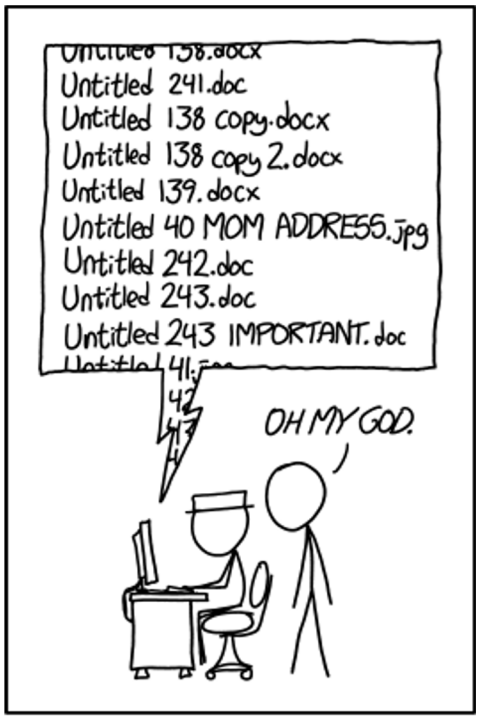
Version Control and Git

Lecture 1

Slides created by Josh Ervin and Hunter Schafer.
Based off slides made by Marty Stepp, Jessica Miller, Ruth Anderson, Brett Wortzman, and Zorah Fung
and Dave Matuszek

VERSION CONTROL - INDIVIDUAL

- Does any of the following sound familiar?
  - Your code was working great! Then you made a few changes and now everything is broken and you saved over the previous version?
  - You accidently delete a critical file and can't get it back.
  - Your computer broken or was stolen and now all of your files are gone!
  - While writing a paper for one of your classes you save each version as final_paper.doc, final_paper2.doc, final_paper_actually_this_time.doc, UGH.doc
- There has to be a better way to manage versions...



VERSION CONTROL - TEAMS

- Does any of the following sound familiar?
  - My partner and I are paired up for a project for one of our CS classes. We usually pair program together in the labs but sometimes we have to work remotely. Who keeps the most up-to-date version of the project? How do we share changes with each other? What if I want to compare the changes my partner made?
  - How do we keep backups of important files? Who stores them on their computer?

VERSION CONTROL

- Version Control: Software that keeps track of changes to a set of files.
- You likely use version control all the time:
  - In Microsoft Word, you might use Ctrl+Z to undo changes and go back to an earlier version of the document.
  - In Google Docs you can see who made what changes to a file.
- Many people have a use-case for version control
  - We often think of version control as related to managing code bases, but it's also used by other industries such as law firms when keeping track of document changes over time.

VERSION CONTROL Systems

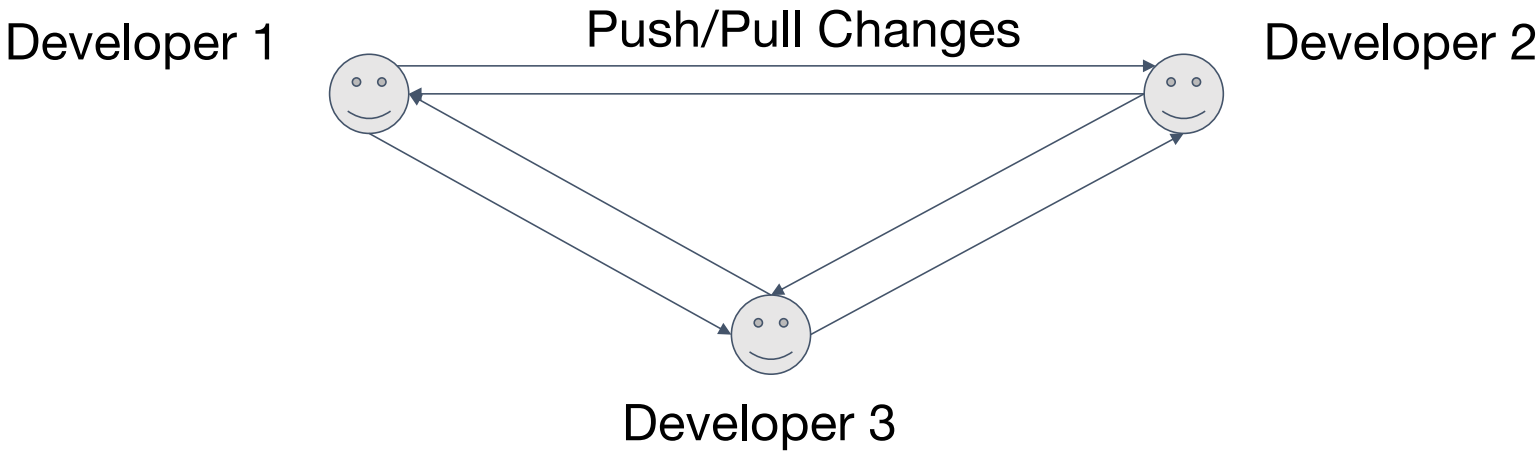
- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion (Git has most market share)

REPOSITORY

- A repository, commonly referred to as a *repo* is a location that stores a copy of all files.
  - The **working directory** *(or working tree)* is different from the **repository** (see next slide)
- What should be inside of a repository?
  - Source code files (i.e. .c files, .java files, etc)
  - Build files (Makefiles, build.xml)
  - Images, general resources files
- What should **not** be inside of a repository (generally)
  - Object files (i.e. .class files, .o files)
  - Executables

REPOSITORY

- With git, everyone working on the project has a complete version of the repository.
  - There is a **remote** repository, which is the defacto central repository
  - Remote repositories are hosted on services like GitHub or Gitlab
  - Everyone has a local copy of the repository, which is what we use to commit.



GIT: FOUR PHASES

NOTE: There are way more git commands than what is listed here - this is a simplified model to get us started.

INSPECTING A REPOSITORY

git status

Working directory

Staging Area

git log

Commit History

Itempool

Suppose a run of git status show the following:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt
```

Based on this information, at what location are the changes to file1.txt and file2.txt within the git phases on the previous slides.

A. Working directory
B. Index (Staged)
C. Local Repository
D. Remote Repository

Itempool

Suppose a run of git status show the following:

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified: file1.txt
        modified: file2.txt

We want to run a sequence of commands that includes the changes to file1.txt in a single commit followed by the changes to file2.txt in another commit in the **local repository**. What commands should we run?

GIT COMMANDS

git clone ***url [dir]***	Copy a git repository
git add ***files***	Adds file contents to staging area
git stage ***files***	Same as git add ***files***
git commit	Takes a snapshot of staging area and creates a commit
git status	View status of files in working directory and staging area
git diff	Show difference between staging area and working directory
git log	Show commit history
git pull	Fetch from remote repository and try to merge
git push	Push local repository to remove repository

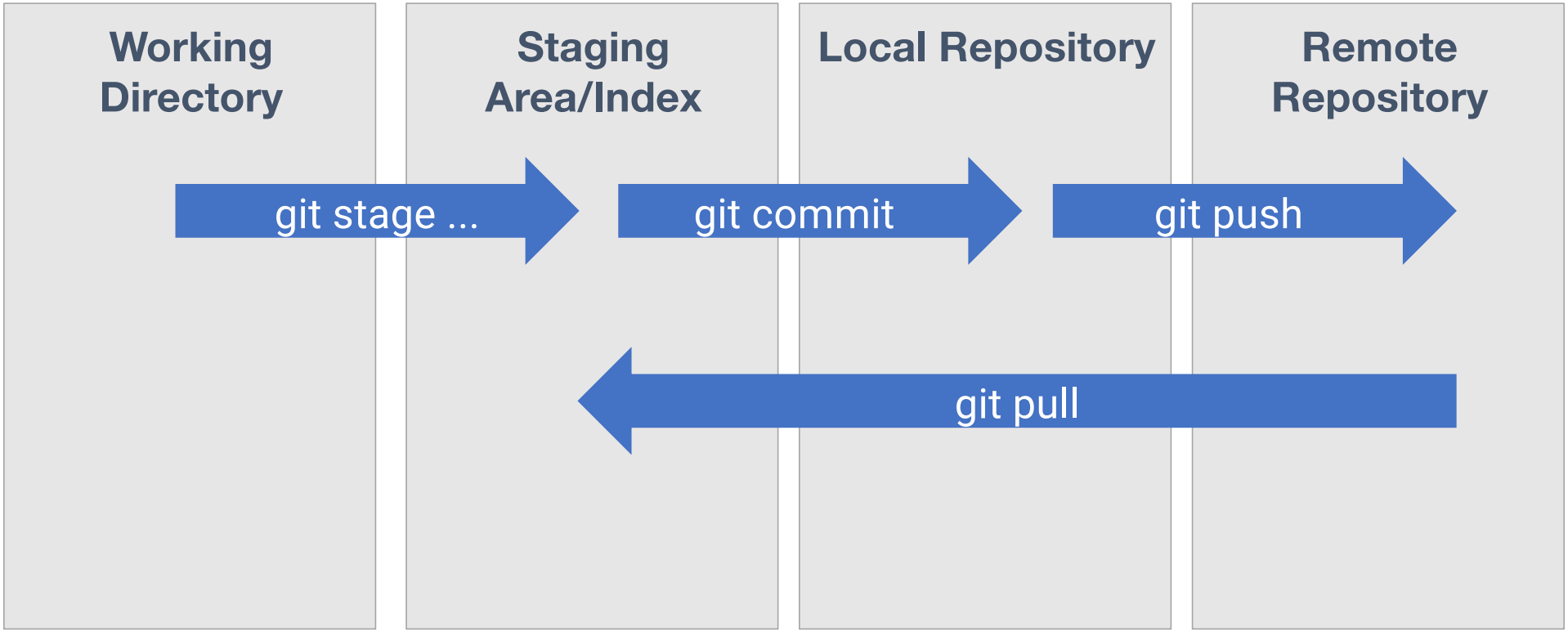
ADDING AND COMMITTING FILES

- The first time we asked a file to be tracked, and every time before we commit a file we must add it to the staging area. This can be done with the following command
  - $ git add file1.txt file2.txt
- This takes a *snapshot* of the files and adds it to the staging area. You can still modify files in the working directory, but you will need to add again to have these changes saved in the staging area.
- *Note: To unstage a change, you can use the following*
  - $ git reset HEAD filename
- To move staged changes into the local repository we commit them from the staging area
  - $ git commit -m “Updated filename”
- *Note: All of these commands are acting on the local copy of your repository. You will need to push them to remote to see these changes elsewhere.*

COMMIT MESSAGES

- Running `$ git commit` will bring up a text editor by default for you to type your commit message.
  - Write the subject text (less than 50 characters) on the first line, followed by paragraphs describing your commit in the rest of the page.
- Running `$ git commit -m "Your Message"` will allow you to type the message directly in the command line without opening a text editor.
  - This is useful for simple commits, but avoid using it for more complicated commits.
- Regardless, the subject line should always be written with the following form
  - *If applied, this commit will* **your subject line here**

GIT: FOUR PHASES WITH REMOTE



CS 2820 GITLAB

- For this class we will be using gitlab. To access:

1. Log onto CS 2820 Gitlab (https://research-git.uiowa.edu)
   a. Use your Hawk ID: sLastName
2. If you want:
   a. Add an ssh key (https://gitlab.cs.washington.edu/help/ssh/README.md)
   b. Follow the instructions in README.md which say to type
      i. $ ssh-keygen -t rsa -C "yourHawkID@uiowa.edu" -b 4096
      ii. Hit return to accept the default file location. You do not need a password, so you may hit enter twice when prompted for a password
   c. Then type $ cat ~/.ssh/id_rsa.pub
   d. Copy and paste the key into the SSH-Keys section under Profile Settings in your user profile on Gitlab.
   e. For a brief introduction to SSH keys, see https://jumpcloud.com/blog/what-are-ssh-keys

Get Ready to Use Git

- Set the name and email for Git to use when you commit:
  $ git config --global user.name “Bugs Bunny”
  $ git config --global user.email bBunny@uiowa.edu
  - Only need to do this once
  - You can call git config --list to verify these are set.
  - These will be set globally for all Git projects you work with.
  - You can set variables on a project-only basis by not using the --global flag.
- If you want to use a different name/email address for a particular project, you can change it for just that project
  - cd to the project directory
  - Use the above commands, but leave out the --global
- The latest version of Git will also prompt you that push.default is not set, you can make this warning go away with:
  $ git config --global push.default simple
- You can also set the editor used for writing commit messages (default = vim):
  $ git config --global core.editor emacs

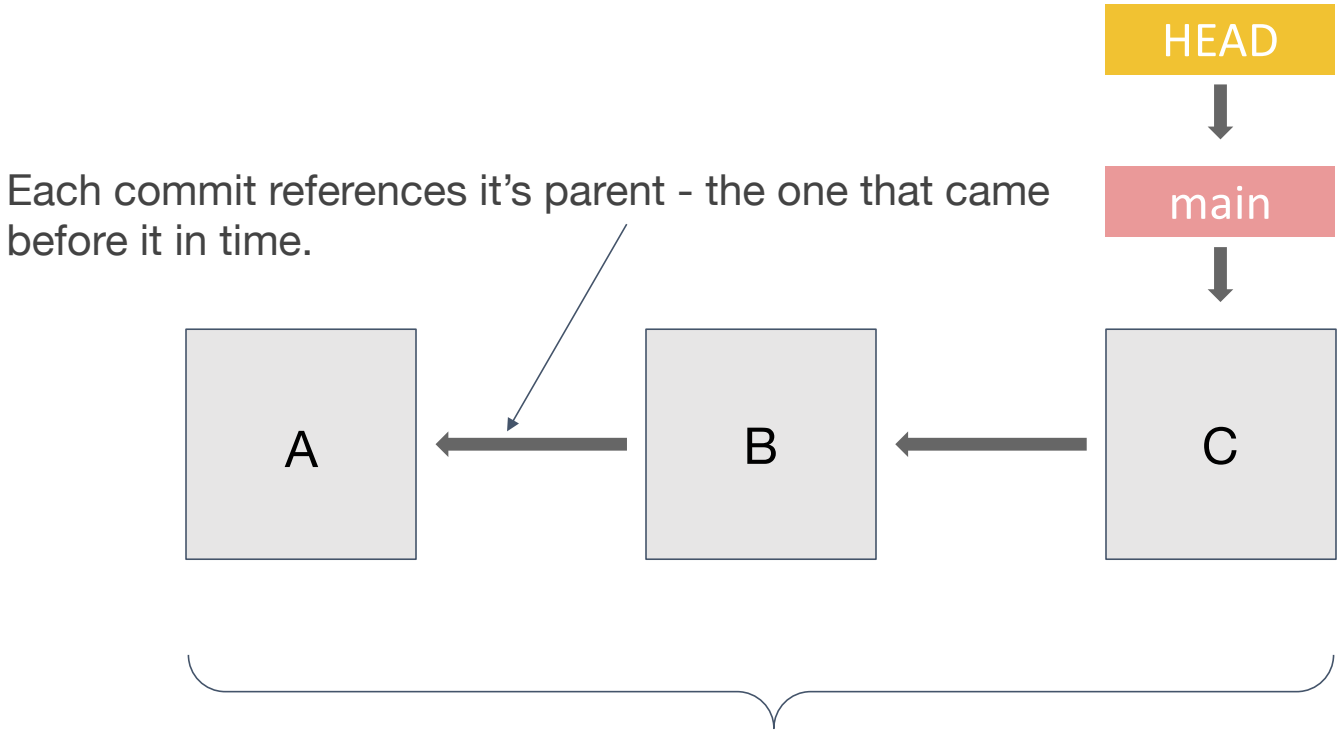
REVIEW: FOUR PHASES

Working Directory	Staging Area/Index	Local Repository	Remote Repository
Working changes	Changes you're preparing to commit	Local copy of the repository with your committed changes	Remote shared repository (Usually stored with a platform like GitHub/Gitlab)

GIT: BRANCHING

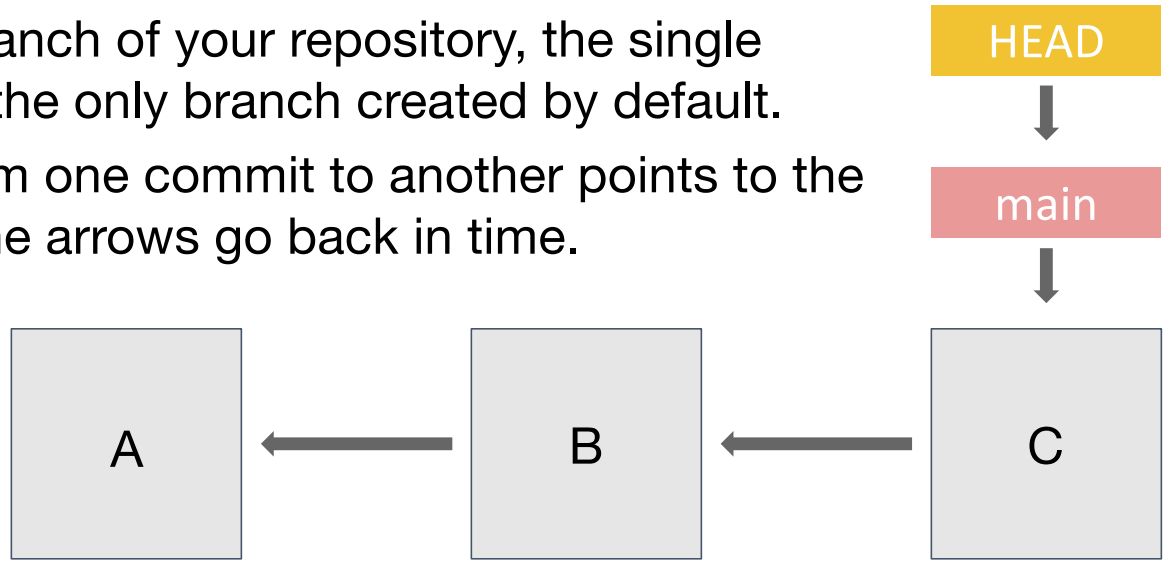
- So far, all the operations we've done have been on the main branch.
- However, it's very rare you'll be working on master directly. Instead, you'll work on a separate branch:
  - Master is the "single source of truth" - the history of the project
  - The code on master should be stable and compile
  - Because of this, it's difficult to share and collaborate on in progress work.
  - Working directly on master makes it difficult to work simultaneously on unrelated features.

COMMIT HISTORY



COMMIT HISTORY

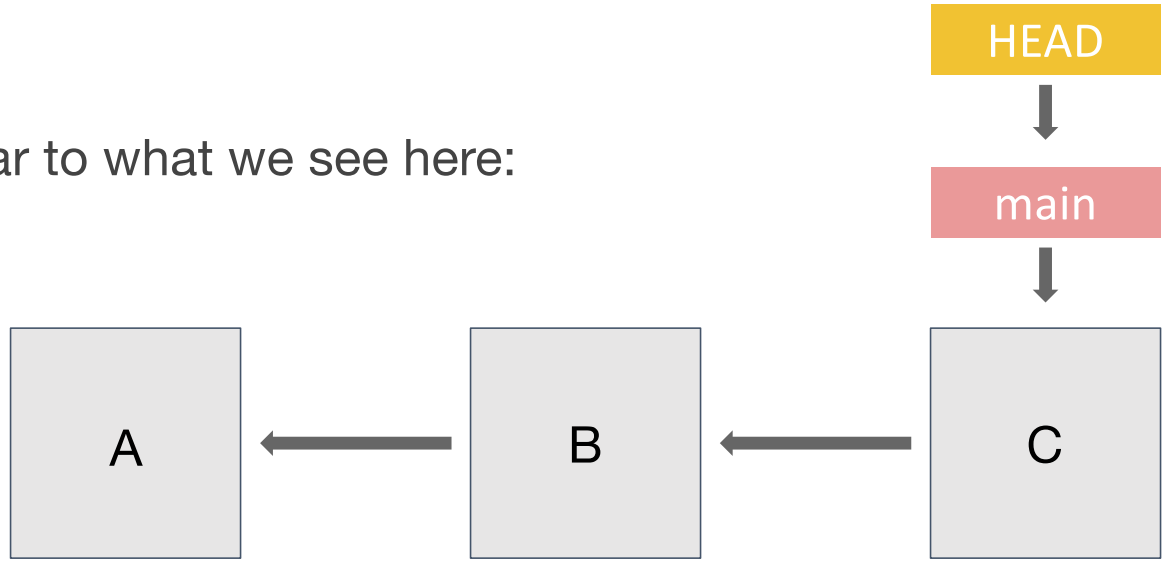
- HEAD: This is a pointer, or reference, to the git branch that you are currently working on.
- Main: This the main branch of your repository, the single “source of truth”, and the only branch created by default.
- **NOTE:** The pointer from one commit to another points to the previous commit, so the arrows go back in time.



COMMIT HISTORY

To view this same information on the command line:
$ git log

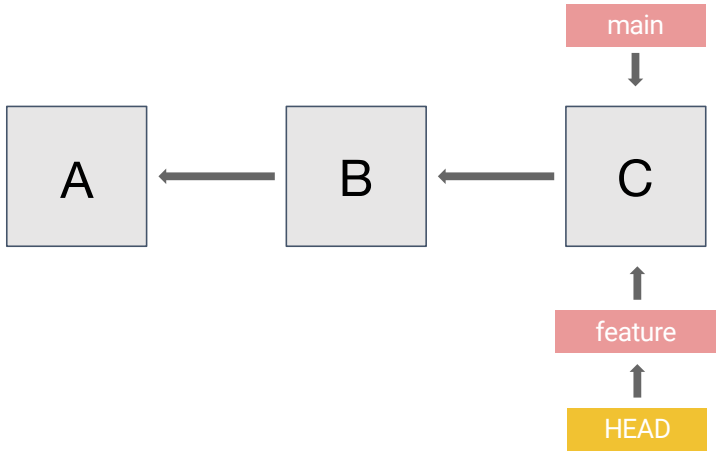
Or, to make it more similar to what we see here:
$ git log --graph --oneline



BRANCHING

Commands that have been run:

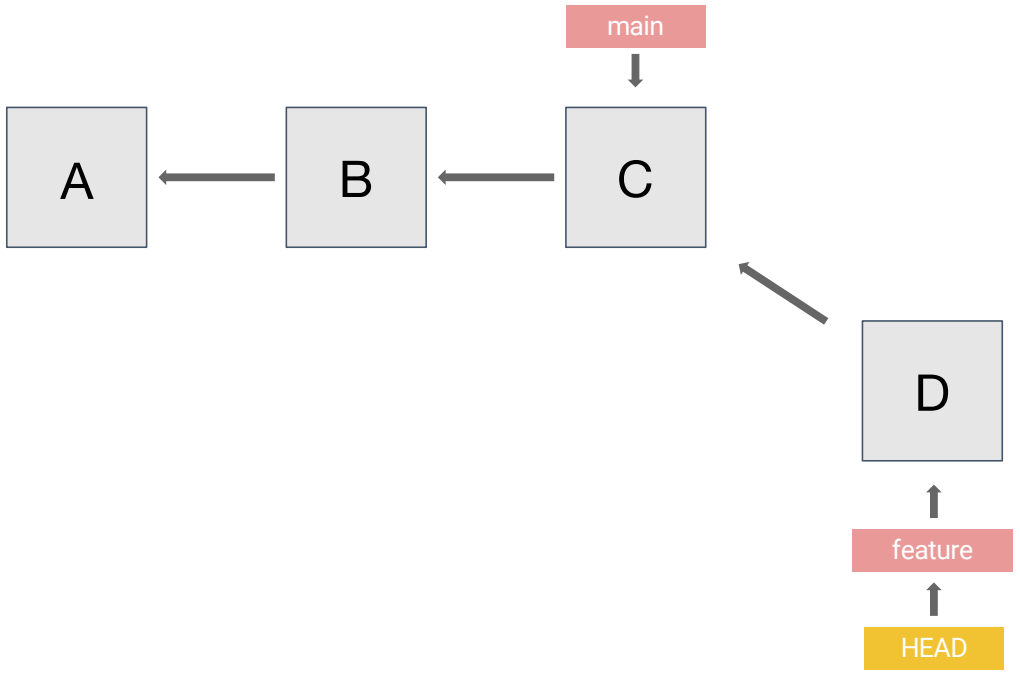
$ git branch feature
$ git checkout feature



BRANCHING

Commands that have been run:

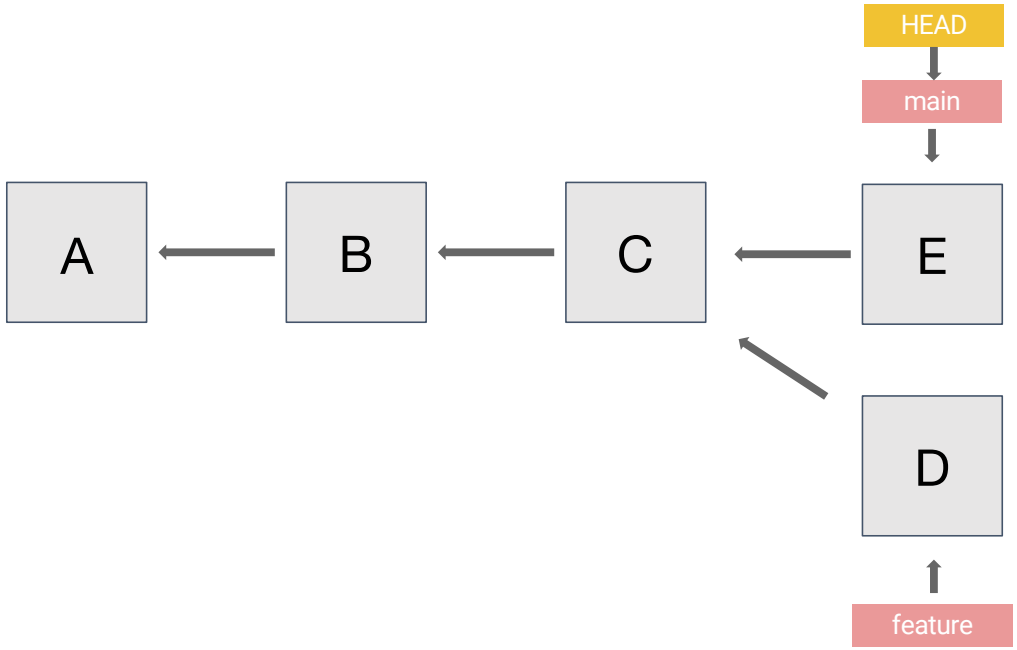
$ git branch feature
$ git checkout feature
$ echo "D" >> file.txt
$ git add file.txt
$ git commit -m "D"



BRANCHING

Commands that have been run:

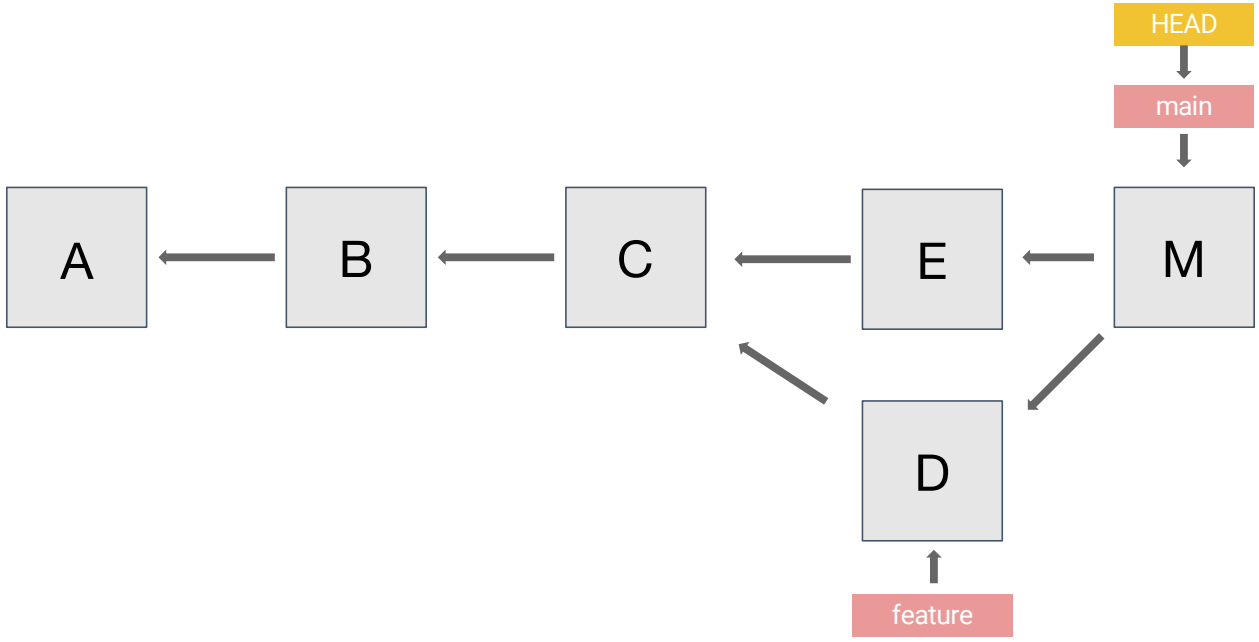
```
$ git branch feature
$ git checkout feature
$ echo "D" >> file.txt
$ git add file.txt
$ git commit -m "D"
$ git checkout main
$ echo "E" >> file.txt
$ git add file.txt
$ git commit -m "E"
```



BRANCHING

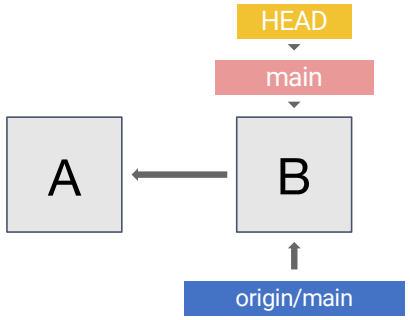
Commands that have been run:

$ git branch feature
$ git checkout feature
$ echo “D” >> file.txt
$ git add file.txt
$ git commit -m “D”
$ git checkout main
$ echo “E” >> file.txt
$ git add file.txt
$ git commit -m “E”
$ git merge feature

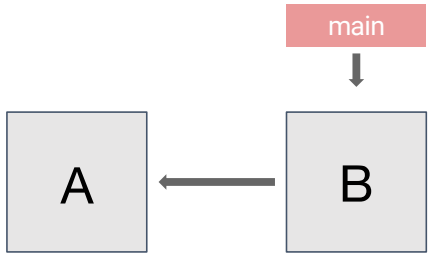


WORKING WITH REMOTE

Local

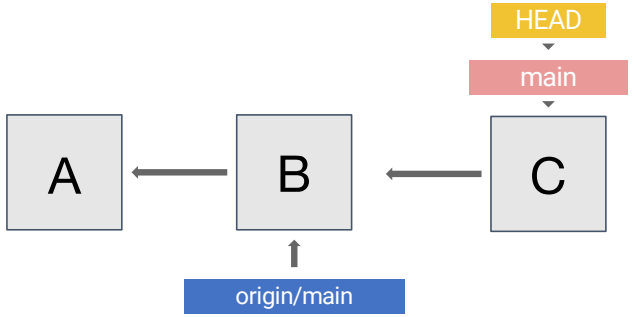


Remote

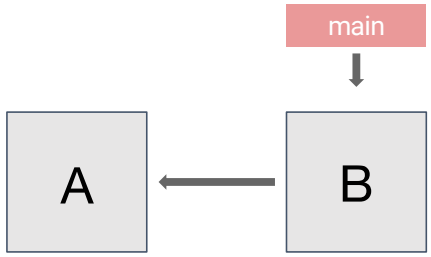


WORKING WITH REMOTE

Local

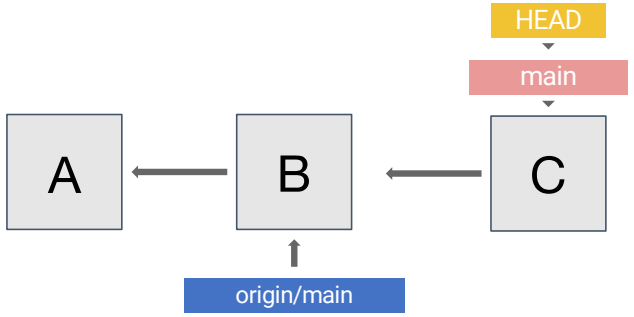


Remote

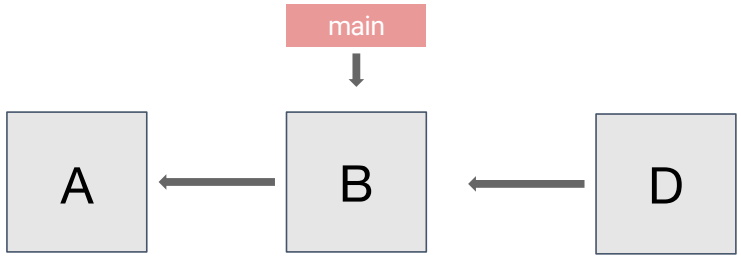


WORKING WITH REMOTE

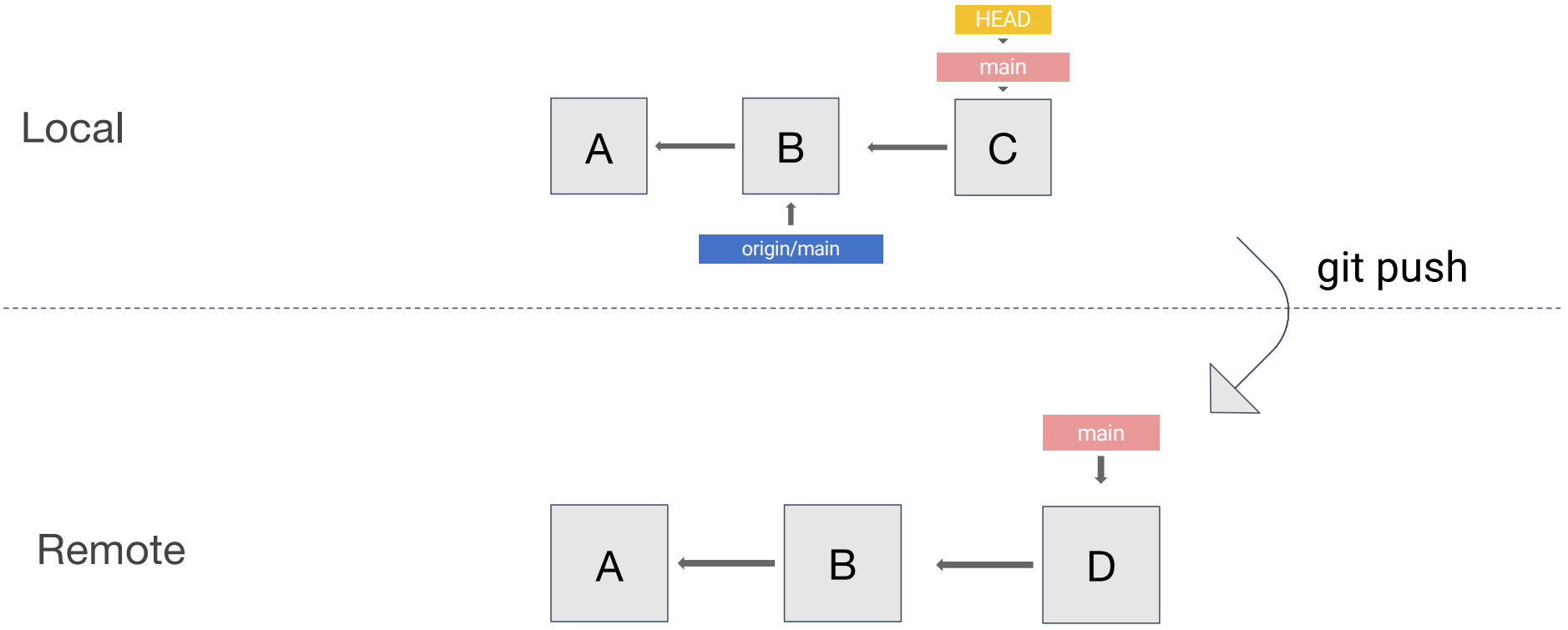
Local



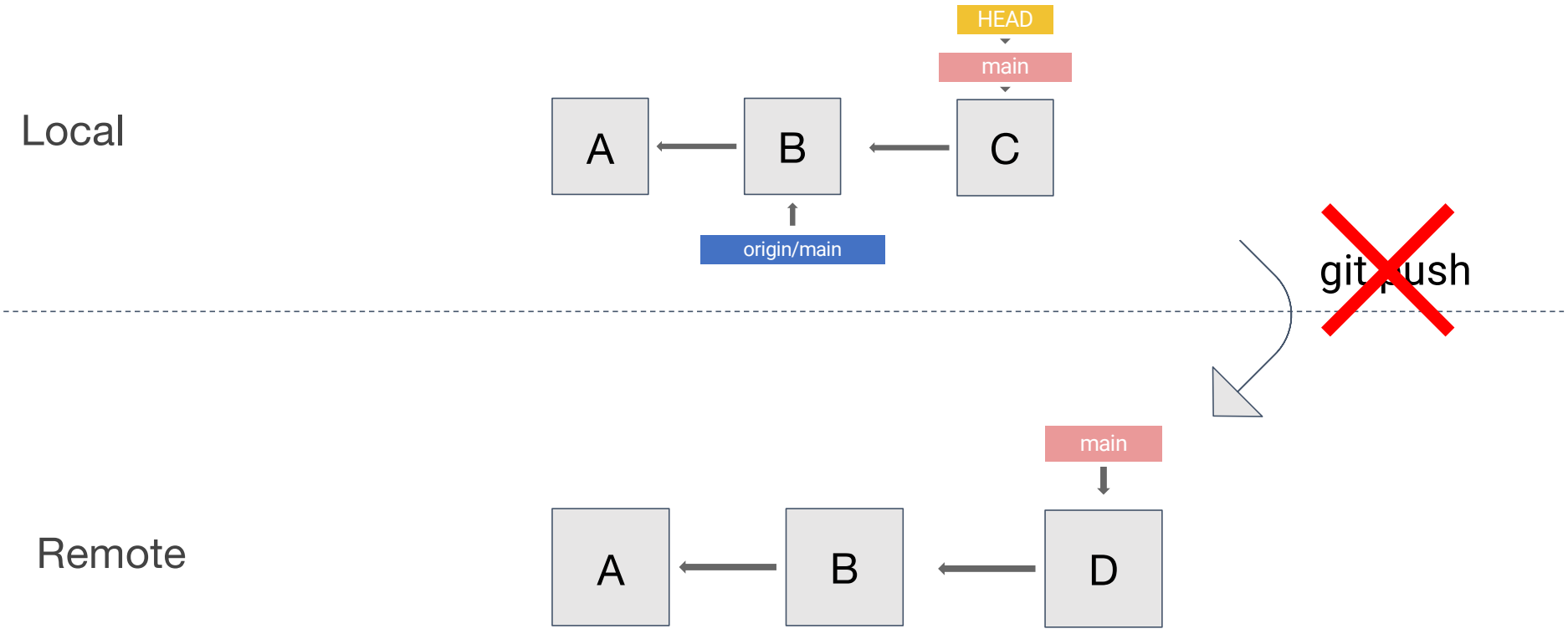
Remote



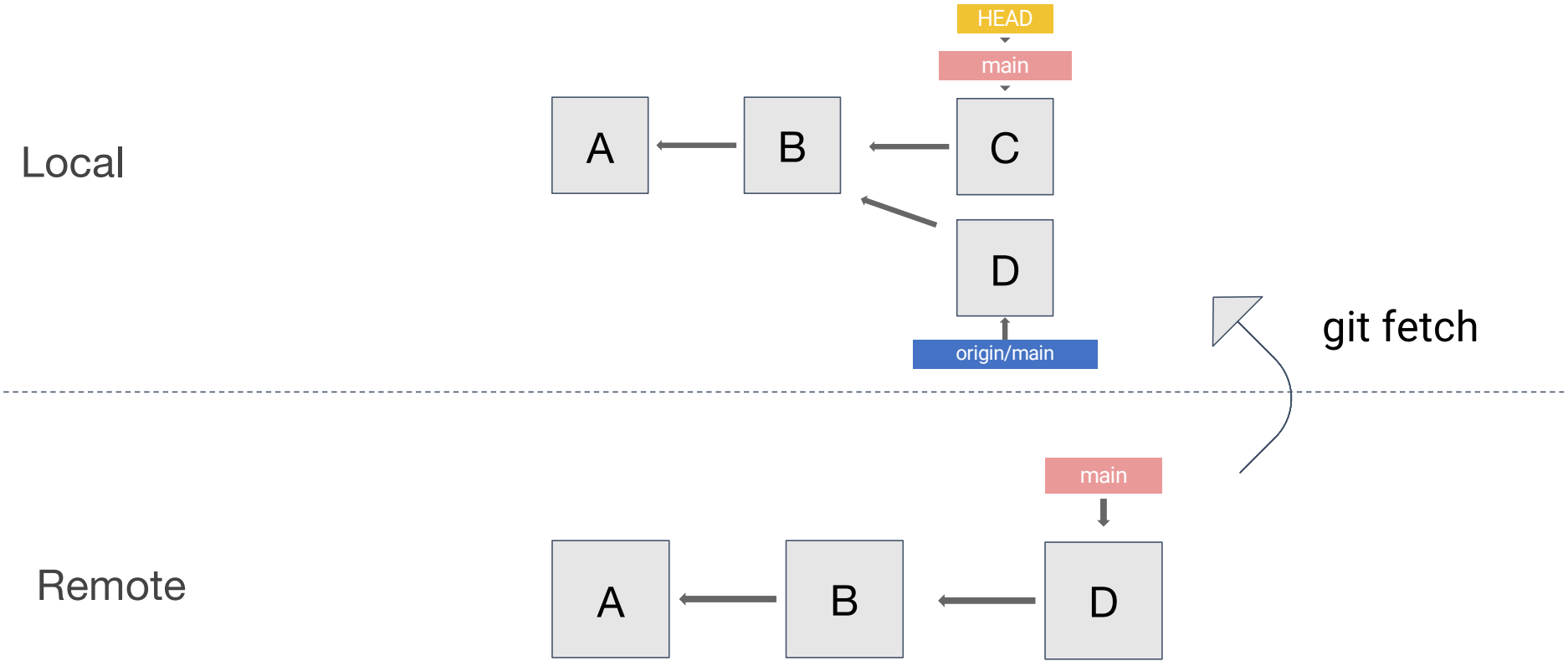
WORKING WITH REMOTE



WORKING WITH REMOTE

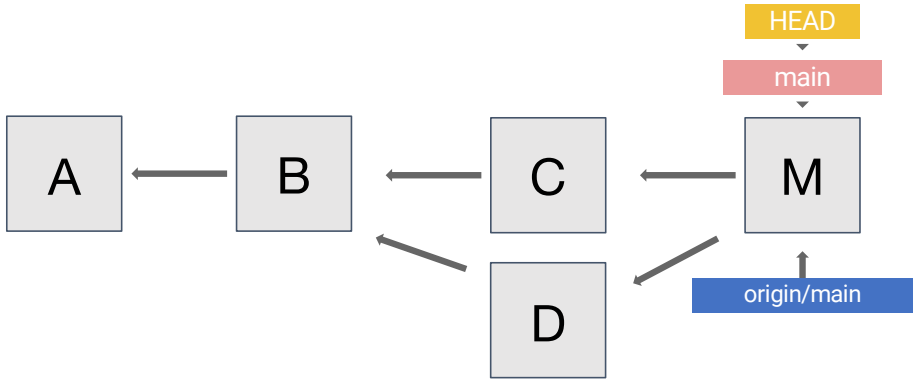


WORKING WITH REMOTE



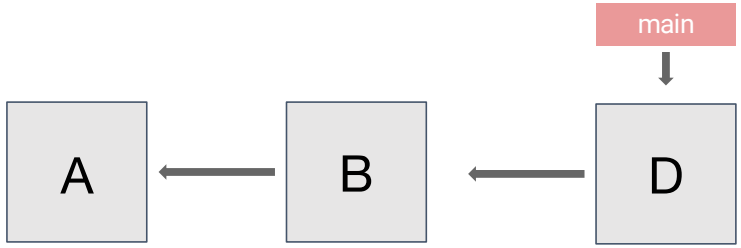
WORKING WITH REMOTE

Local



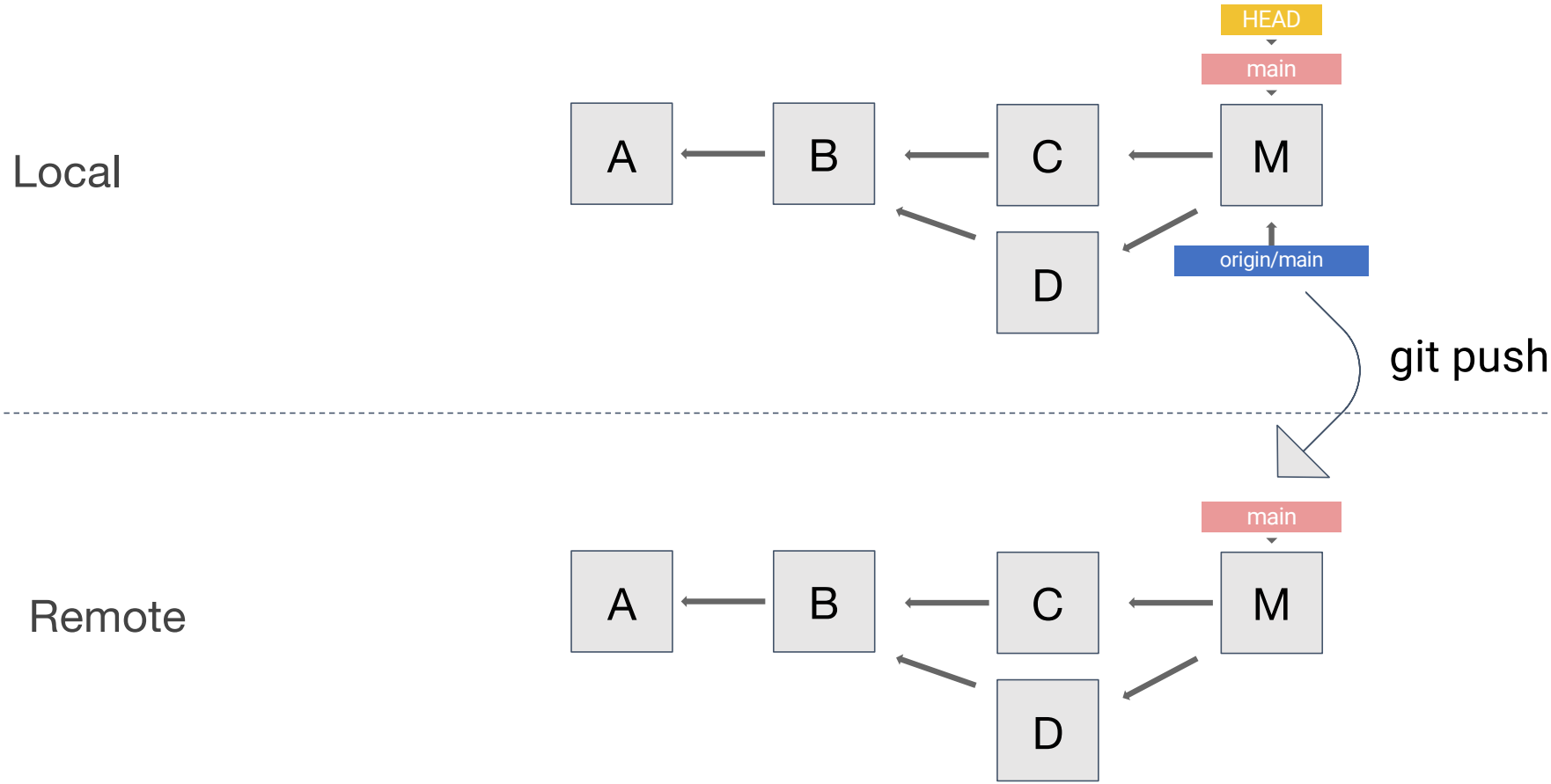
git merge

Remote



Note: It's more common to do git pull which is an alias for git fetch + get merge

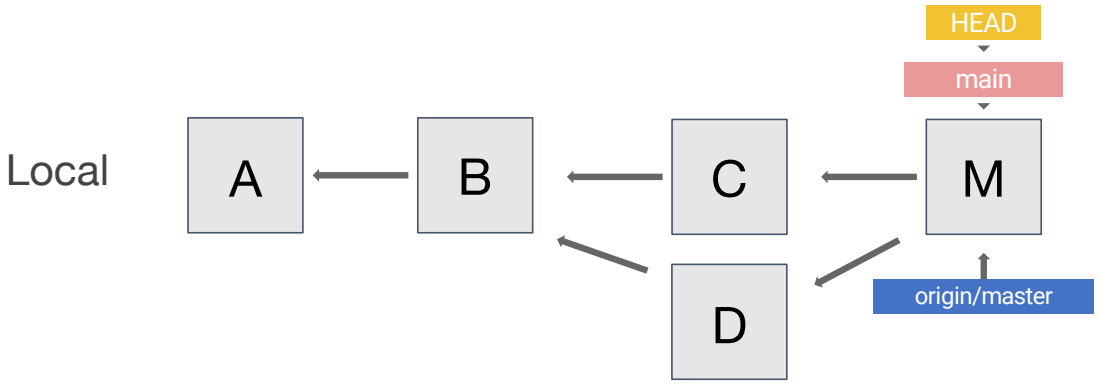
WORKING WITH REMOTE

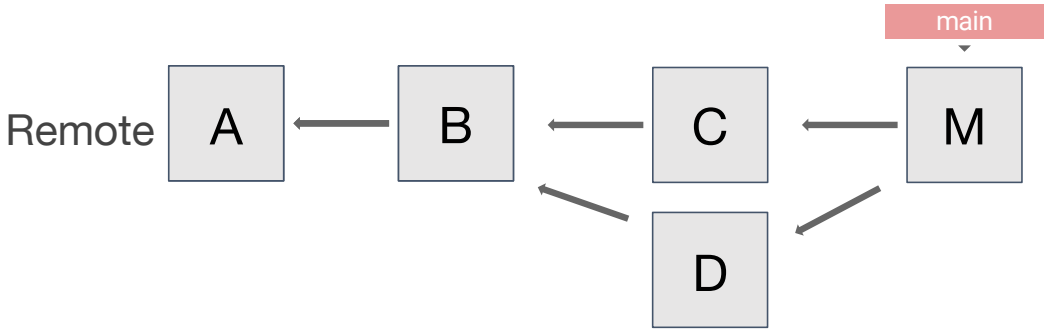


PULL REQUESTS

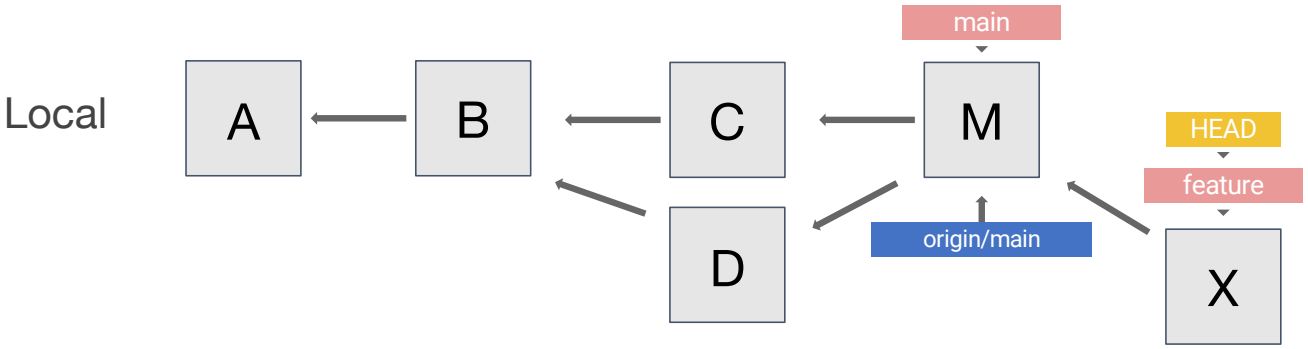
- In practice, it's very rare that we would merge branches locally and push them to remote.
- Instead, we use a service like GitHub or GitLab to help us:
  - Step 1: Create a local branch and make some commits
  - Step 2: Push those commits to remote
  - Step 3: Open a pull/merge request on GitLab
  - Step 4: Collaborate with others, leave comments, and fix conflicts
  - Step 5: Merge into main (or other branch)
- BUT, it's important to know what's going on when you branch and merge, because that's what's going on under the hood on GitHub/GitLab
- **The main goal here is to be very deliberate about what we put on main.**

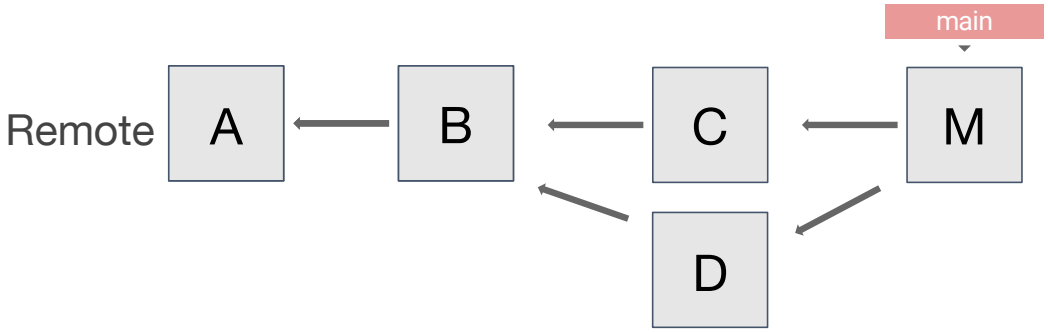
WORKING WITH REMOTE



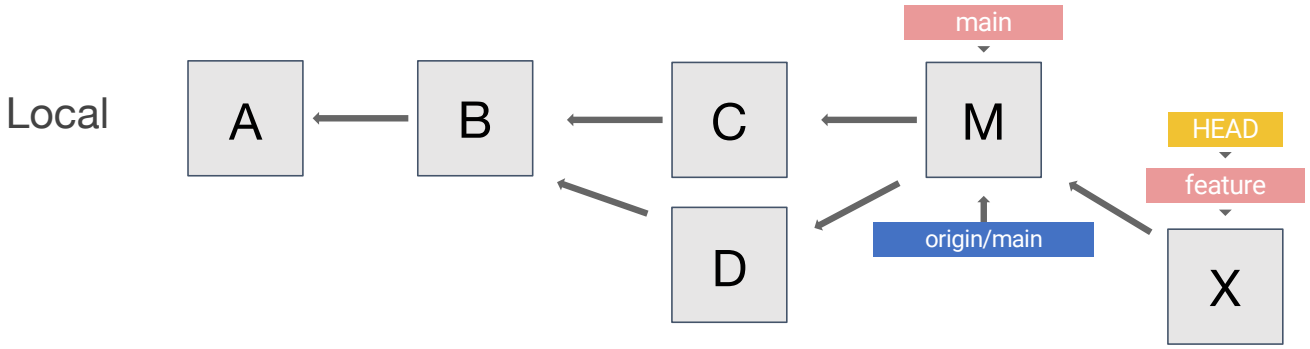


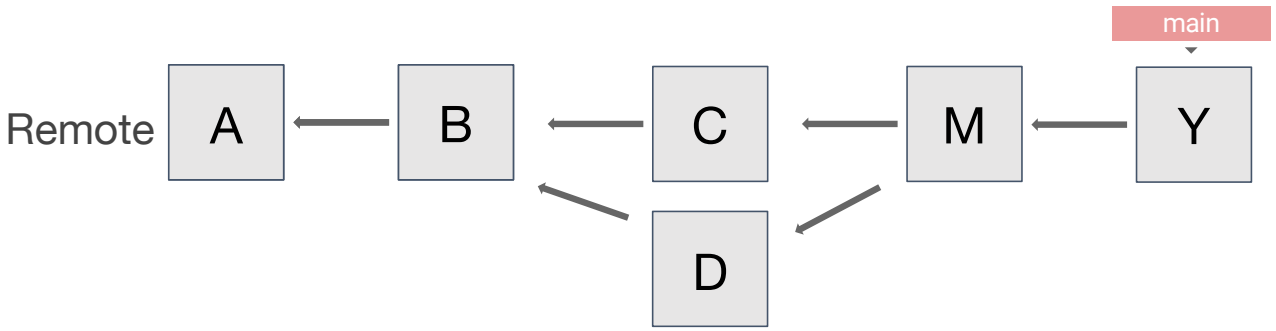
WORKING WITH REMOTE





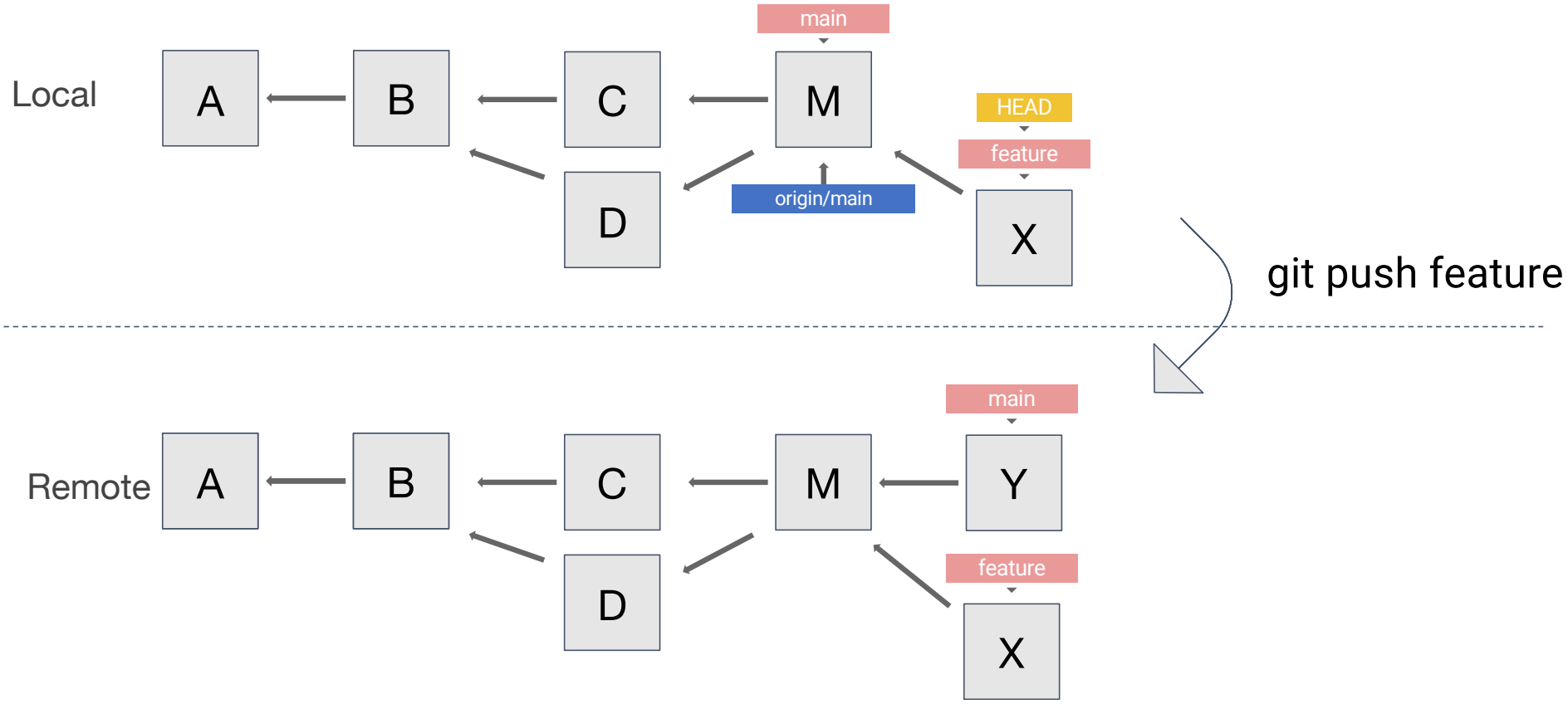
WORKING WITH REMOTE



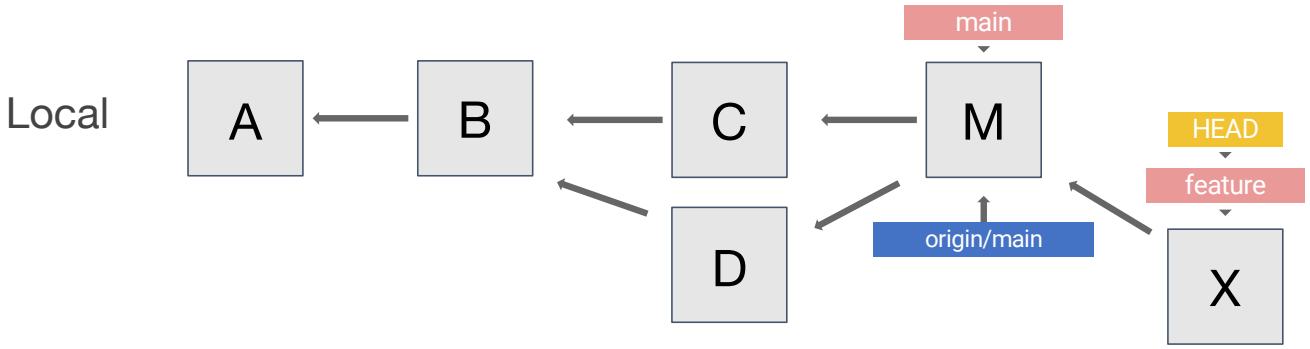


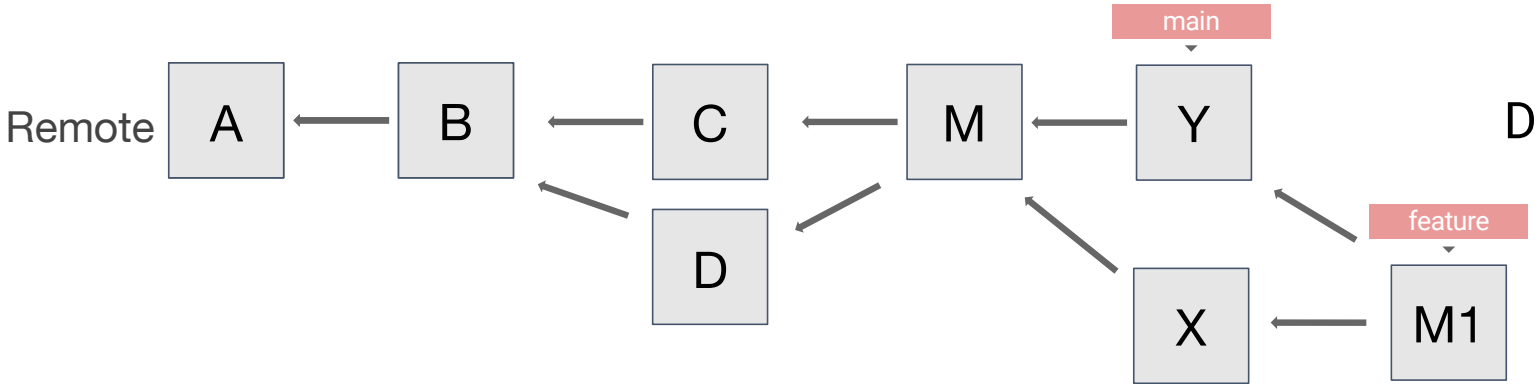
Commit happens on origin

WORKING WITH REMOTE



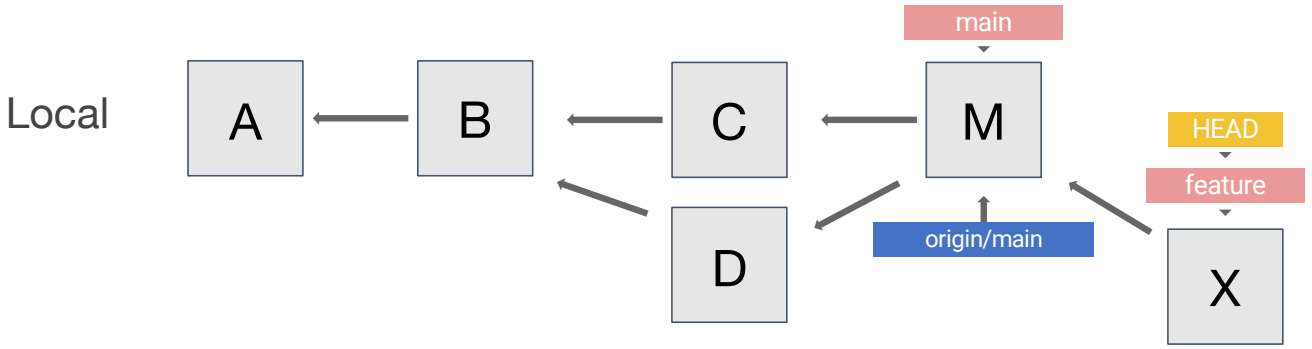
WORKING WITH REMOTE



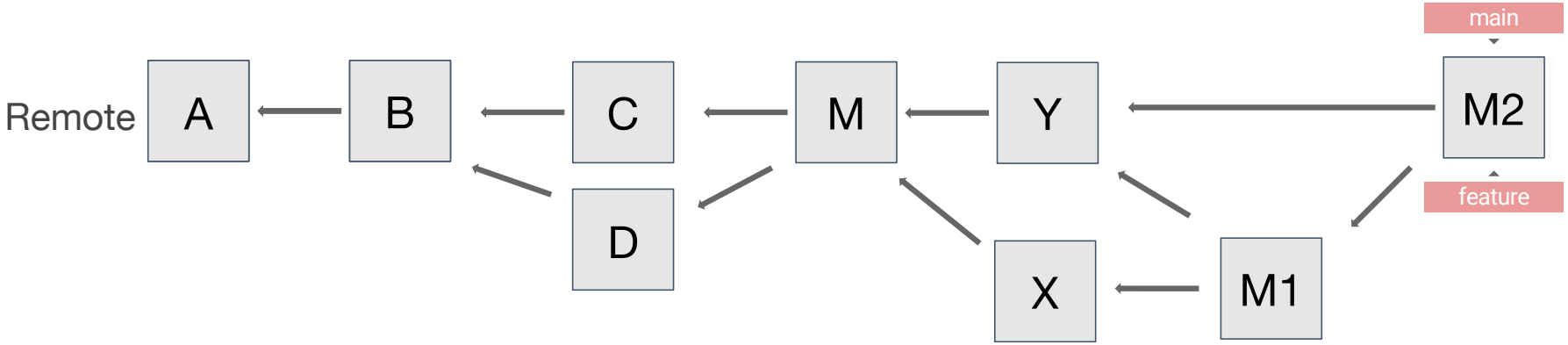


Do this on GitLab

WORKING WITH REMOTE



PULL REQUEST!!!



WORKING WITH GIT

- Many nice Git tutorials on YouTube (and other places)
- Here is a nice 30 minute Git & GitHub Crash Course For Beginners:

  https://www.youtube.com/watch?v=SWYqp7iY_Tc

  By Traversy Media

  Description: In this Git tutorial we will talk about what exactly Git is and we will look at and work with all of the basic and most important commands such as add, commit, status, push and more. This tutorial is very beginner friendly.