Collaborative Features of Version Control

UBCO Master of Data Science – DATA 533

Today’s Class

Fundamentals of version control

Fundamentals of Git

Useful Git commands

Git workflows

Version Control Systems

Version control is a system that

- Records changes to a file
- Set of files over time so that you can recall specific versions later.

Goal of a Version Control System

- Track versions of each file
- Handles concurrent changes from multiple sources (e.g., different developers working on the same code base in collaboration)

Git: Distributed Version Control

No central server required

Every user has a copy of every file

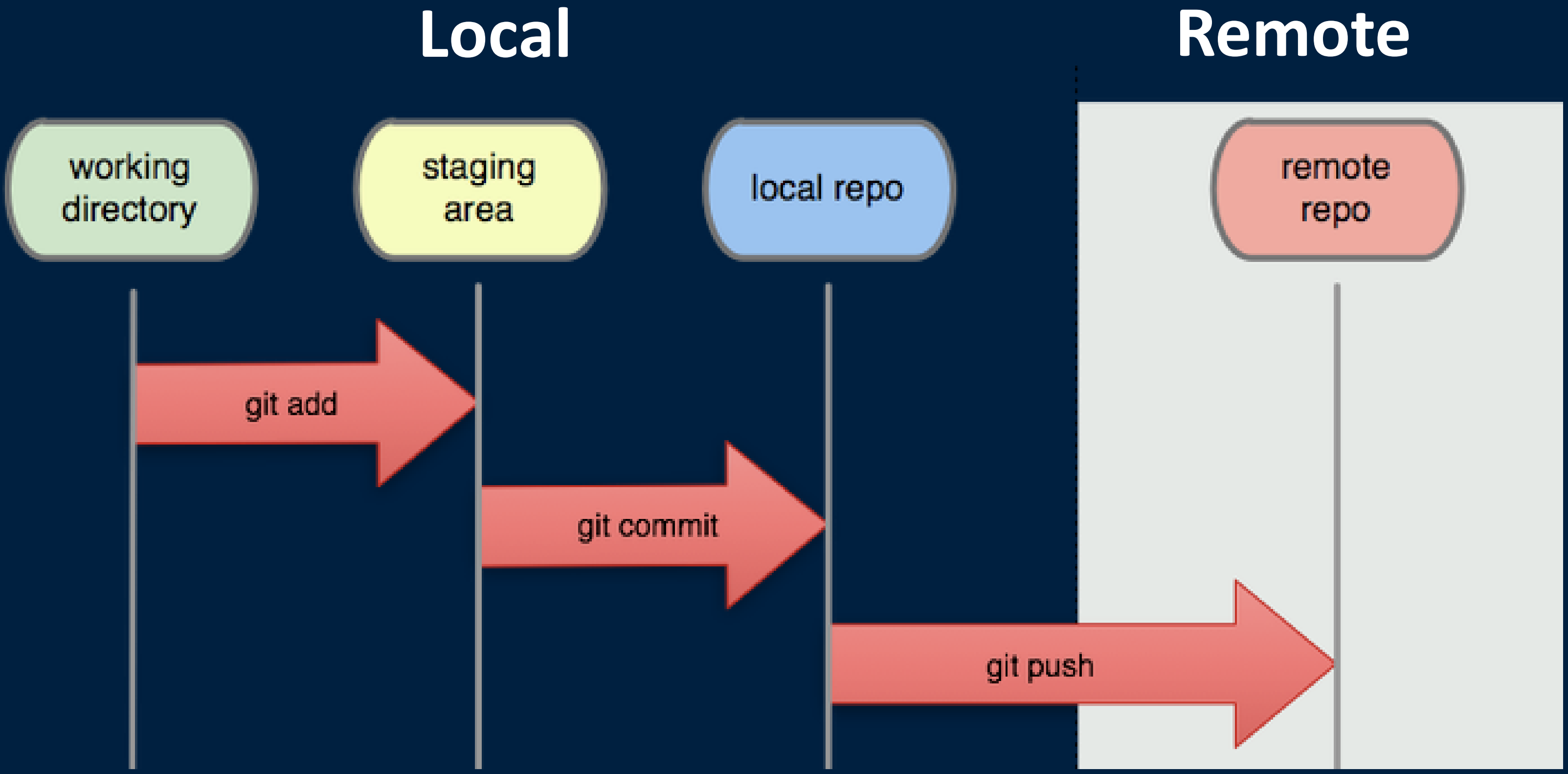
Very specific design goals

- Large-scale development
- Distributed

Git doesn’t require a server, but it’s common to use one for coordination

- Example: GitHub

Concepts


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push

Concepts

Working directory

- Local copy of the files that you’re working with
- This area is also known as the “untracked” area of git

Staging area

- A “place” where you tell Git to hold a set of changes, temporarily

Repository

- A place where Git stores copies of your files and their history
  - Local repository: on your working machine
  - Remote repository: a server (e.g., GitHub)

Get Ready to Use Git

Install a Git client: https://git-scm.com/downloads

Set the name and email for Git to use when you commit:

```
   $ git config --global user.name “your name”
  $ git config --global user.email your_email@email.com
```

Initializing a new repository:

```
$ git init <project directory>
```

Cloning an existing repository: Make a copy from a remote repo to your working directory

```
git clone <URL>
```

Inspecting and Saving Changes

The `git status` command displays the state of the working directory and the staging area

To add a file from the working directory to the staging area

```
$ git add file
$ git add directory
$ git add .
```

To Commit changes from staging area to repo

```
git commit -m "commit message"
```

Repo-to-Repo Collaboration

To push changes from local repo to remote repo. This enables other team members to access a set of saved changes.

```
$ git push
```

To pull (merge) changes from a remote repo to the working directory

```
$ git pull
```

Git Basic Question

Question: How many of the following statements are **TRUE**?

1) Git is a is distributed version control system
2) Git is designed to support linear development
3) Git is a web-based repository service
4) Git doesn’t require a central server

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Git Basic Question

Question: Which of the following command is used to merge changes from a remote repo to the working directory.

A) `git add`

B) `git commit`

C) `git clone`

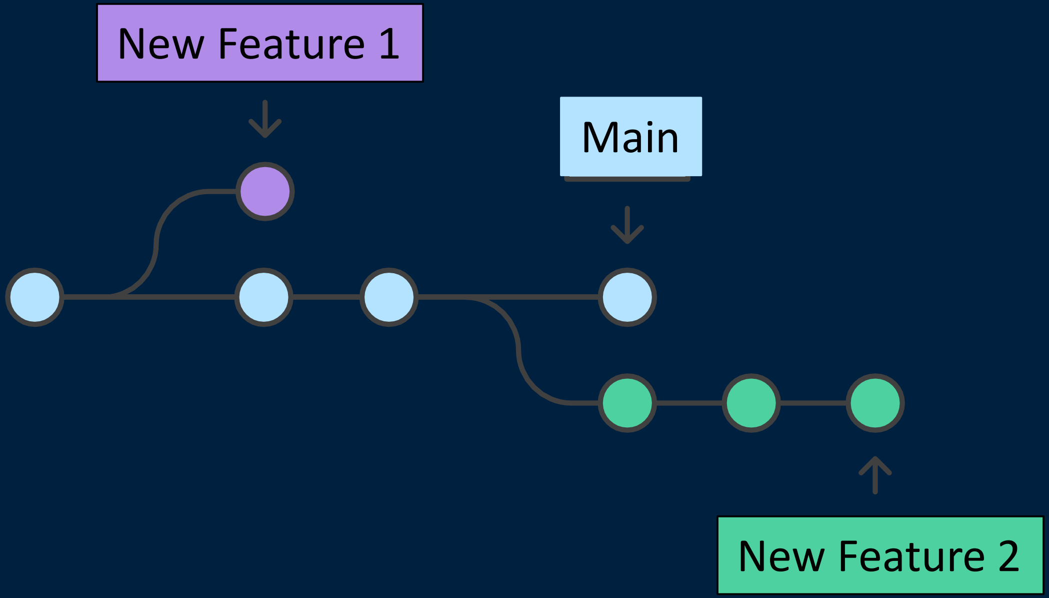
D) `git pull`

E) `git push`

Git Branch

A branch represents an independent line of development

It can be viewed as a way to request a brand new working directory, staging area, and project history

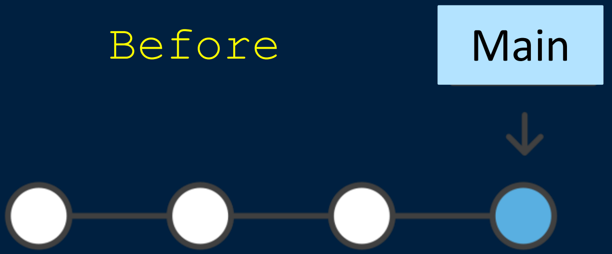


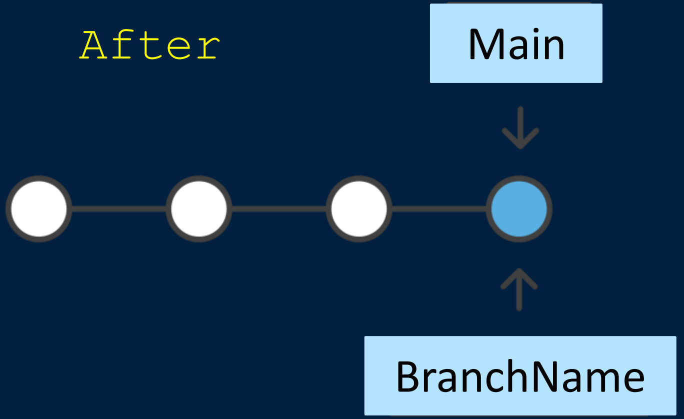
Git Branch

New commits are recorded in the history for the current branch

To show a list of branches: `git branch`

To create a new branch: `git branch BranchName`





Checking Out Branches

The `git checkout` command lets you navigate between the branches created by `git branch`

Checking out a branch updates the files in the working directory

The `git checkout` command accepts a `-b` argument that acts as a convenience method which will create the new branch and immediately switch to it.

```
git checkout -b <new-branch>
```

Checking Out Branches

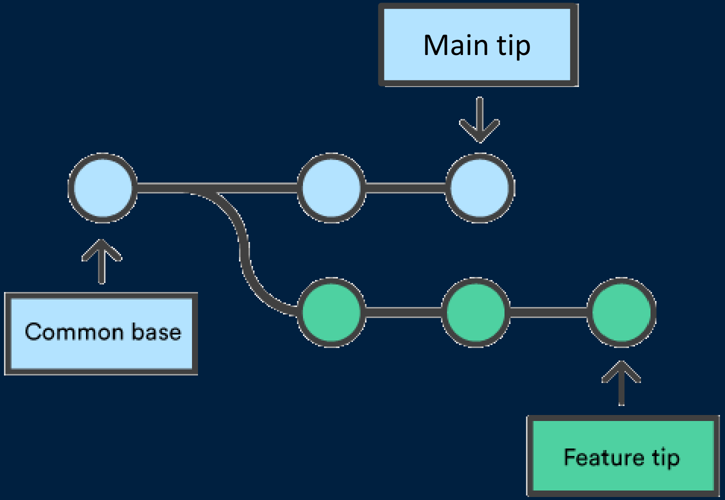
`git checkout` vs. `git clone`:

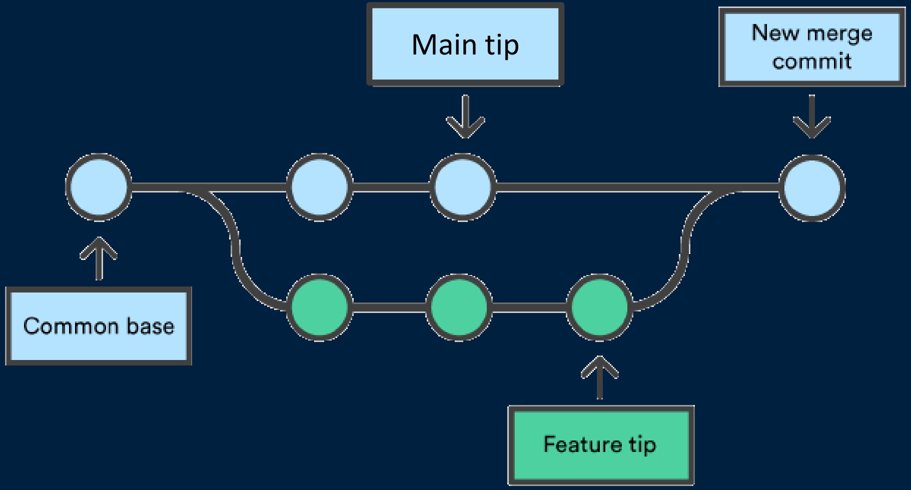
- `clone` works to fetch code from a remote repository
- `checkout` works to switch between versions of code already on the local system.

Git Merge

The `git merge` command lets you take the independent lines of development created by `git branch` and integrate them into a single branch.

Merge a branch into main branch: `git merge BranchName`





About git push

The git push command takes two arguments:

- A remote name, for example, origin
- A branch name, for example, main

`git push -u origin [branch]:` Useful when pushing a new branch, this creates an upstream (`--set-upstream` or `-u`) tracking branch with a lasting relationship to your local branch

The advantage of `-u` is, you may use `git pull` without any arguments

"`origin`" is a shorthand name for the remote repository

`git push --all:` Push all branches

Deleting Branches

Once you've finished working on a branch and have merged it into the main code base, you can delete the branch:

```
git branch -d BranchName
```

To force delete a specified branch, even if it has unmerged changes:

```
git branch -D BranchName
```

To delete a branch remotely (e.g., in GitHub):

```
git push <remote> --delete BranchName
```

Note: In most cases, `<remote>` will be `origin`.

Git Branch Question

Question: How many of the following statements are **TRUE**?

1) `git branch NewBranch` creates a new branch

2) `git checkout -b NewBranch` does not check out to the `NewBranch`

3) `git checkout` checkout command lets you navigate between the branches

4) `origin` is a shorthand name for the remote repository

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Viewing an Old Revision

The `git log` command displays committed snapshots.

```
git log --oneline
```

Output:

```
b7119f2 Updated World
872fa7e Created World.txt
a1e8fb5 Updated hello.txt
435b61d Created hello.txt
9773e52 Initialed import
```

Now find the ID of the revision you want to see and issue the following command:

```
git checkout a1e8fb5
```

To continue developing, you need to go back to the current state of your project:

```
git checkout main
```

Git Workflows

A Git Workflow is a recommendation for how to use Git to accomplish work.

As Git provides flexibility, there is no standardized process on how to interact with Git.

When selecting a workflow, it's most important that you consider a workflow that enhances the effectiveness of your team.

Git Workflows

Centralized Workflow/Basic Workflow

Forking Workflow

GitHub Flow

Feature Branching Workflow

Centralized Workflow

Each developer clones the central repository

Works locally on the code

Makes a commit with changes

Push it to the central repository for other developers to pull

Centralized Workflow: Steps

Hosted central repositories

- Example: GitHub

Clone the central repository

- `git clone URL`

Make changes and commit

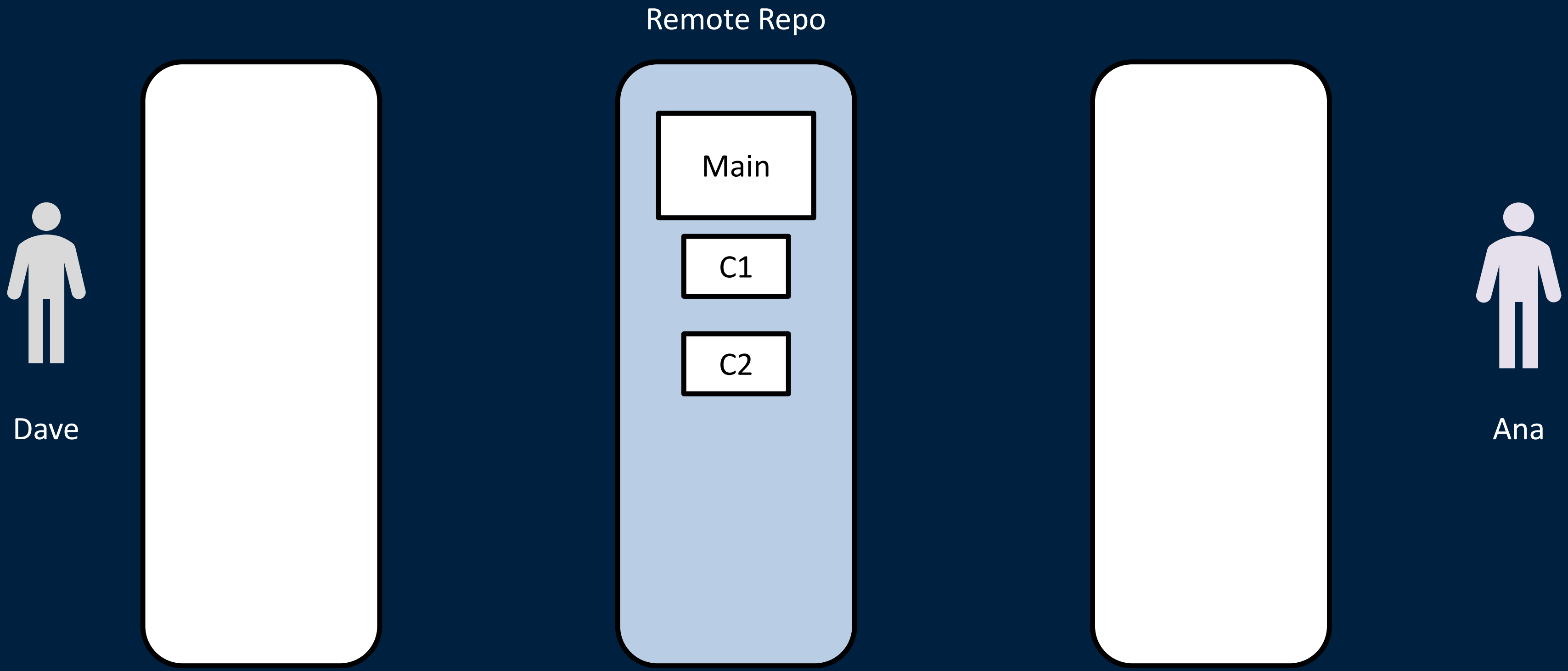
```
• git status                 # View the state of the repo
• git add <some-file>        # Stage a file
• git commit                 # Commit a file</some-file>
```

Push new commits to the central repository

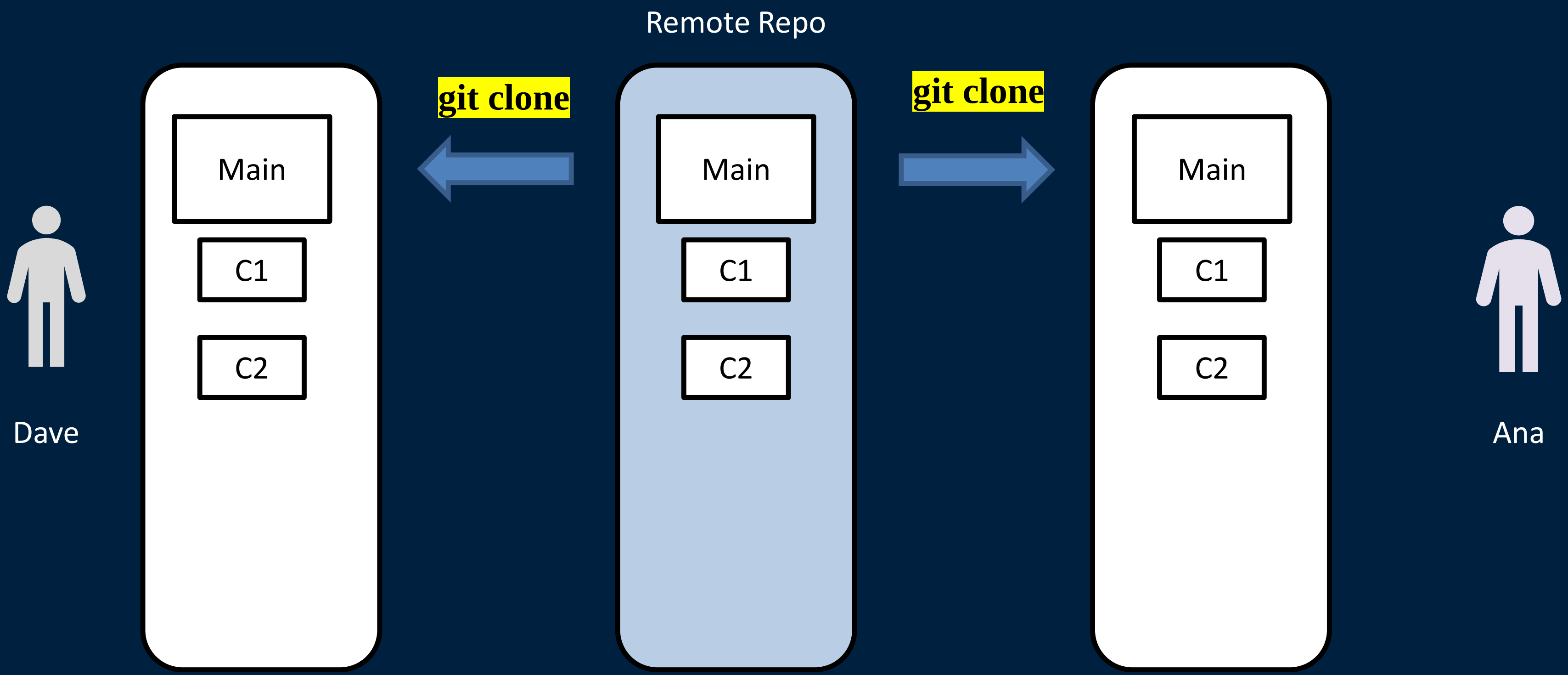
- `git push -u origin main`

Managing conflicts

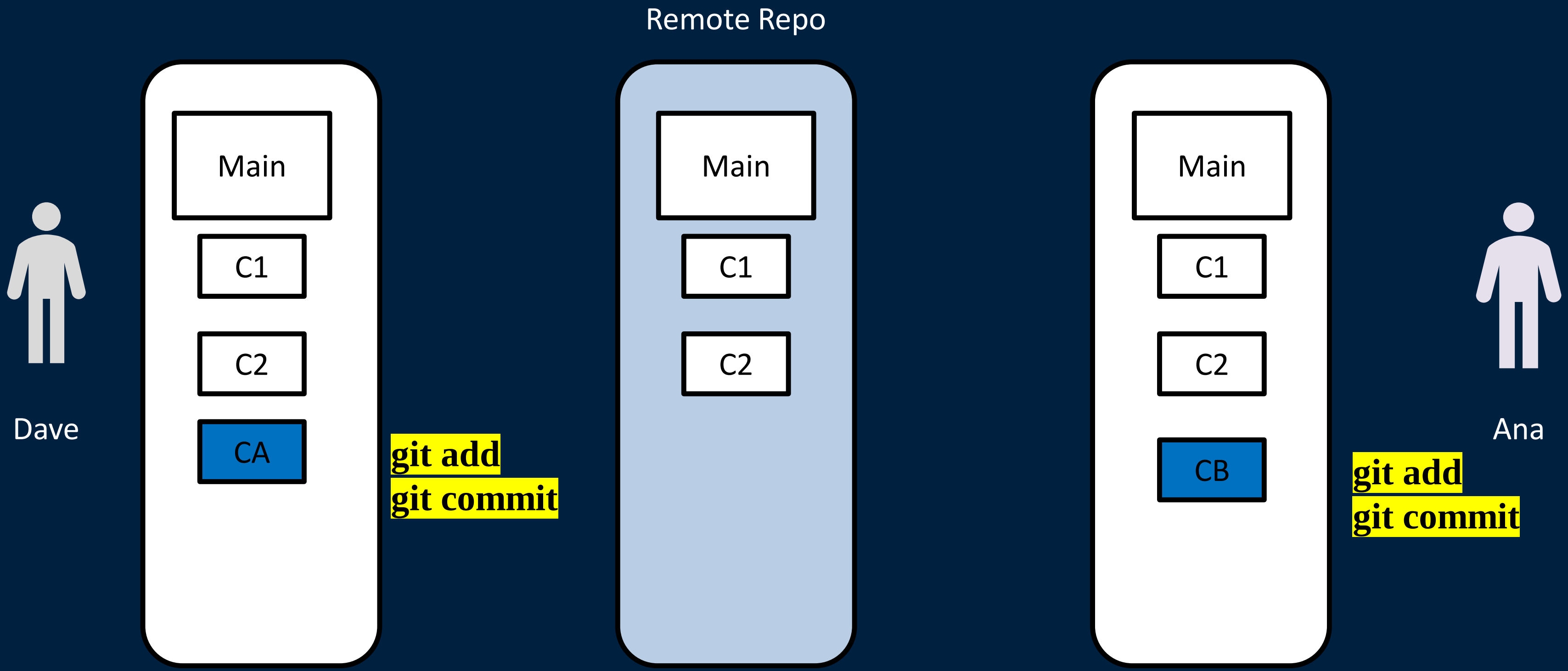
Collaboration with GitHub



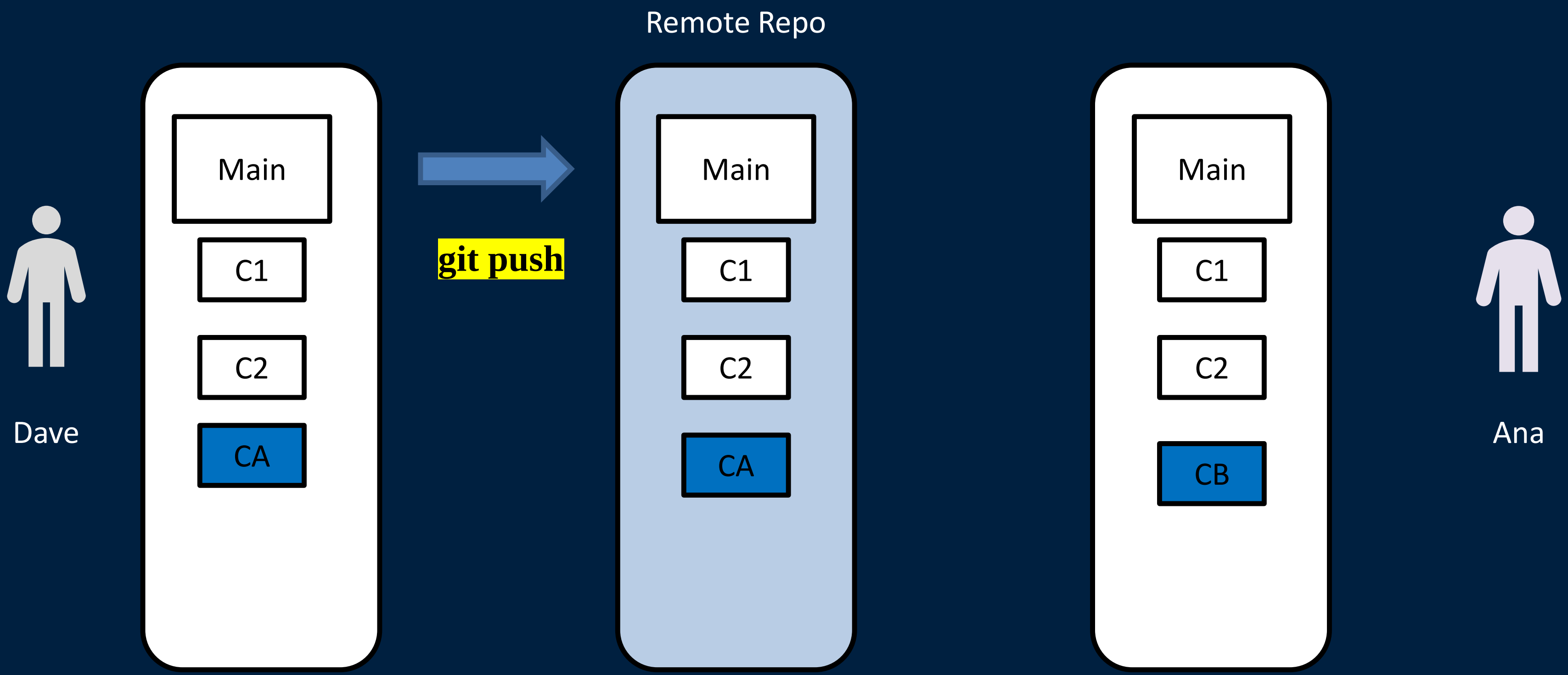
Collaboration with GitHub



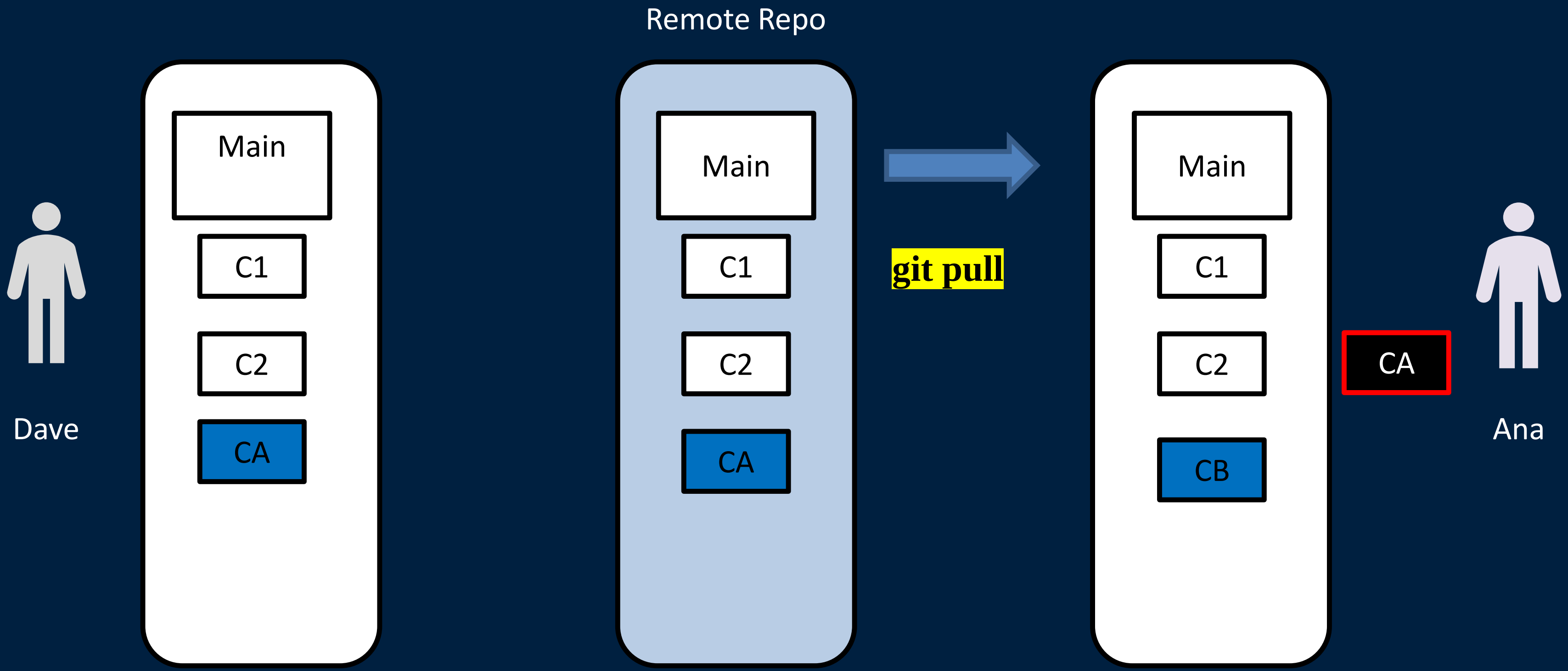
Collaboration with GitHub



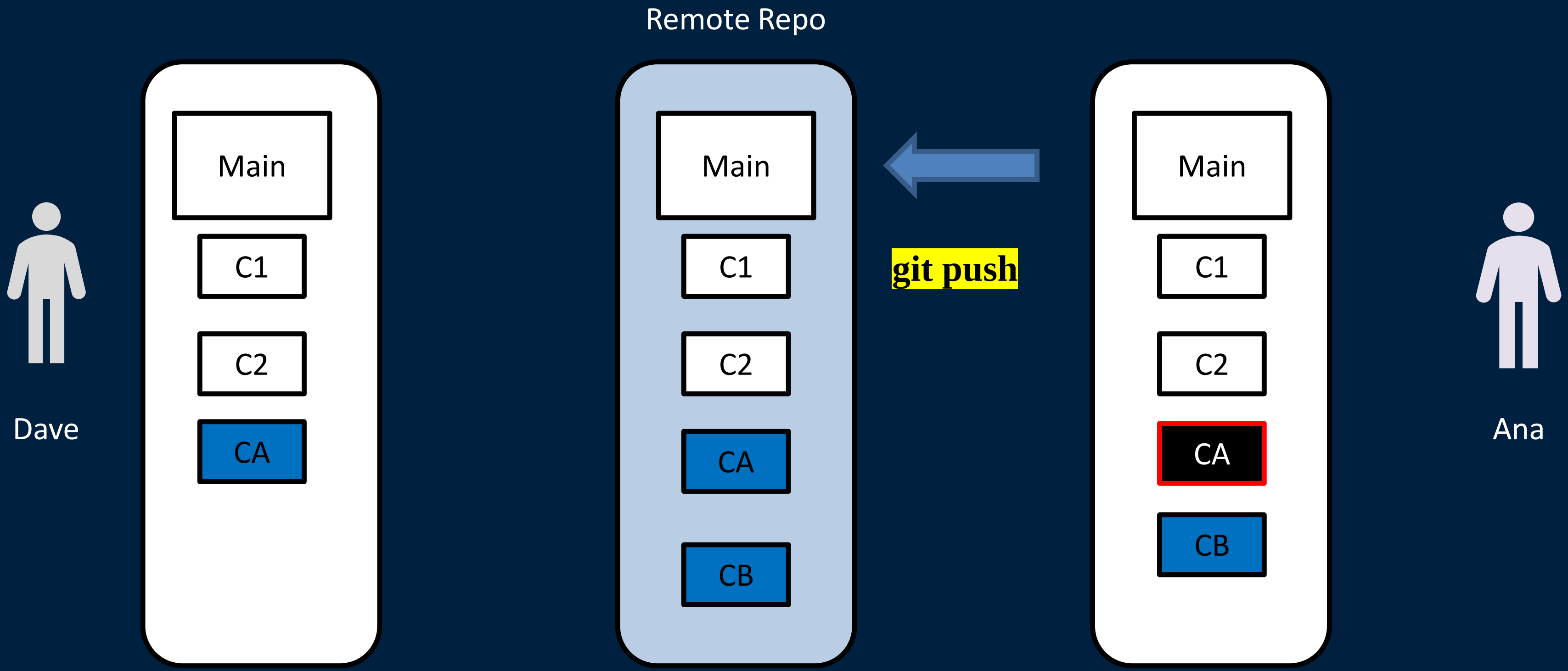
Collaboration with GitHub



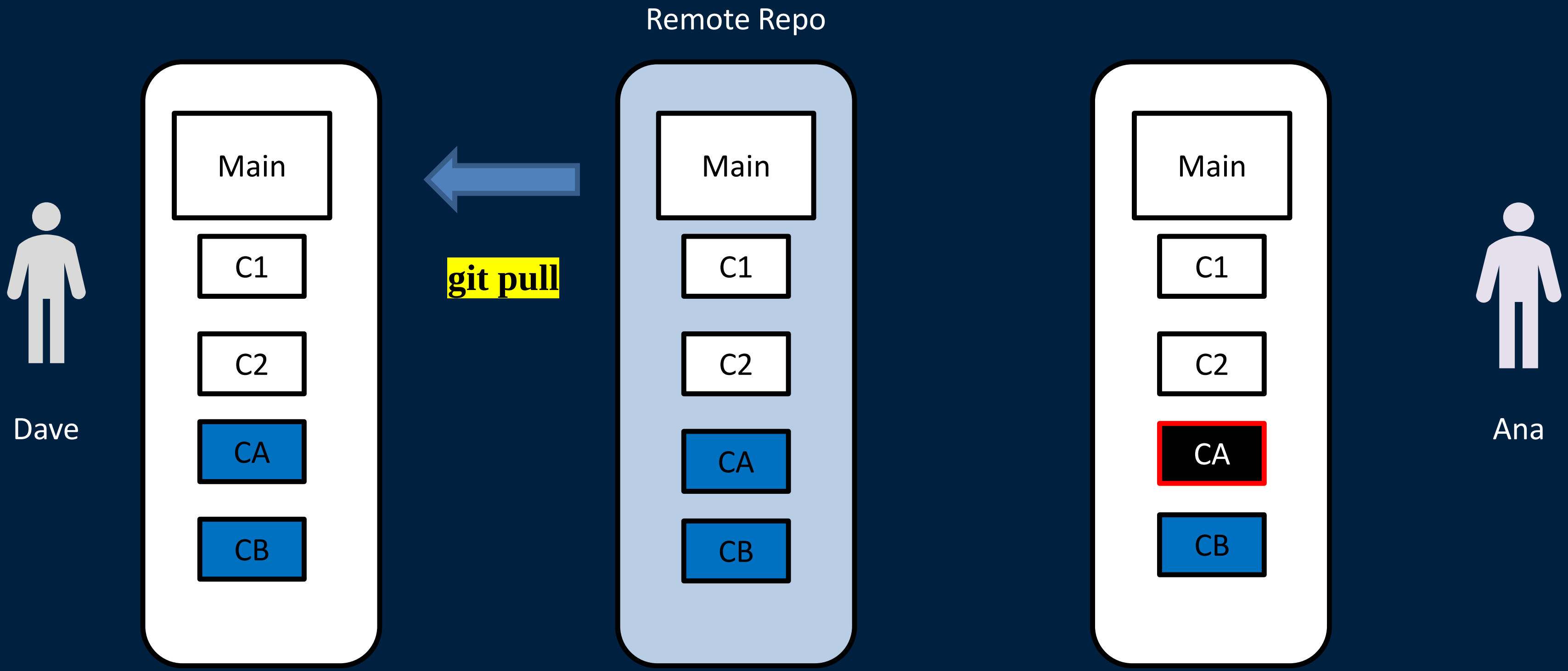
Collaboration with GitHub



Collaboration with GitHub



Collaboration with GitHub



Identifying Conflicts

A message appears to tell you that you need to integrate the remote changes before pushing

To do this you can make a pull request

Pull request shows information on what files the conflict resides in

```
error: failed to push some refs to 'https://github.com/…..'

hint: Updates were rejected because the remote contains work
that you do not have locally. This is usually caused by another
repository pushing to the same ref. You may want to first
integrate the remote changes

hint: (e.g., 'git pull ...') before pushing again.
```

Centralized Workflow: Steps

The markup follows a specific pattern.

```
<<<<<<< <pointer>
<local version>
=======
<pulled version>
>>>>>>> <pulled commit id>
```

When the conflict has been resolved, you can make the push to the remote repository

Open files that has a conflict and let you choose a solution.

Try It: Centralized Workflow

Member 1

- Create a new repository in your GitHub account with a README.md file
- Add member 2 as a contributor to the repository (login to GitHub, go to “Settings” → “Collaborators” →“Manage Access” → “Add people”)
- Clone the repository on your local machine (also ask member 2 to start his/her tasks)
- Make some changes to the README.md file
- Add, commit and push your repo to GitHub
- If there is a conflict, resolve the conflict by accepting all the changes

Member 2

- Clone the repository on your local machine
- Make some changes to the README.md file
- Add, commit and push your repo to GitHub
- If there is a conflict, resolve the conflict by accepting all the changes

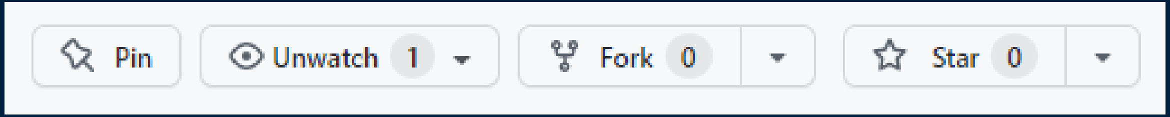
Forking Workflow

Developers can "fork" a project :

- If developers want to contribute to an existing project to which you don't have push access

When developers "fork" a project, GitHub will make a copy of the project.

To fork a project, visit the project page and click the "Fork" button at the top-right of the page.



Forking Workflow

They can clone it locally, create a branch, make the code change and finally push that change back up to GitHub.

```
$ git clone URL
$ git checkout -b branch_name


$ git add .
$ git commit -m "added button feature"
$ git push origin branch_name
```

Pull Requests

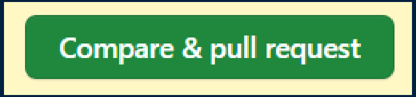
Once someone completes a feature, they don’t immediately merge it into main.

They push the feature branch to the central server and file a pull request asking to merge their additions into main.

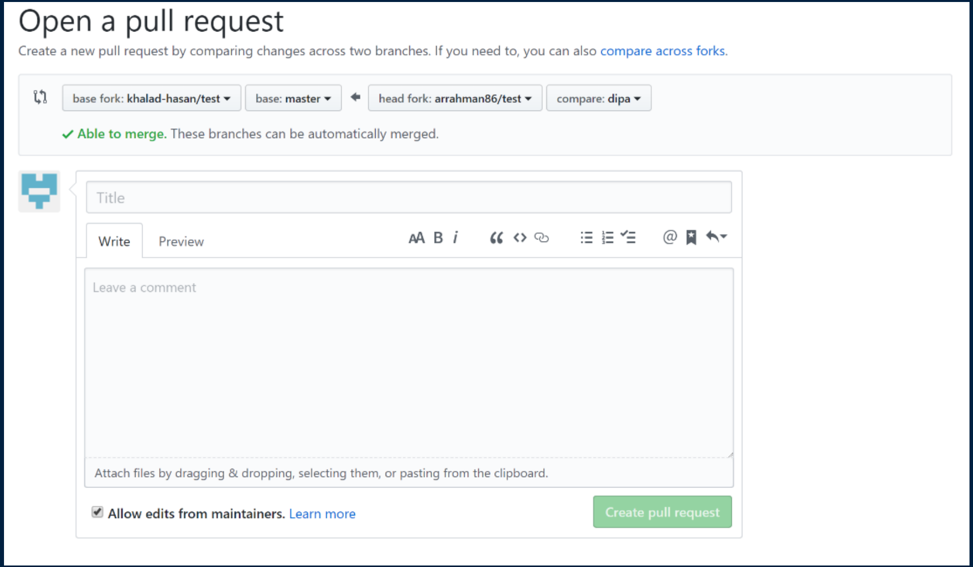
Pull Requests initiate discussion about your commits

Forking Workflow

On the GitHub, the developer can see that GitHub noticed that a new branch is pushed and open a Pull Request.



Once the button is clicked, a screen asks Pull Request description.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base fork: khalad-hasan/test ▾ | base: master ▾ | ← | head fork: arrahman86/test ▾ | compare: dipa ▾

✓ Able to merge. These branches can be automatically merged.

Title

Write | Preview

Leave a comment

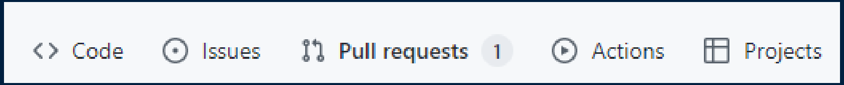
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

☑ Allow edits from maintainers. Learn more

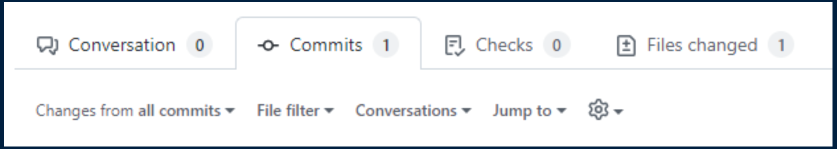
Create pull request

Forking Workflow

When the developer hits the *Create pull request* button, the owner of the project get a notification that someone is suggesting a change

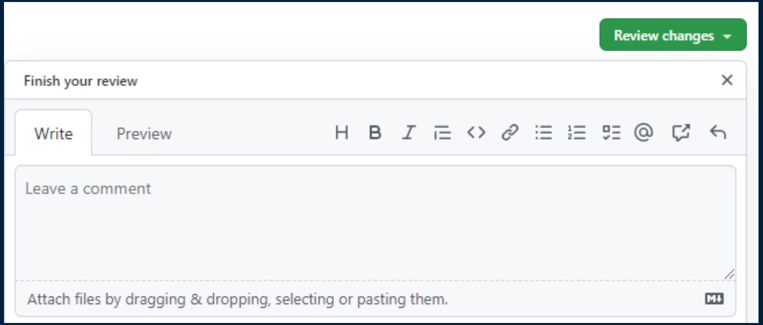


At this point, the project owner can look at the suggested change and merge it, reject it or comment on it.



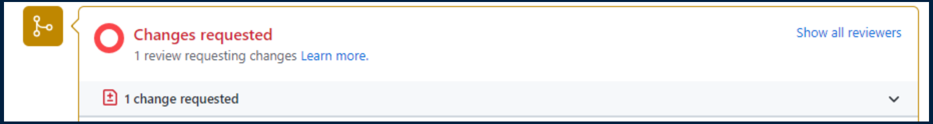
Forking Workflow

At this point, the project owner can look at the suggested change and merge it, reject it or comment on it.



Leave a comment

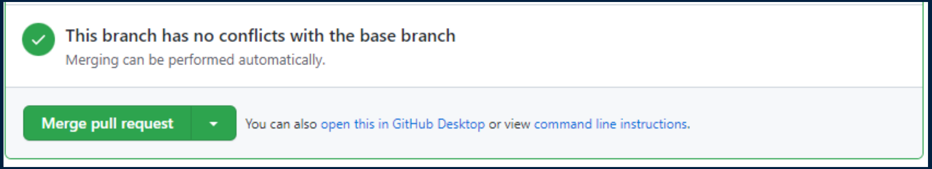
Attach files by dragging & dropping, selecting or pasting them.



Changes requested

1 review requesting changes Learn more.

1 change requested



This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Try It: Forking Workflow

Member 1:

- Create a repository in GitHub and share the link with your group member.
- Ask the other group member to Create a fork of the repo.

Member 2:

- Follow the steps as shown the slides (clone $\rightarrow$ create new branch $\rightarrow$ make some changes $\rightarrow$ add $\rightarrow$ commit), push it to a branch your repo, and make a pull request.

Member 1:

- Review the changes and take necessary steps to merge the pull request.

Feature Branch Workflow

All feature development should take place in a dedicated branch.

Multiple developers to work on a particular feature without disturbing the main codebase.

`main` branch never contains broken code

Feature Branch Workflow: Steps

Start with the main branch

- `git checkout main`

Create a new branch

- `git checkout -b new-feature`

Update, add, and commit

- `git status`
- `git add <some-file>`
- `git commit`

Push feature branch to remote

- `git push -u origin new-feature`

Resolve feedback (Pull request)

- Team members comment and approve the pushed commits

Example

Mary begins a new feature branch

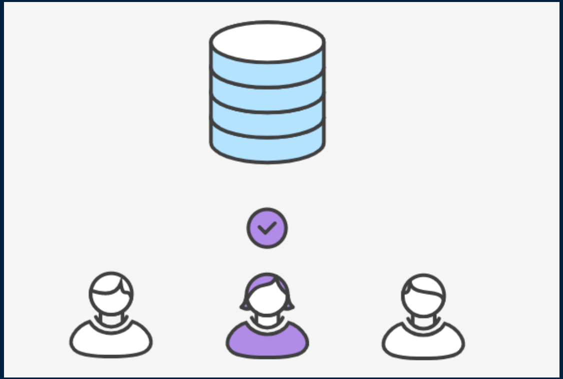
```
git checkout -b marys-feature
```

On this branch, Mary adds, and commits changes

```
git status
git add <some-file>
git commit
```

Example

Mary finishes her feature



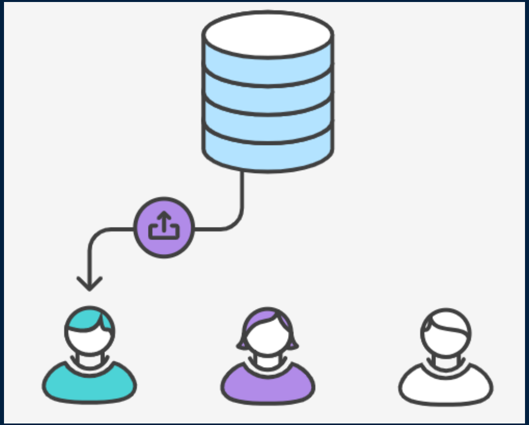
```
git push -u origin marys-feature
```

She files the pull request to merge marys-feature into main

Team members will be notified automatically

Example

Bill receives the pull request



Bill takes a look at `marys-feature`.

He and Mary have some back-and-forth via the pull request to make a few changes and finally publishes her work.

Git Branch Question

Question: How many of the following statements are **TRUE**?

1) We use `git fork` to download a remote repository from GitHub to our local computer

2) We use `git checkout` to upload changes and code back to GitHub

3) If we want to make major changes to a project, we should implement our changes in a branch

4) Pull request can be issued when someone completes a feature, but they don't want it to merge into main.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

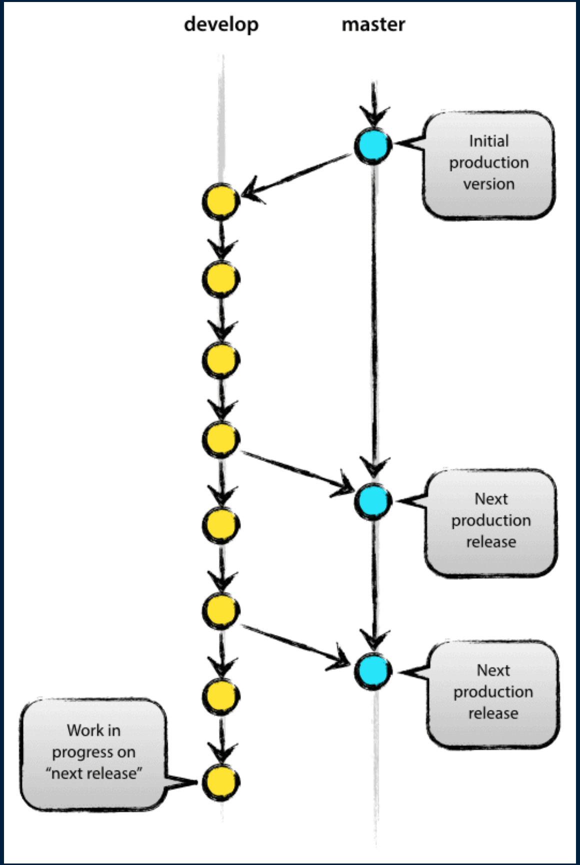
Git Branching Model

The central repo holds two main branches with an infinite lifetime:

- Main
- Develop

Main to be the main branch where the source code of HEAD always reflects a production-ready state.

Develop to be the main branch where the source code of HEAD always reflects a state with the latest development for the next release


develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

Feature Branch

Branch off of develop

Primary working branches for an individual(s)

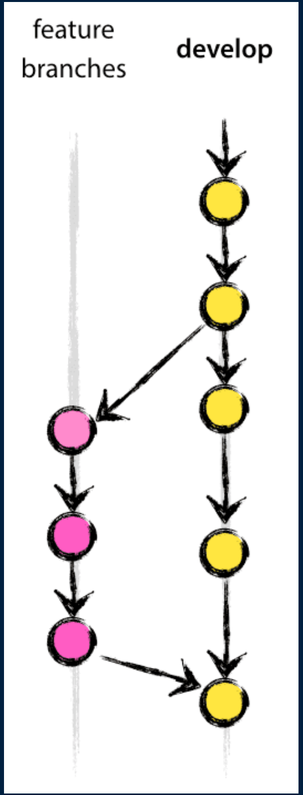
When a new feature is finished

- Merge into develop
- Code reviews

Can be many feature branches being developed in parallel and merged it later

If the feature is a failure

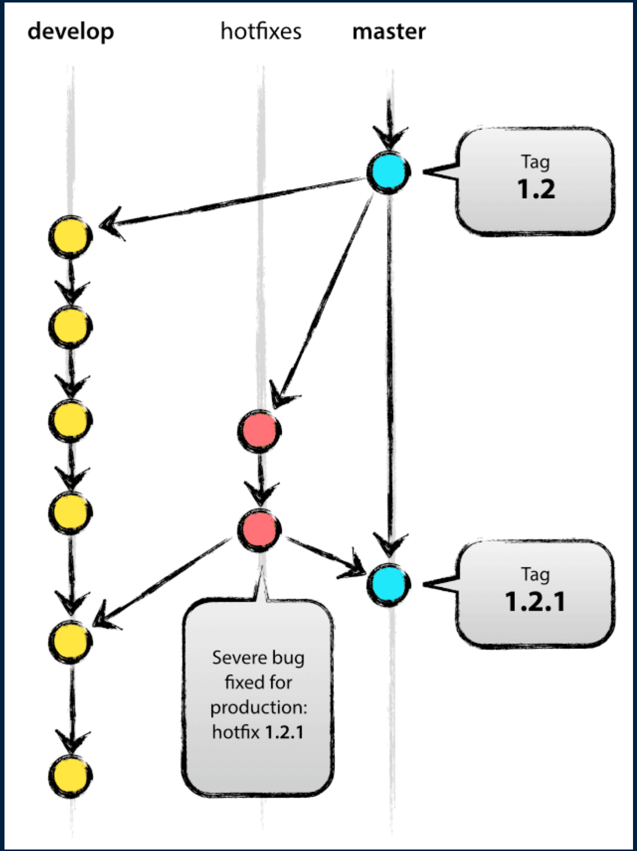
- Delete the branch without merging into develop



Hotfixes

Production release contains a bug

- Create a hotfix branch from main
- Fix the bug
- merge fix into:
  - Develop



Release Branch

Branch off develop when approaching a release

No features added

Extensive testing and bug fixes

- Merge all changes back to develop

When confidently stable

- Merge into main as a release

Objectives

- Set up a Git repository
- Perform repository-to-repository collaboration
- Resolve Git conflicts
- Create, check out and merge branches
- Learn and apply Git Workflows

THE UNIVERSITY OF BRITISH COLUMBIA