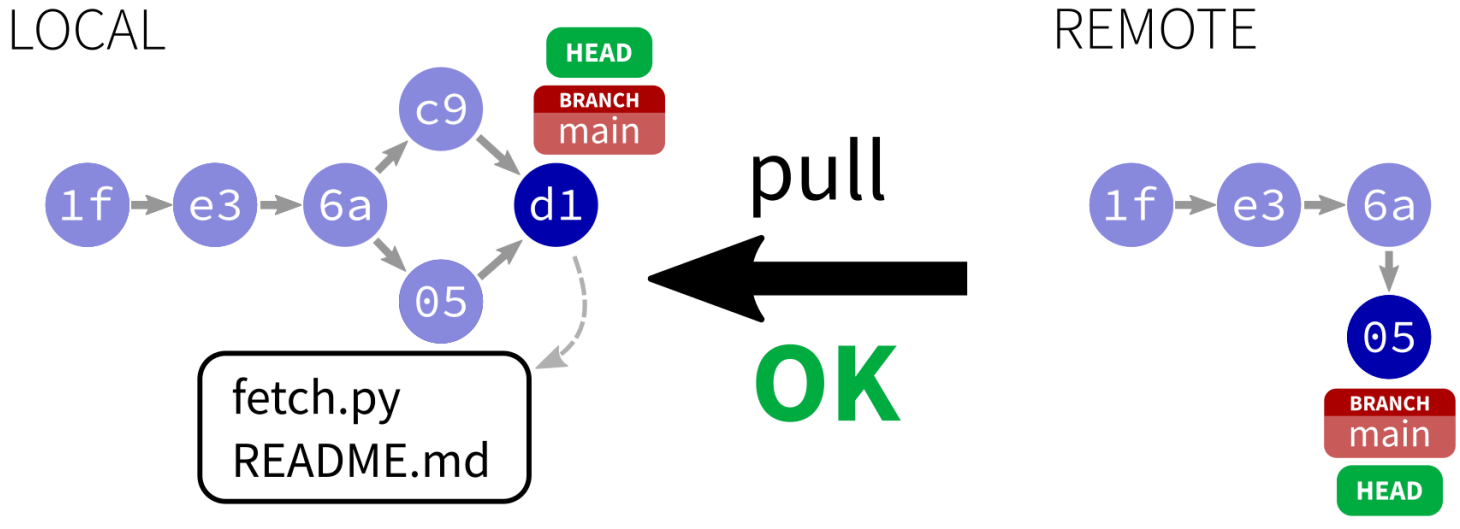


The local and remote repos have a commit labeled HEAD, the "latest".

You can **push** only if the local HEAD derives from the remote HEAD.

You can always **pull**, but it may trigger a merge.

WHEN CAN YOU PUSH/PULL?



The local and remote repos have a commit labeled HEAD, the "latest".

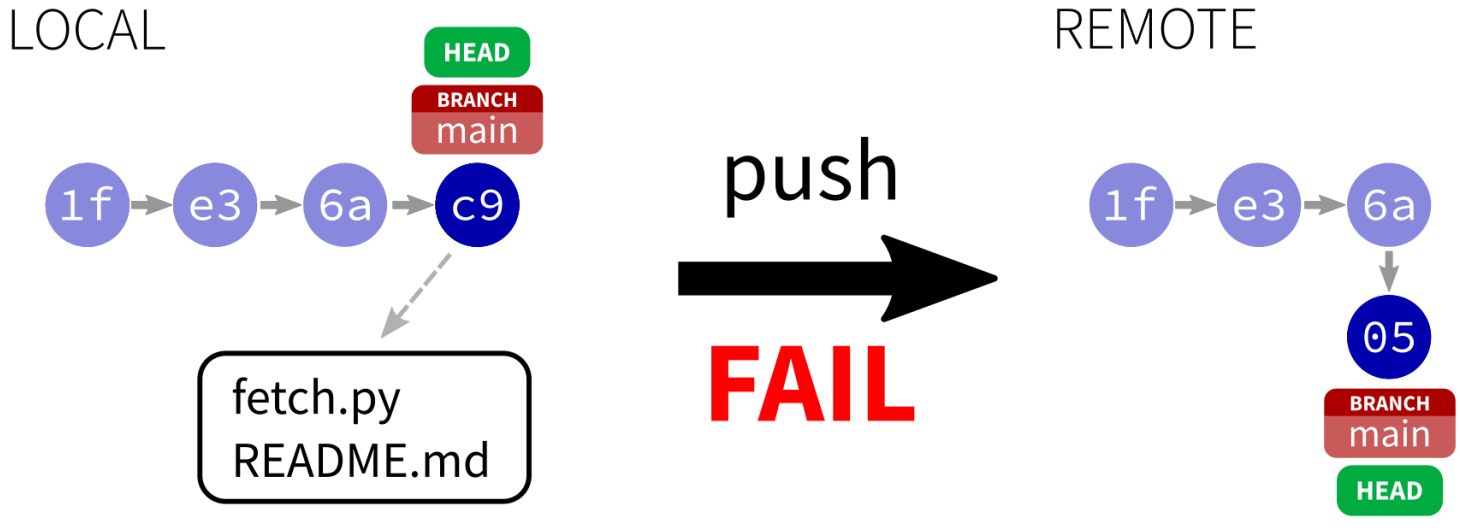
You can **push** only if the local HEAD derives from the remote HEAD.

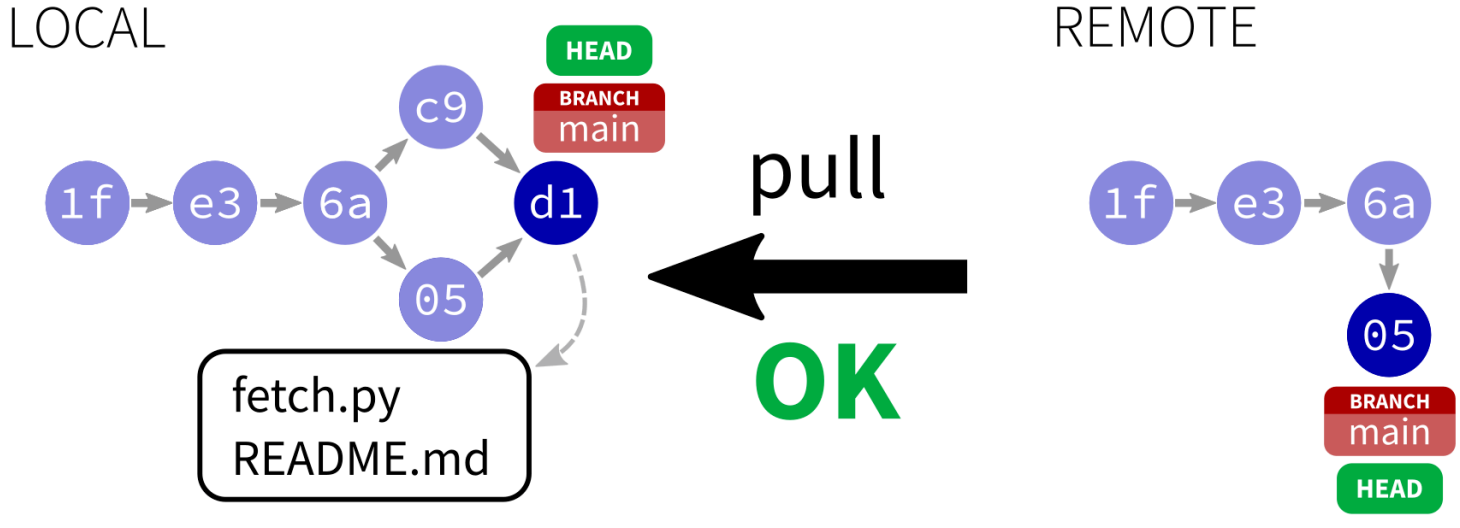
You can always **pull**, but it may trigger a merge.

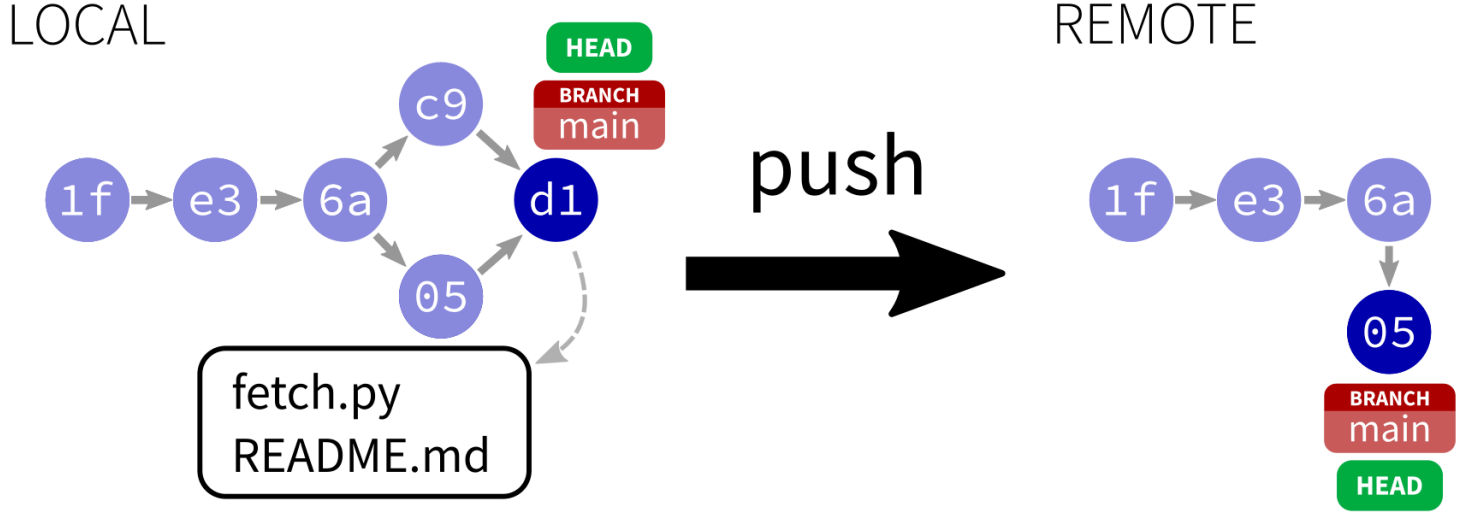
FAILED PUSH

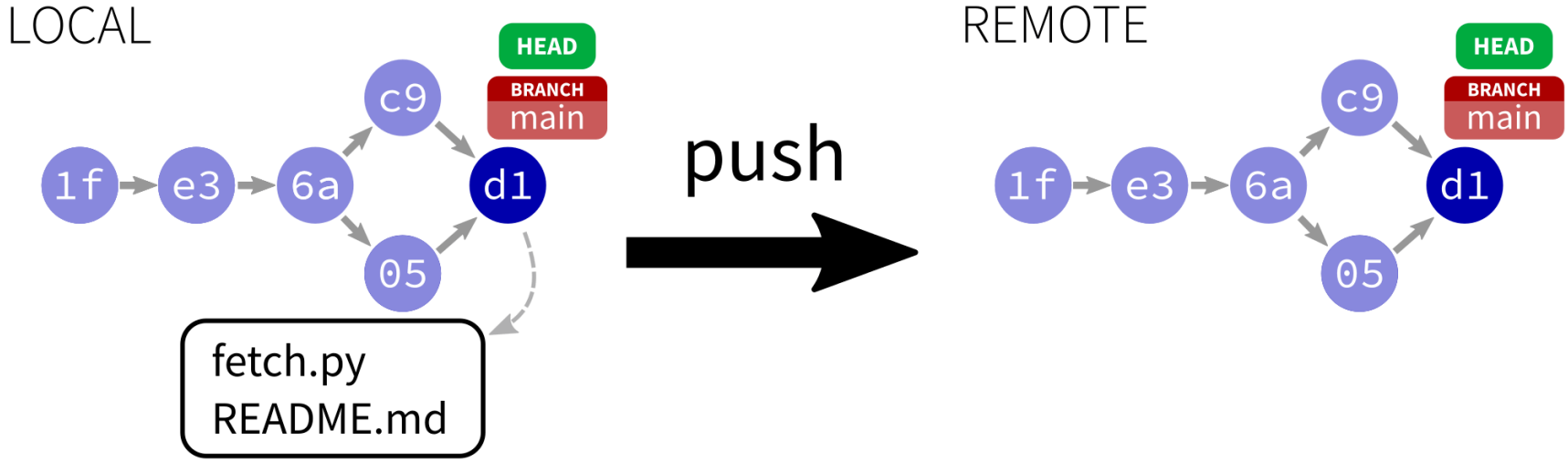
If a `git push` fails, the solution is to:

- `git pull` – starts a merge, often completes automatically
- `git push` – to send the unified updates to the remote


LOCAL
HEAD
BRANCH
main
1f
e3
6a
c9
fetch.py
README.md
push
FAIL
REMOTE
1f
e3
6a
05
BRANCH
main
HEAD


LOCAL
HEAD
BRANCH
main
c9
1f
e3
6a
d1
05
fetch.py
README.md
pull
OK
REMOTE
1f
e3
6a
05
BRANCH
main
HEAD


LOCAL
HEAD
BRANCH
main
c9
1f
e3
6a
d1
05
fetch.py
README.md
push
REMOTE
1f
e3
6a
05
BRANCH
main
HEAD


LOCAL
HEAD
BRANCH
main
c9
1f
e3
6a
d1
05
fetch.py
README.md
push
REMOTE
HEAD
BRANCH
main
c9
1f
e3
6a
d1
05

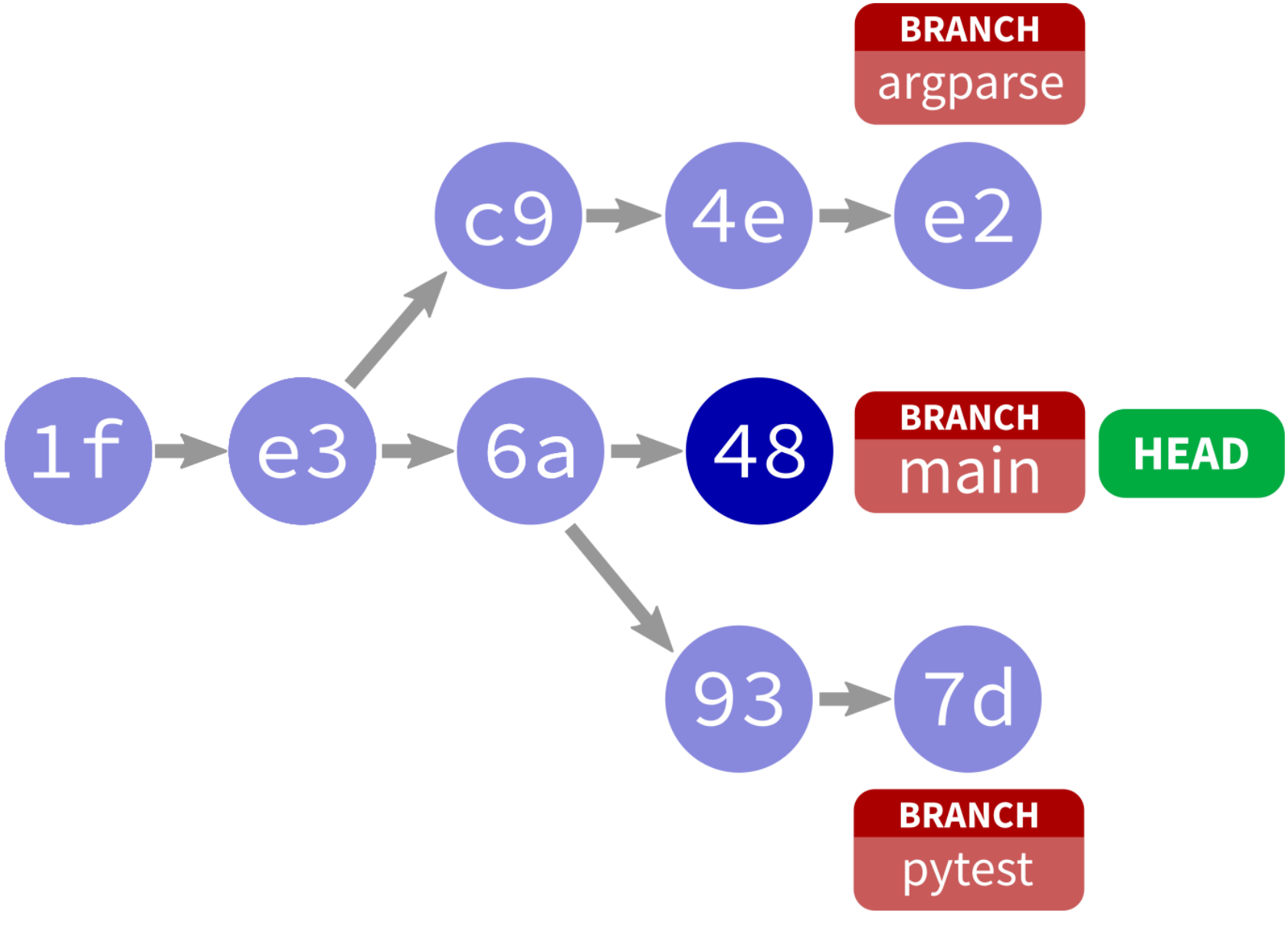
BRANCHES

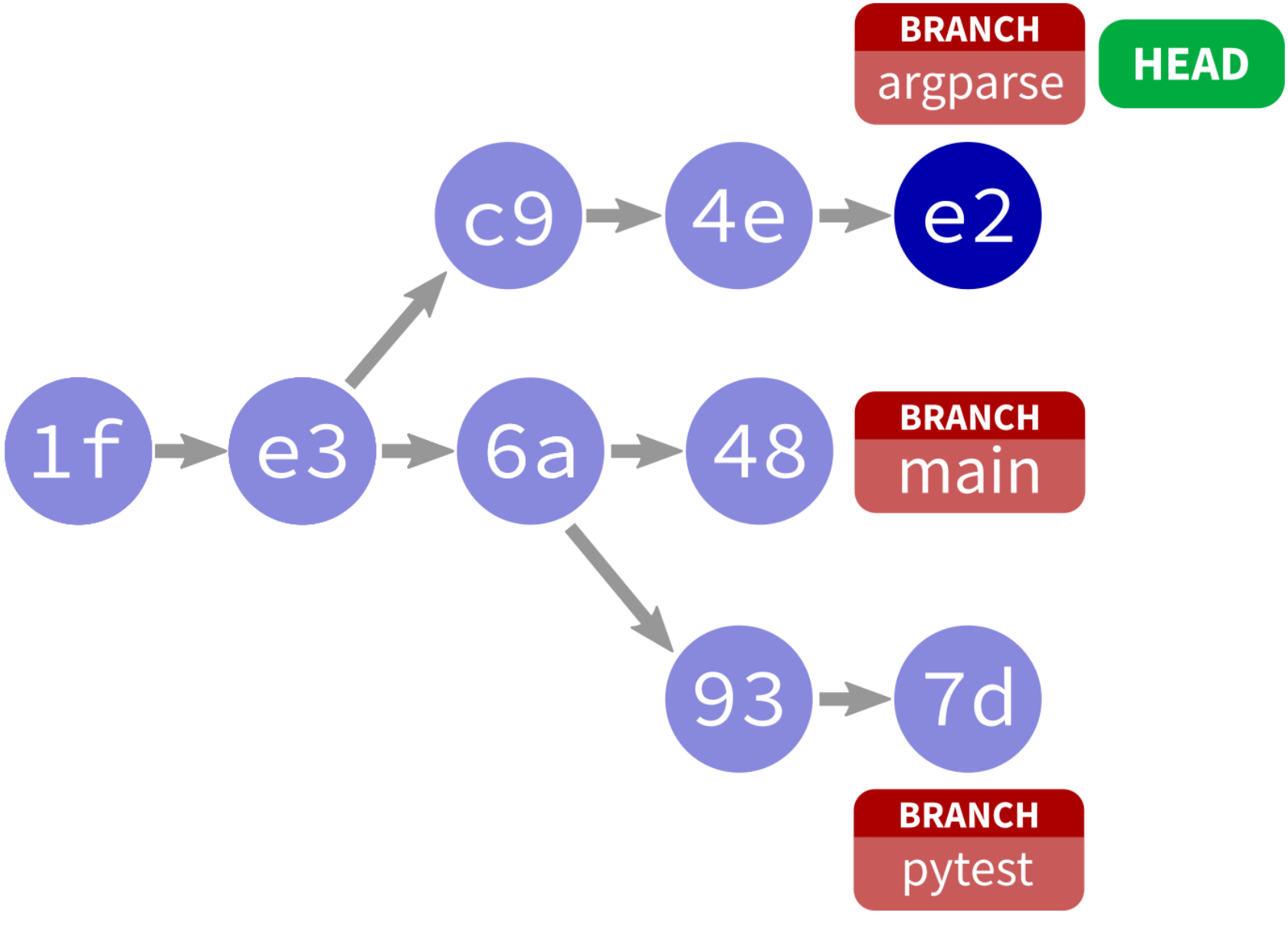
You can have multiple lines of development underway at once, each with their own name and history. You can switch between them at will.

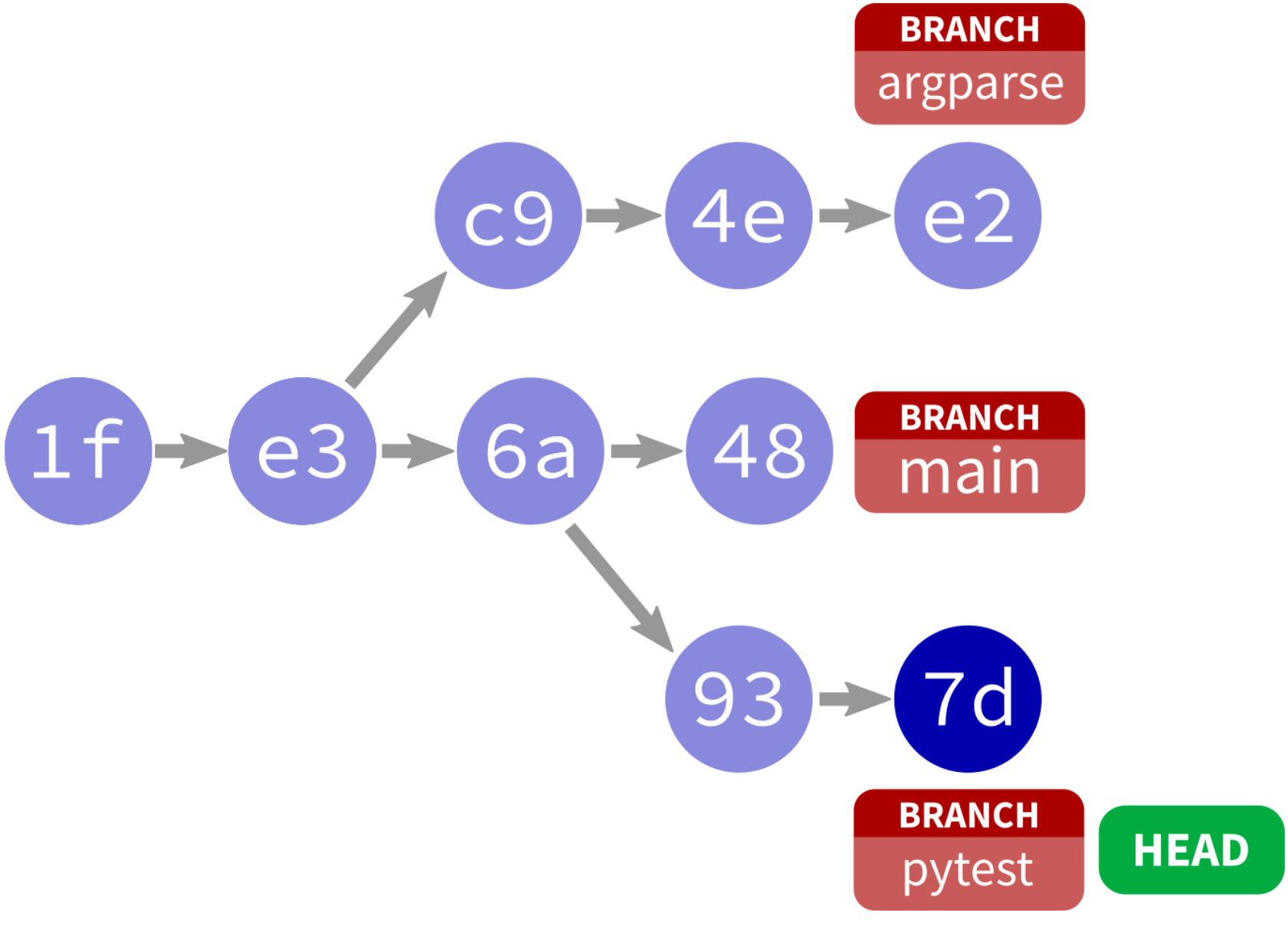
A repo starts with one branch called `main`.

Each branch is just a pointer to its latest commit.

Branches can be local or shared.


BRANCH
argparse
c9
4e
e2
1f
e3
6a
48
BRANCH
main
HEAD
93
7d
BRANCH
pytest


BRANCH
argparse
HEAD
c9
4e
e2
1f
e3
6a
48
BRANCH
main
93
7d
BRANCH
pytest


BRANCH
argparse
c9
4e
e2
1f
e3
6a
48
BRANCH
main
93
7d
BRANCH
pytest
HEAD

WORKING WITH BRANCHES

- **`git checkout -b my_branch_name`** – Create new branch
- **`git checkout my_branch_name`** – Switch to existing branch
- **`git checkout main`** – Switch to the main branch
- **`git branch -d my_branch_name`** – Delete a branch
- **`git commit`** – commit staged stuff to active branch
- **`git push origin my_branch_name`** – push a branch to remote (default is to keep them local)

WHY BRANCHES?

Often projects keep `main` clean, and do all work on changes in branches.

One branch per feature (or task) is common.

This way, features can be worked on in parallel.

END OF A BRANCH

Branches "end" in two ways:

- **Abandon**: switch away, never go back (maybe delete)
- **Merge**: bring changes from one branch into another (often into main)

MERGING

To merge `alpha` into `beta`, ending `alpha`:

- `git checkout beta`
- `git merge alpha`
- Resolve any conflicts identified.

The merge is a commit to `beta`.

LOOSE ENDS

`git blame`

`git blame FILE` shows commit that most recently changed each line of a file.

If you know where a problem is, this helps figure out how it got there.

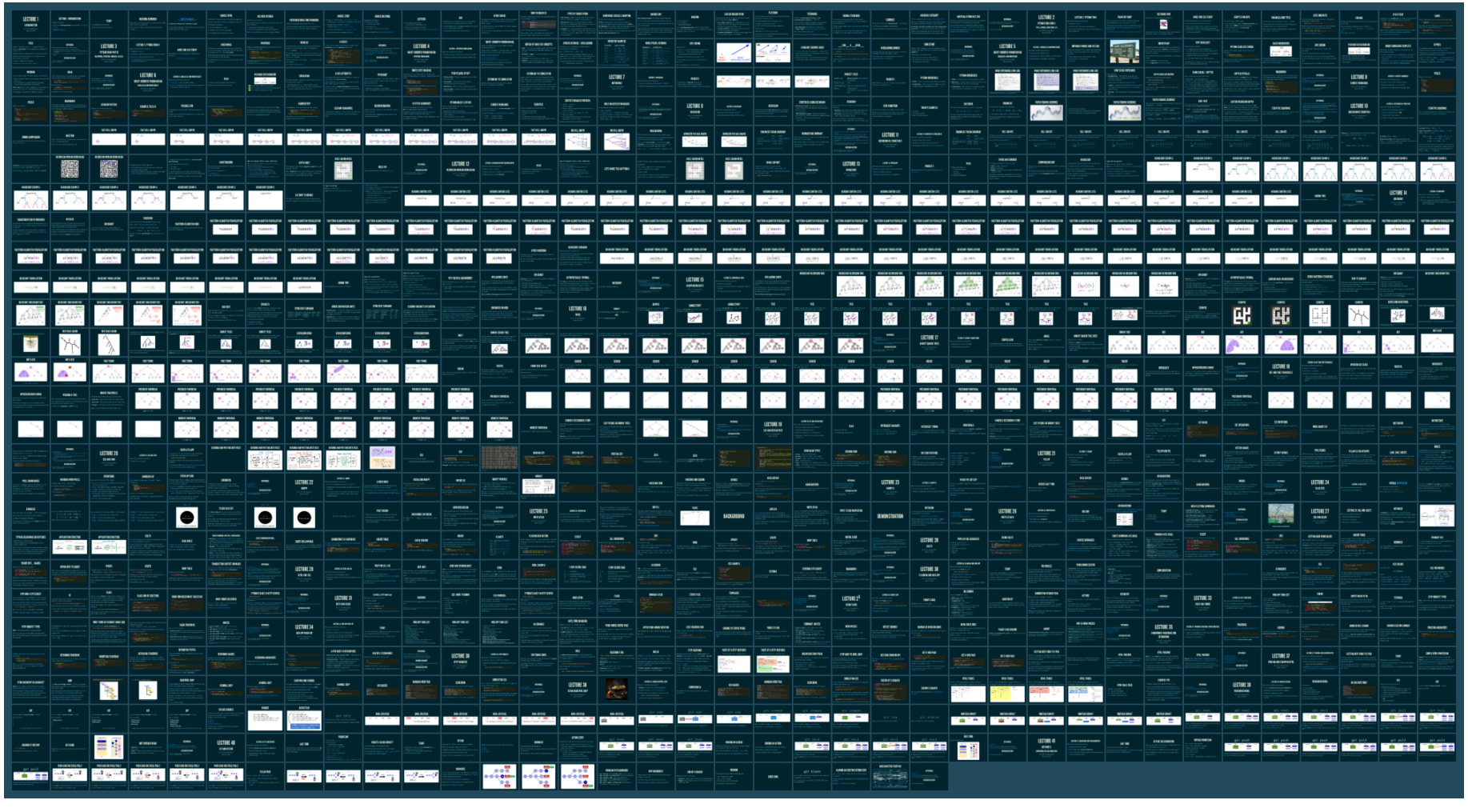
CLONING AN EXISTING GITHUB REPO

`git clone URL` – make a local copy of an existing repository.

Works with local repositories, too. Substitute directory for URL.

Like `git init`, this is a rare event.

OUR SEMESTER TOGETHER



REFERENCES

- git home page
- Official git documentation (includes tutorial videos, Pro Git book)
- git - the simple guide (nice practical introduction, with mild profanity)

REVISION HISTORY

- 2023-04-28 Initial publication.