

# Version Control

UBCO Master of Data Science – DATA 541



# Today's Class

---

Fundamentals of version control

Fundamentals of Git

Useful Git commands

Git workflow

# How Do We Share Code

Working in a team, you need to share code

- How?

Email attachments?

Dropbox?

What if two people edit the same file at the same time?

- Google Docs? Dropbox?
  - No merging in Dropbox
  - GoogleDocs not meant for code
    - No syntax highlighting, etc.



# Version Control Systems

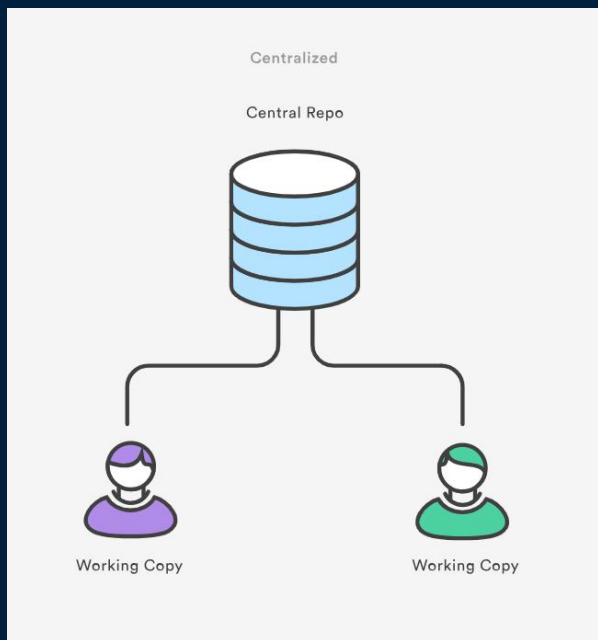
## Goal of a Version Control System

- Software that manages changes that you make to your files (source code)
- Track versions of each file
- Handles concurrent changes from multiple sources (e.g., different developers working on the same code base)



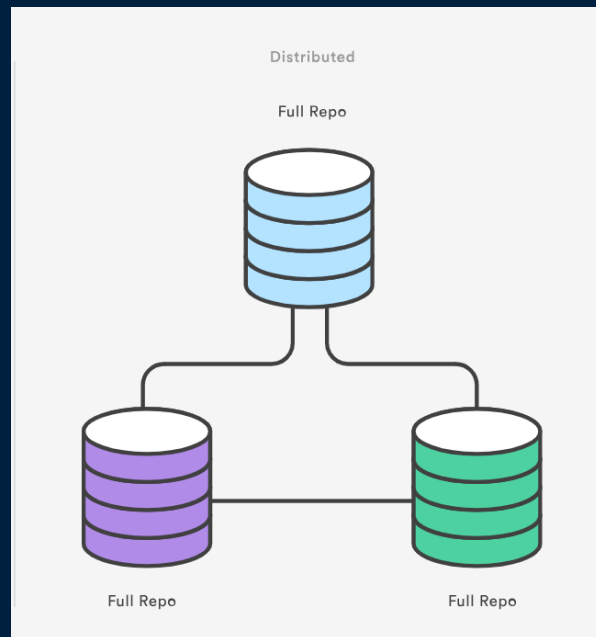
# Centralized and Distributed Models

## Centralized Model



Example: Subversion

## Distributed Model



Example: 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

# Git History

---

Came out of the Linux development community

Linus Torvalds, 2005

Initial goals:

- Support for non-linear development
- Fully distributed
- Speed
- Able to handle large projects



# 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

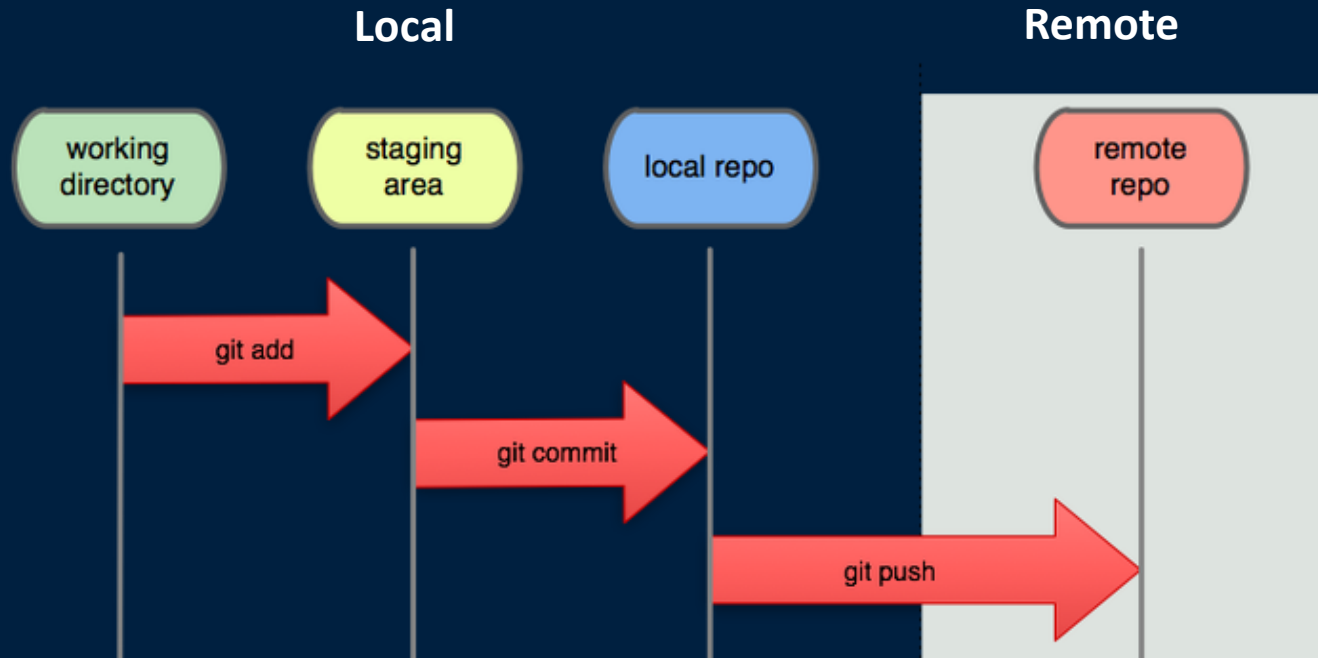
- A “place” where you tell Git to hold a set of changes, temporarily

## 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)



# Concepts



# Commands

---

You perform these operations using a Git client

- Command-line or GUI

Commands typically move files between working directory, staging area and local or remote repository

# Git Basic Question

---

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

- 1) Git is a distributed version control system
- 2) Git is designed to support parallel development
- 3) Git is a web-based repository Service
- 4) Git doesn't require a server

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

# Get Ready to Use Git

---

First, you need to install and configure 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
```

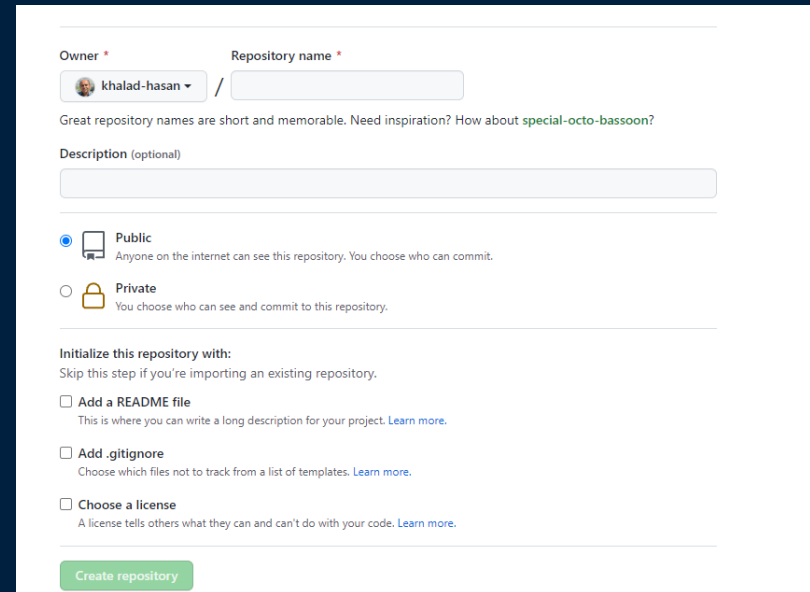
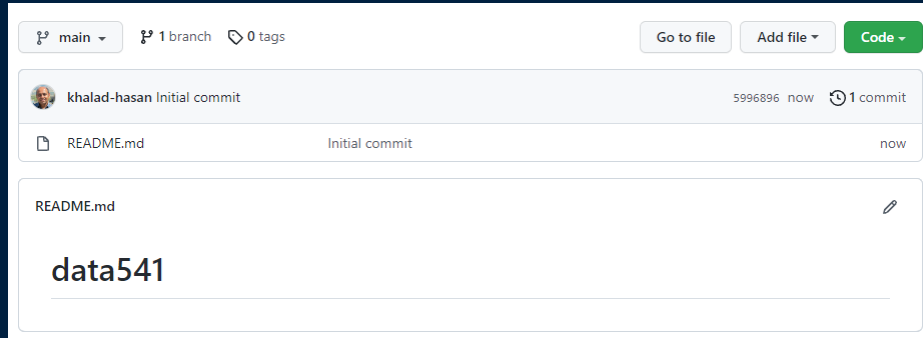
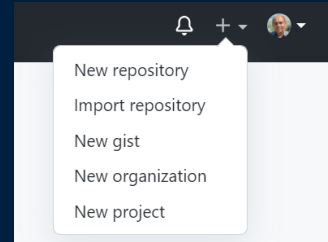
You can call `git config --list` to verify these are set.

# Create a new repository

In the upper right corner, next to your avatar, click and then select **New repository**.

Name your repository

Initialize repository with a README.



# Setting Up a Repository

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

```
git clone <URL>
```

The screenshot shows a GitHub repository page for user 'khalad-hasan'. The repository is named 'data541'. The 'main' branch is selected. A 'Clone' dropdown menu is open, showing options for cloning via HTTPS, SSH, or GitHub CLI. The HTTPS URL is `https://github.com/khalad-hasan/data541.git`. A blue arrow labeled 'Main' points to the 'main' branch selector. Another blue arrow labeled 'URL' points to the HTTPS clone URL in the dropdown menu.

# Inspecting a Repository

---

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

The `git log` command displays committed snapshots. It let you see a list of the project history and search for specific changes.

```
git log      or
```

```
git log --oneline
```

```
a1e8fb5 Updated hello.txt  
435b61d Created hello.txt  
9773e52 Initial commit
```

Note: changes will be listed by commitID



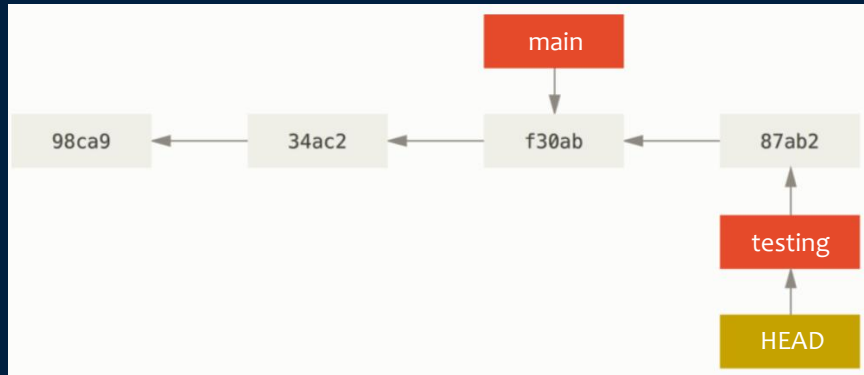
# Key Concepts

## Main

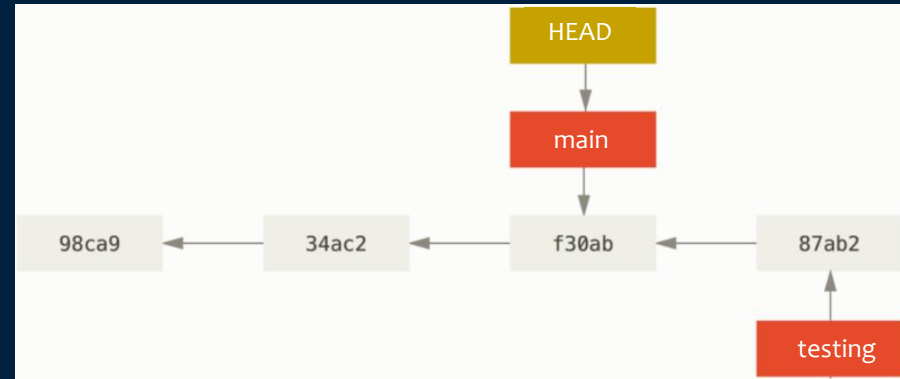
- The main (previously master) branch in your project

## Head

- A reference to the most recent commit (in most cases – not always true!)



git checkout testing



git checkout main

git status to see what branch you are on

# Saving Changes

---

To add a file from working directory to 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 remote repo to working directory

```
$ git pull
```

# Git Basic Question

---

**Question:** Which of the following command creates a copy of a remote repository to your local machine.

- A) `git add`
- B) `git commit`
- C) `git clone`
- D) `git pull`
- E) `git push`

# Setting Up a Repository

---

To set up a project locally then create the repository on GitHub and push it to remote.

Initializing a new repository:





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

Next steps

- Add files and commit
- Go to GitHub first
- Create a new repository and name it whatever you want to store it in GitHub.

# Setting Up a Repository

Quick setup — if you've done this kind of thing before

 Set up in Desktop
 or
  HTTPS
  SSH
 


Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# data_541" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/khalad-hasan/data_541.git
git push -u origin main
```

...or push an existing repository from the command line

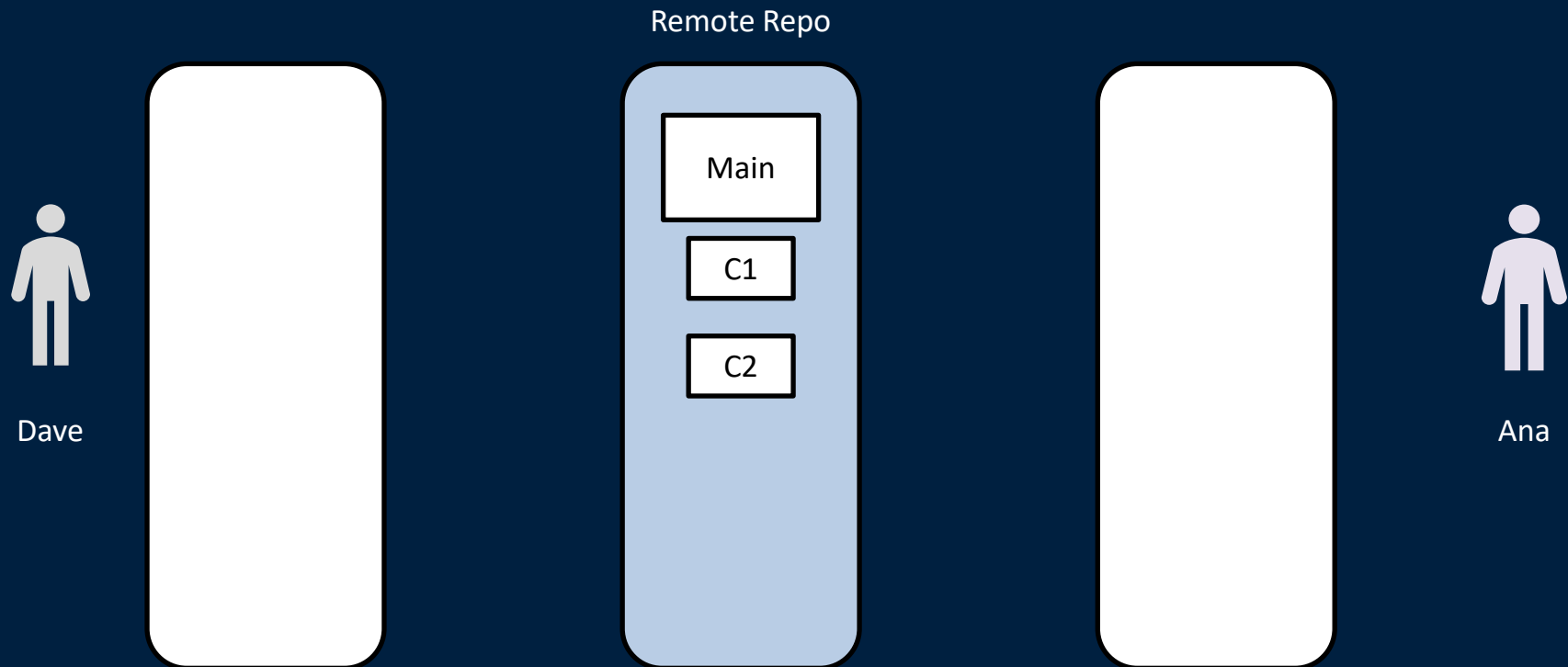
```
git remote add origin https://github.com/khalad-hasan/data_541.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

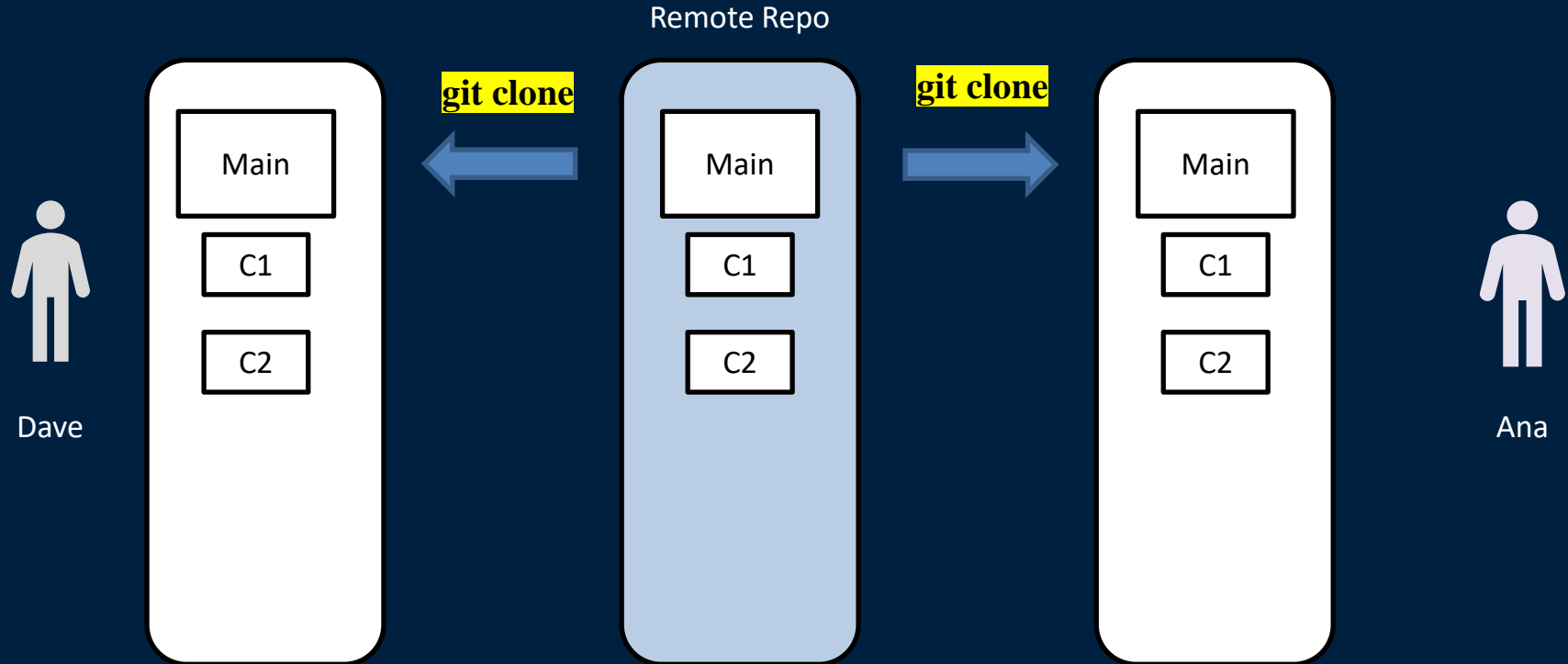
[Import code](#)

# Collaboration with GitHub

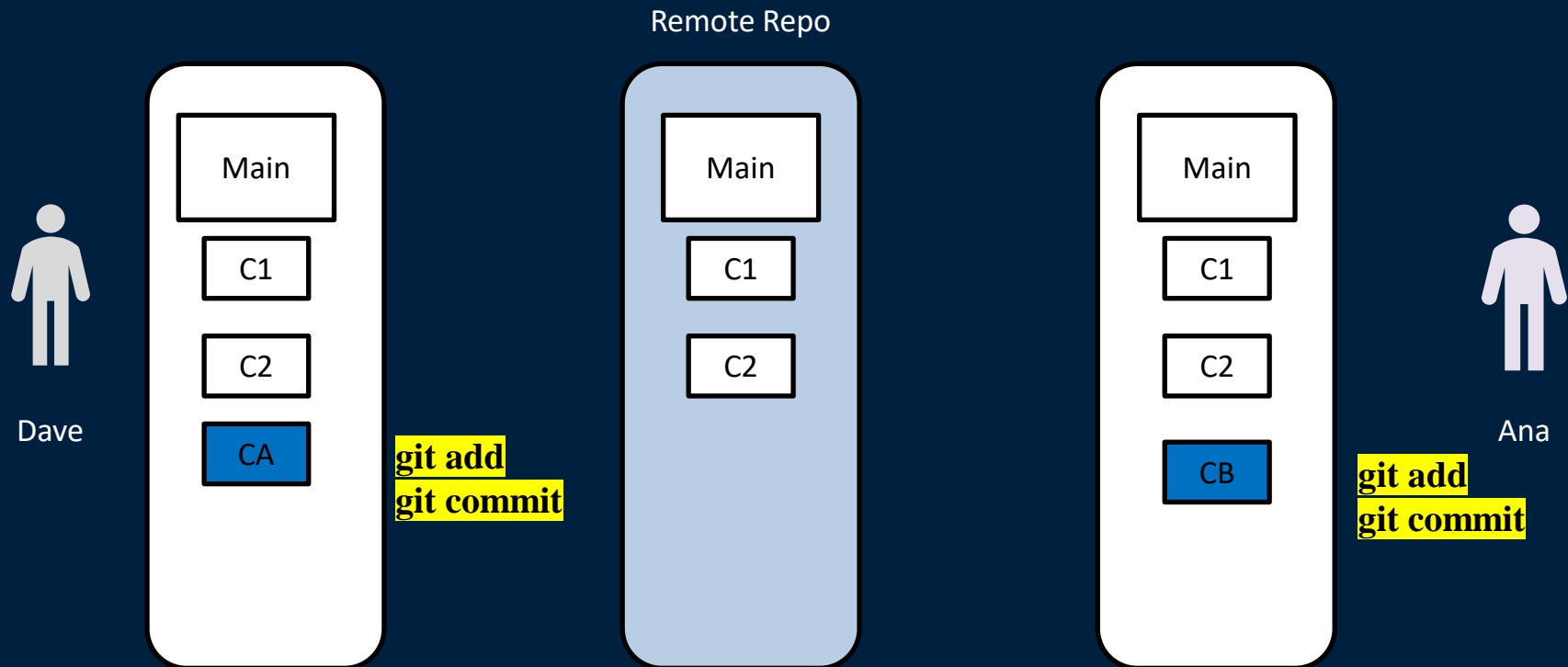




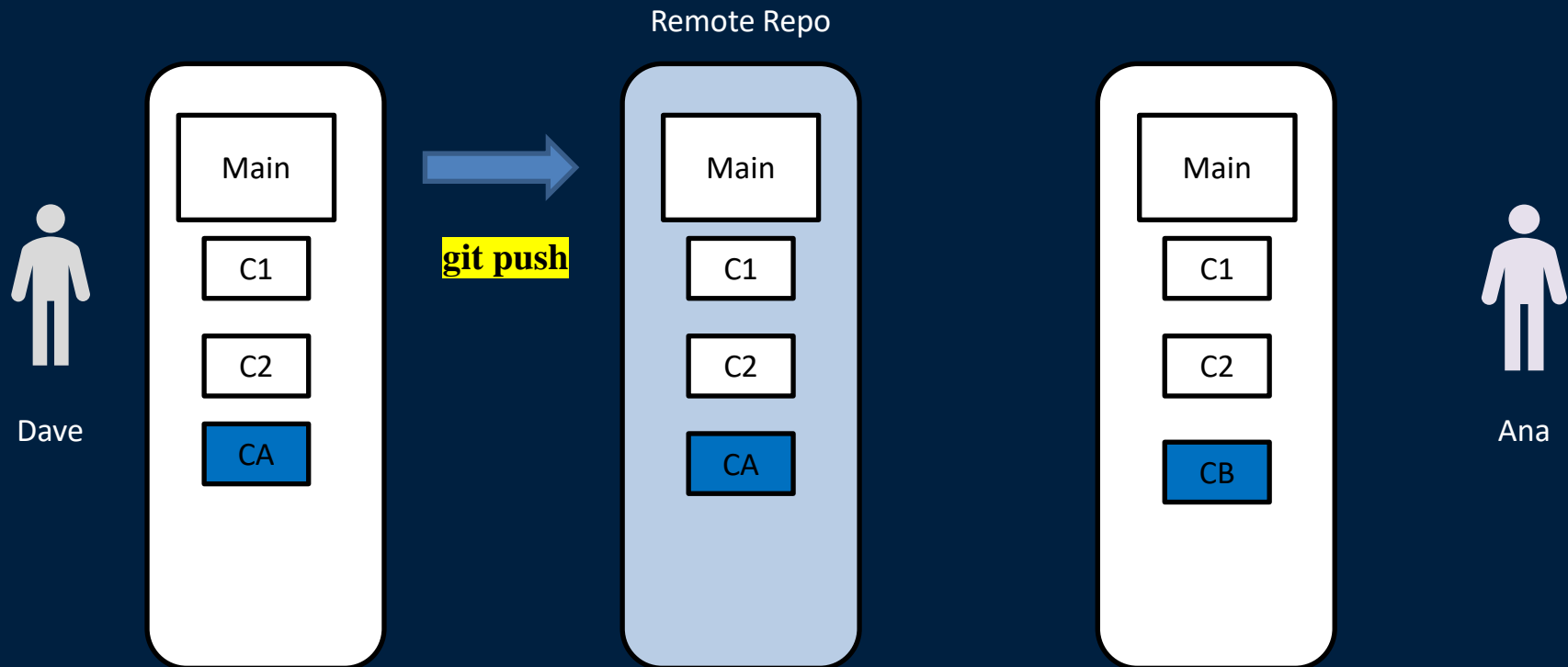
# Collaboration with GitHub



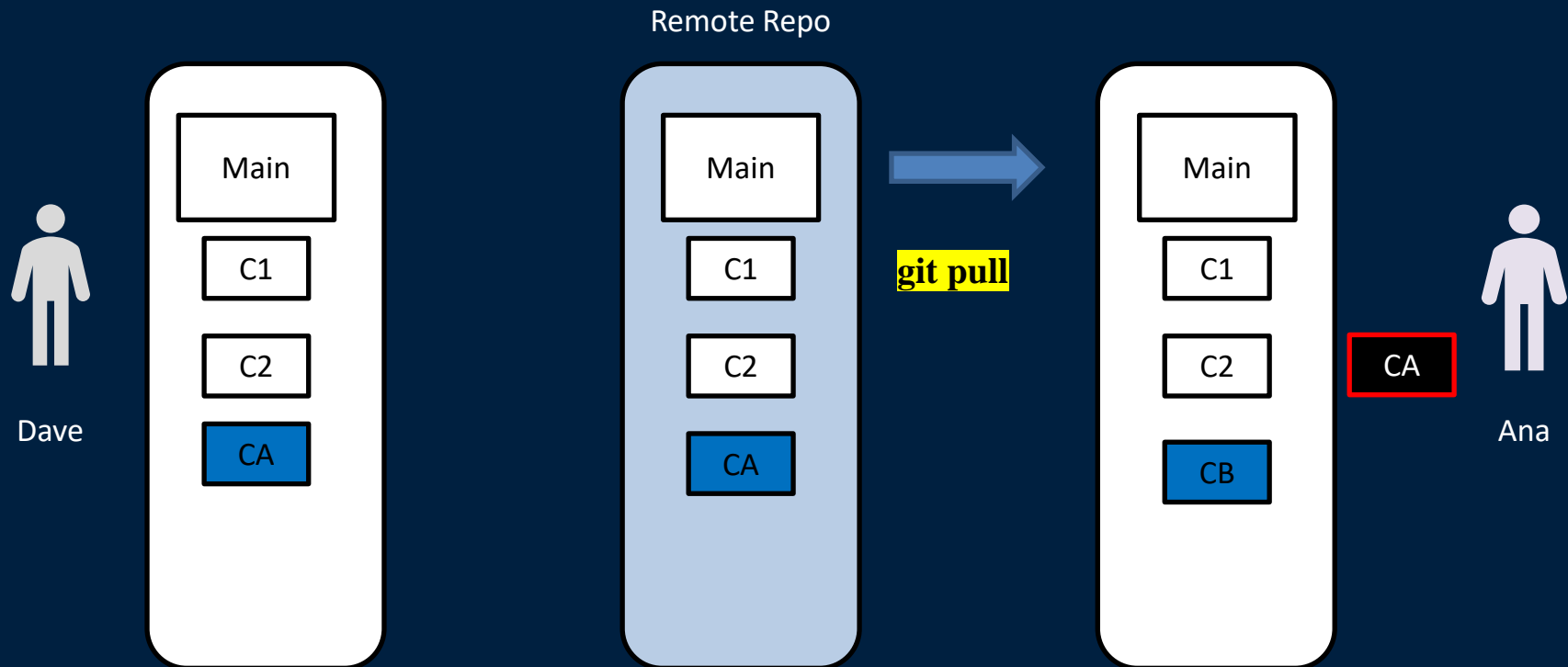
# Collaboration with GitHub



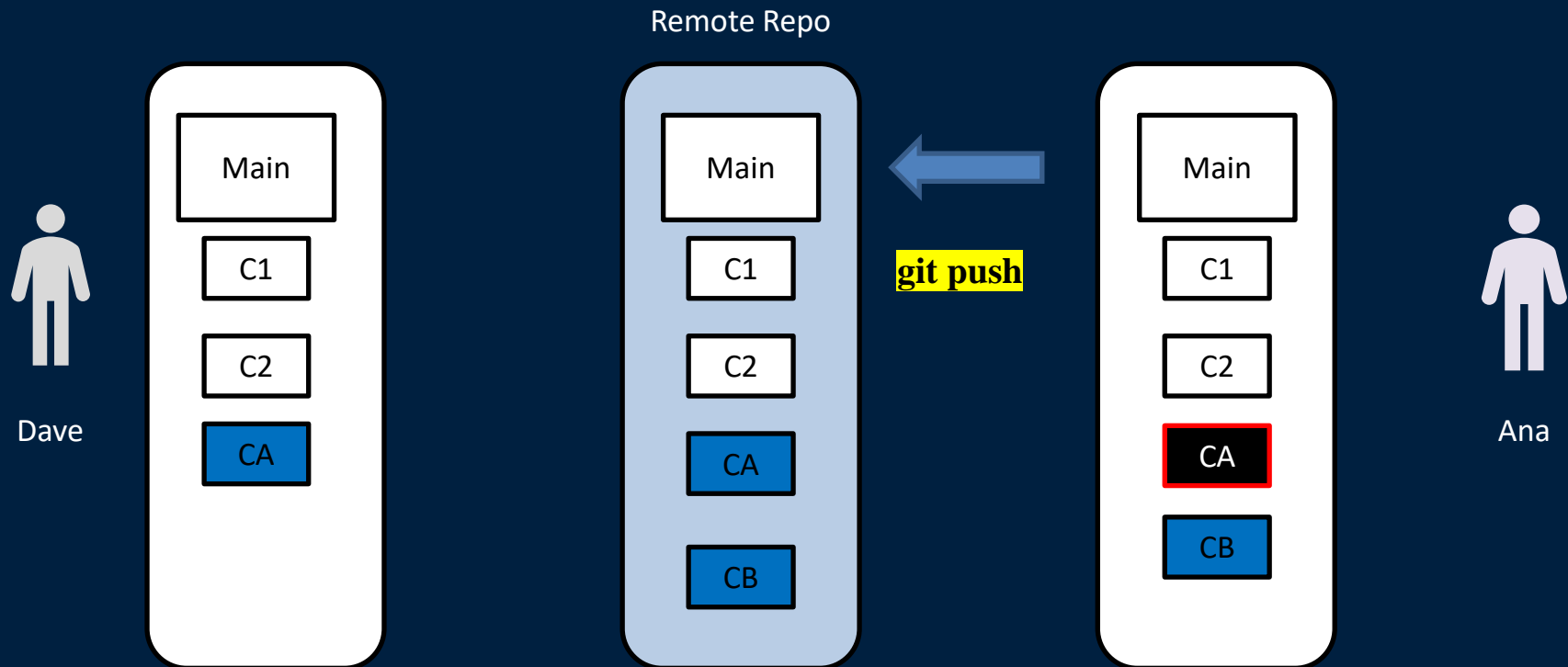
# Collaboration with GitHub



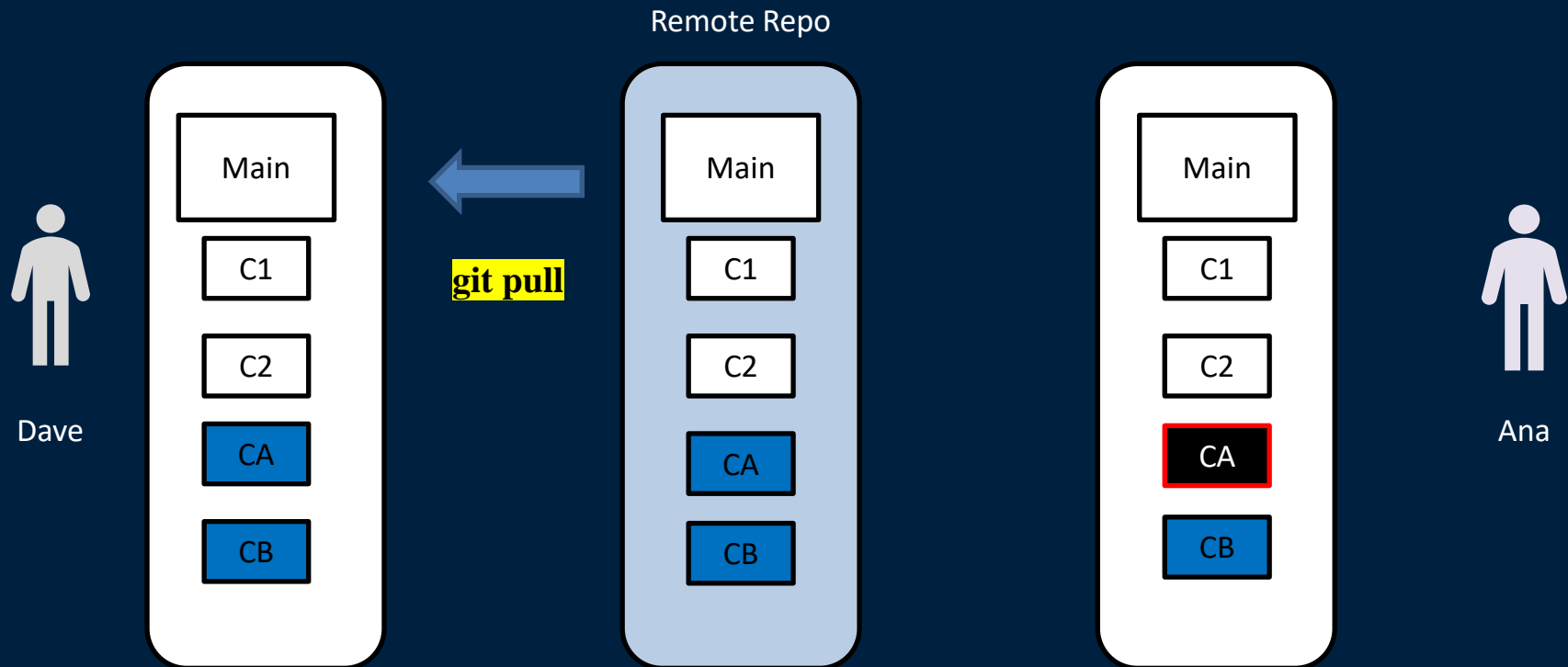
# Collaboration with GitHub



# Collaboration with GitHub



# Collaboration with GitHub



# Identifying Conflicts

---

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

To do this you can do a pull request

Pull request shows information on what files the conflict resides in

```
remote: Counting objects: 16, done.
```

```
[...]
```

```
6385a287..52d5a master -> origin/master
```

```
Auto-merging <file>
```

```
CONFLICT (content): Merge conflict in <file>
```

```
Automatic merge failed; fix conflicts and then commit the result.
```



# Resolving Conflicts

---

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

You can also use `mergetool` in command line.

```
git mergetool
```

It will open files that has a conflict and let you choose a solution.

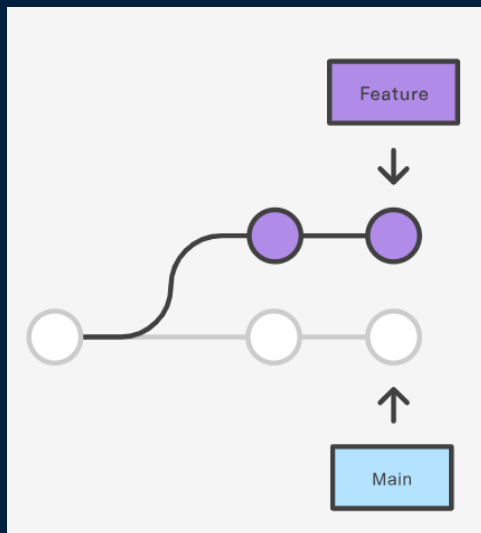
# Key Concepts

## Repositories

- The act of copying a repository from a remote server is called `cloning`
- Cloning from a remote server allows teams to work together

## Branches

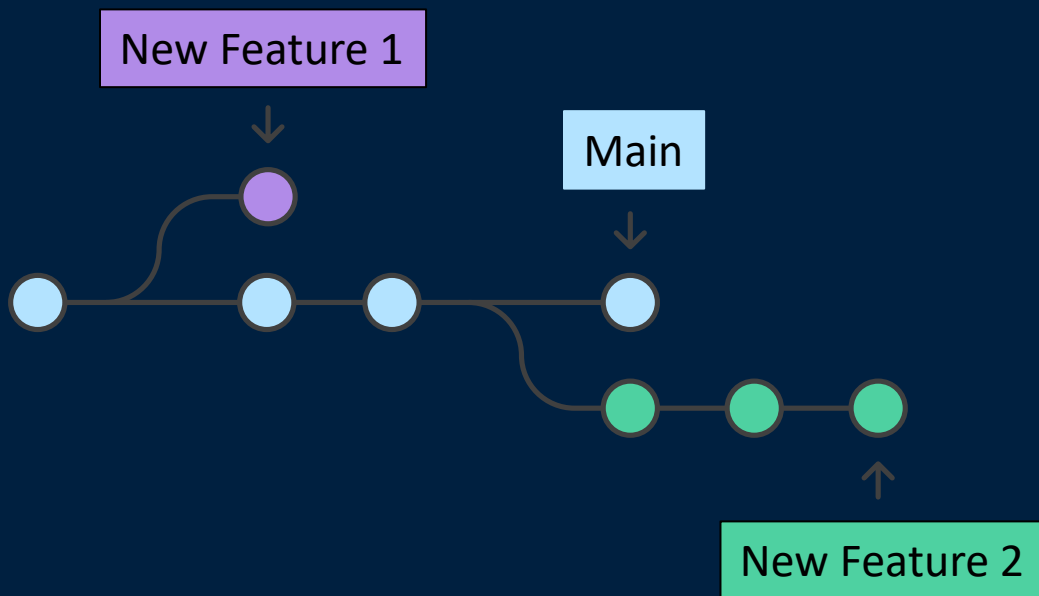
- All commits in git live on some branch
- But there can be many, many branches
- The main branch in a project is called the `main` branch



# 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

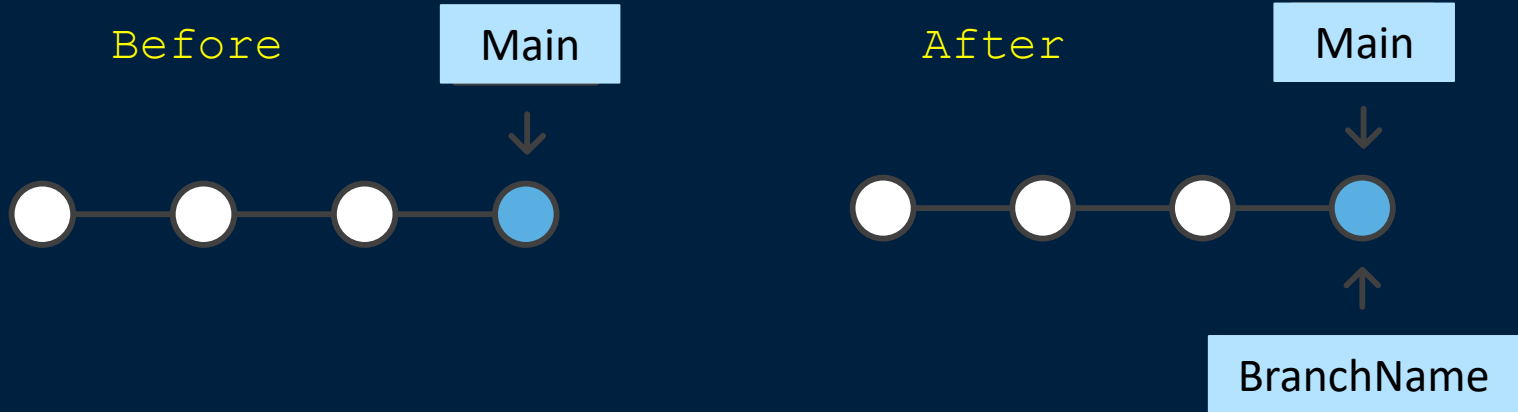


# Git Branch

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.

# Try it: Branching with GitHub

---

## Tasks

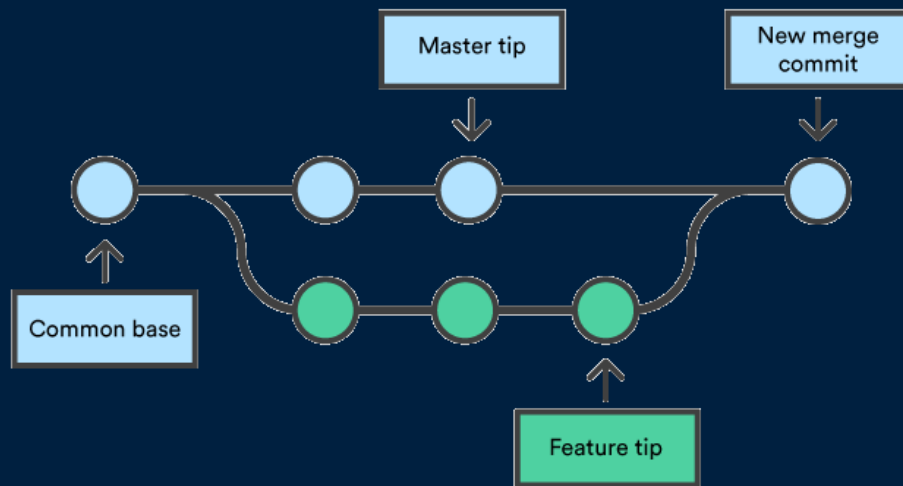
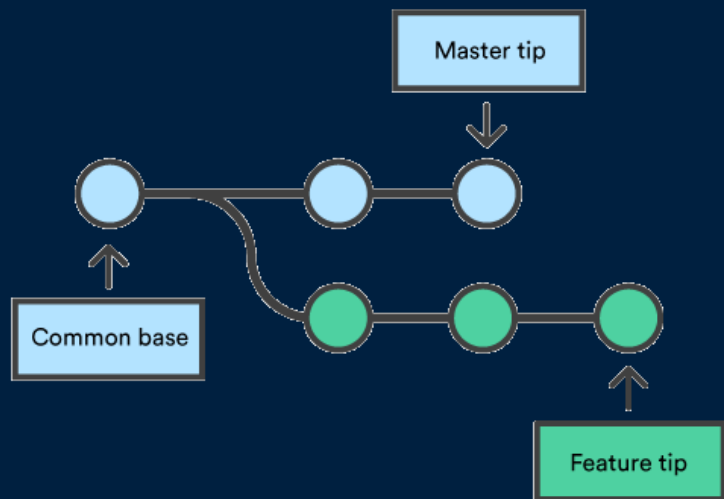
- Create a new repository in your GitHub account
- Clone the repository on your local machine
- Add a text file (e.g., touch master.txt) to the local repository
- Create a new branch and then checkout to the new branch
- Add another text file (e.g., touch branch.txt) to the new branch and commit the changes
- Navigate back to the main branch and merge the new branch back to the main



# 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`



# 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
```

# Push to remote

---

Push any commits you have made locally on the `main` branch to the remote repository on `origin`

```
git push
```

```
git push origin main
```

Push any commits you have made locally on the `BranchName` branch to the remote repository on `origin`

```
git push origin BranchName
```

For branch, first time, you may need to create upstream reference

```
git push --set-upstream origin BranchName
```

# Deleting Branches – Remote repos

---

To delete a remote branch execute the following.

```
git push origin --delete BranchName
```

# 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 Branch Question

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

- 1) `git branch NewBranch` creates a new branch
- 2) `git branch NewBranch` does not check out to the `NewBranch`
- 3) `git branch` is tightly integrated with the `git checkout` and `git merge` commands
- 4) `git branch -d NewBranch` can be used to force delete the branch called `NewBranch`

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

# Objectives

---

- Set up a Git repository
- Perform repository-to-repository collaboration
- Resolve Git conflicts
- Create, check out and merge branches
- Git branching models



THE UNIVERSITY OF BRITISH COLUMBIA

