# Git & Github

Sam Ahmadizadeh - CSSA IUST

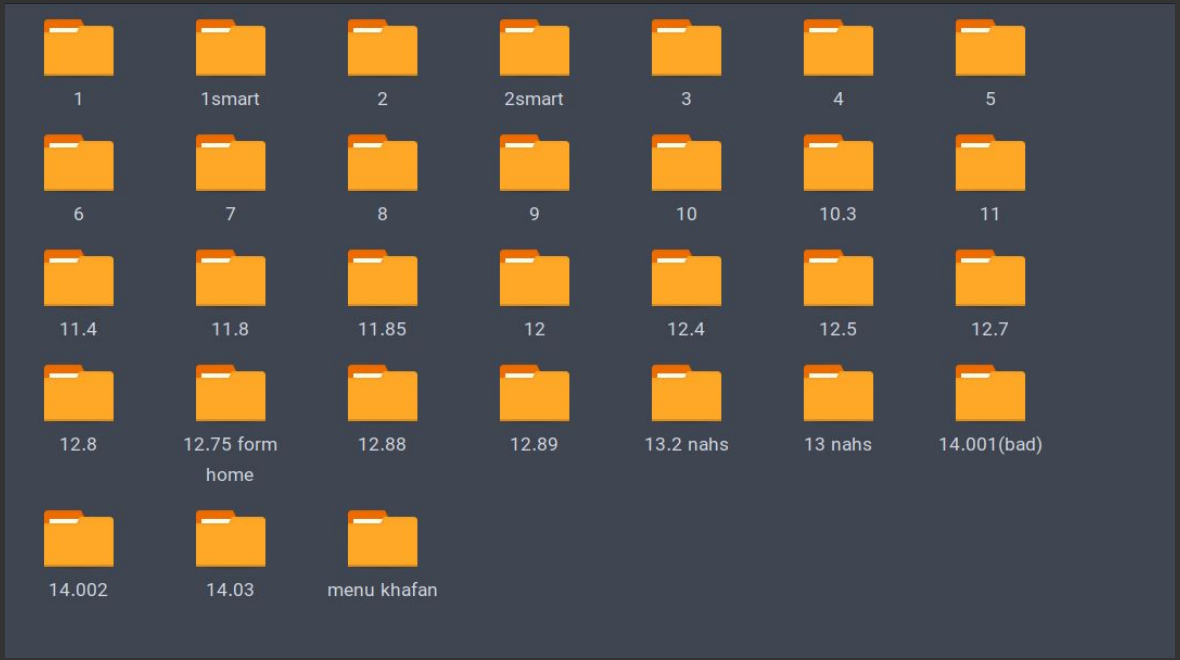
# What is git?

Git (/git/)[7] is a distributed version-control system for tracking changes in source code during software development.

# What is a version control?

Version control, also known as source control, is the practice of tracking and managing changes to software code.

# This is also a kind of version control



# What are the disadvantages of the previous slide approach

- Copying same things over and over
- Difficulty in Collaboration
- No Change History
- Difficulty Tracking Bugs and Features
- And more

# Advantages of using git

- Making collaboration easier on a project
- Developing Features and fixing bugs parallel
- Reverting changes easily
- Debugging

## Do we only have git as version control?

Surely NO!

## Other version controls

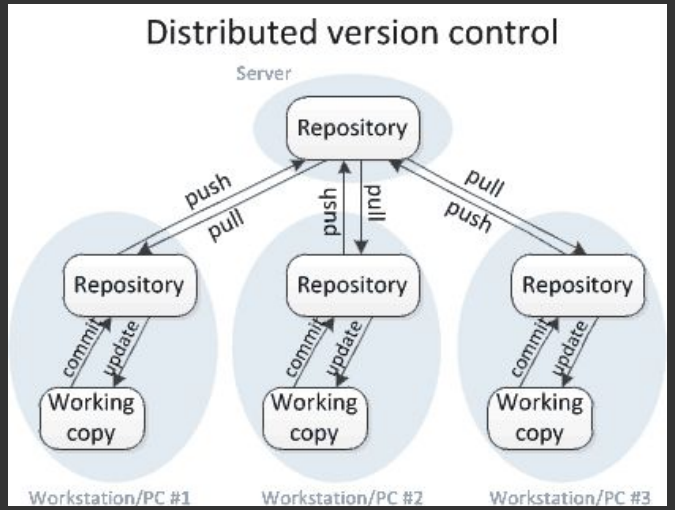
## Then why git?

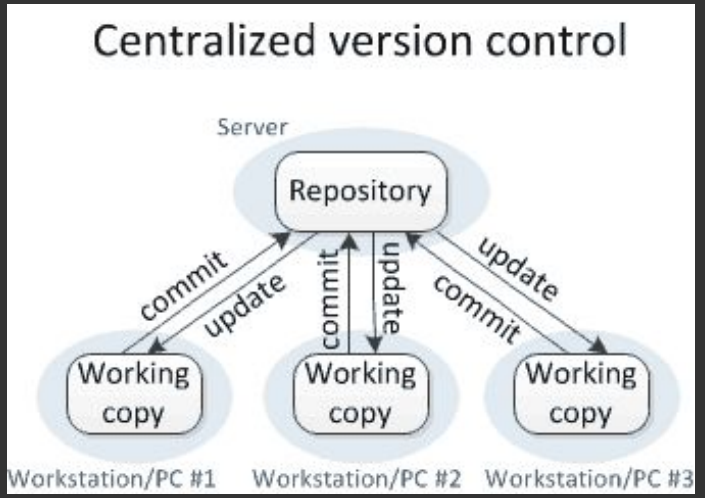
- Distributed
- Offline Work
- Branching and Merging
- Performance

## Then why git?

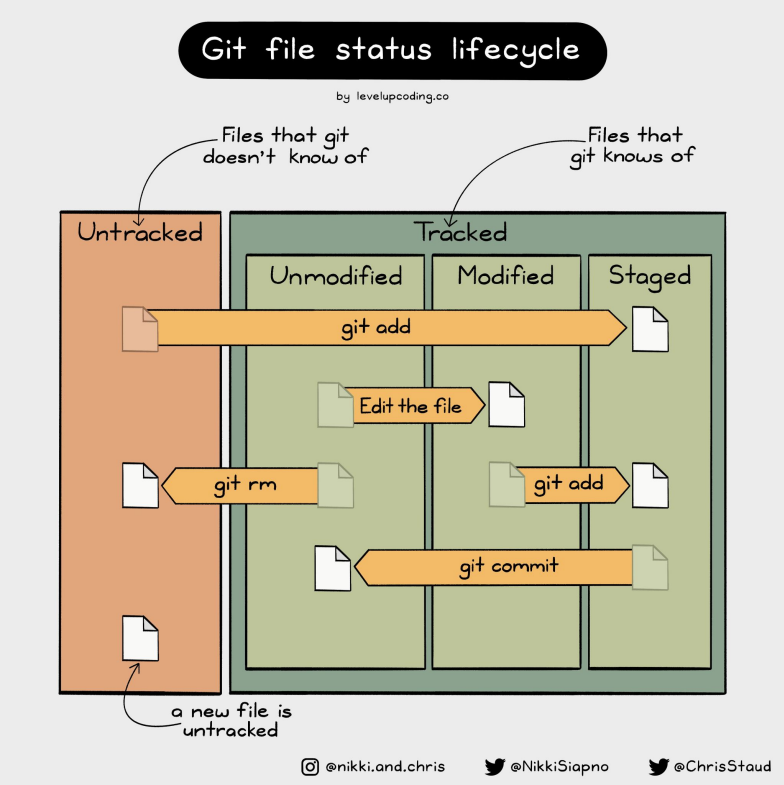
- Distributed
- Offline Work
- Branching and Merging
- Performance

# Centralized vs Distributed



# File life cycle in git



## Let’s use git!

- First of all we have to install git
  - https://git-scm.com/downloads/win

# Setting up the environment

```
-> git config --global user.name "Your Name"
-> git config --global user.email "your.email@example.com"
```

```
-> git config user.name "Your Name"
-> git config user.email "your.email@example.com"
```

# Starting our very first Repository

When you use the first command it will create a repo on your local machine

```
-> git init
-> git status
```

# Tracking and committing our first file

For adding a new file to your git you use the commands below:

```
-> git add <File name>
-> git commit -m “msg you want to represent your commit”
```

- “-m” flag stands for the commit message

# How to add all the files to staging area?

By using “.”

```
-> git add .
-> git commit -m “msg you want to represent your commit”
```

- “-m” flag stands for the commit message

# How to add all the files to staging area?

By using “.”

```
-> git add .
-> git commit -m “msg you want to represent your commit”
```

You can also use regex patterns for staging

```
-> git add *.txt
-> git commit -m “msg you want to represent your commit”
```

- *.txt basically means all the files that their name ends with “.txt”

# How to add all the files to staging area?

By using “.”

```
-> git add .
-> git commit -m “msg you want to represent your commit”
```

You can also use regex patterns for staging

```
-> git add *.txt
-> git commit -m “msg you want to represent your commit”
```

- *.txt basically means all the files that their name ends with “.txt”

# Let’s practice!

- Download the book.txt file

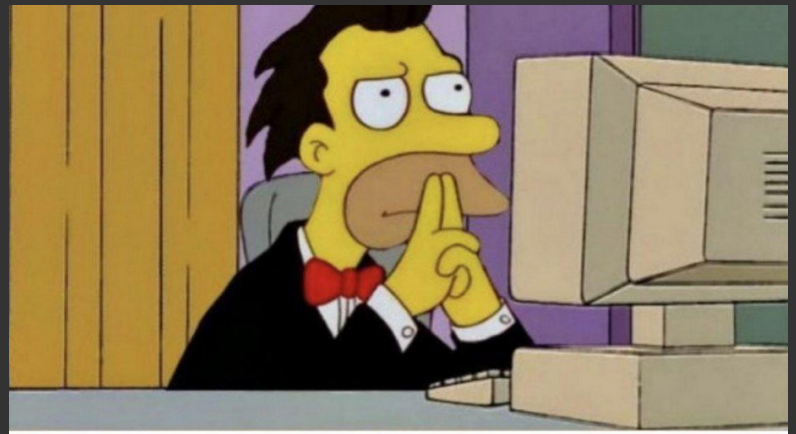
- Make the file into 4 different files:
  - `Intro.txt : First line from chapter 1`
  - `Chapter1.txt : Chapter 1 upto chapter 2`
  - `Chapter2.txt : Chapter 2 upto chapter 3`
  - `Chapter3.txt : Chapter 3 upto the end`

- Stage the files and commit them with message : `"extracted the intro and first three chapters"`

# How to add all the files to staging area?

File name can be used in commit command to individually commit the staged files

```
-> git add .
-> git commit <File name> -m “msg you want to represent your commit”
```



```
% git add .
% git commit -m '
```

# Some tricks about commit message

- **Follow the Conventional Format**

sql

Copy code

```
feat: add user authentication

Implement login and registration functionality using JWT. This feature
includes secure password hashing and error handling.
```

# Some tricks about commit message

- **Follow the Conventional Format**
  - **feat: A new feature**
  - **fix: A bug fix**
  - **docs: Documentation changes**
  - **style: Code formatting, no logic change**
  - **refactor: Code refactoring, no new features or bug fixes**

# Inside of git

- Is Git Running in the Background?
- What if We Change a File Twice?
- What Does Tracking Mean?
- What if I Uninstall Git?
- What About Deleting .git?

# What if i don’t want to include a file in git?

- **The answer is using .gitignore**
  - **Create a file in git directory named .gitignore**
  - **Write the pattern ot name of the files that you want to ignore**
  - **Commit your changes on .gitignore !**


node_modules
git
.gitignore
git

# Let’s use gitignore

- **Gitignore**

# Branching in git

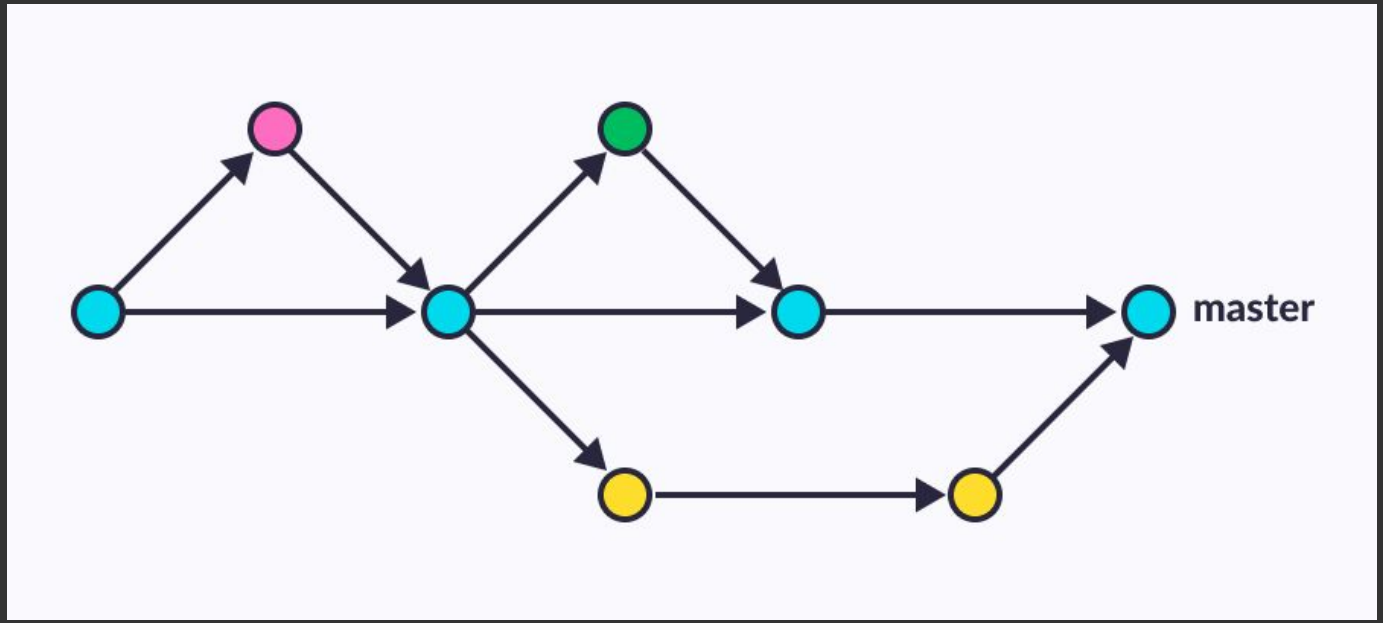
Think of a scenario where you and your co-workers are working on a code. Also your working on the different parts of the project

What are the problems that may happened?

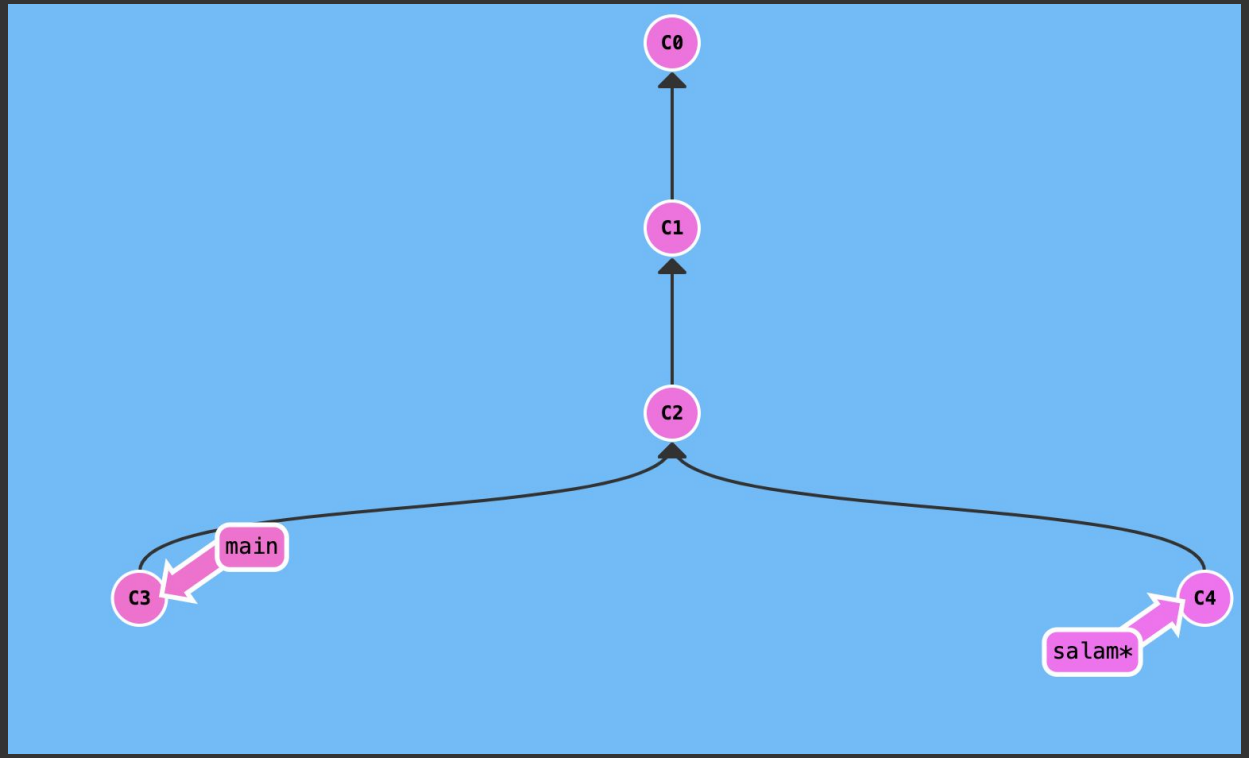
1. You don't have the changes that your colleagues have applied
    a. Both have to commit on a same repo then which should git save?
2. If a bug happens in a commit then you should delete all the changes not the feature that is creating the bug.

# Branching in git

Branching can solve all of these problems.



# Git visualizer



# Branch and Checkout

For creating a new branch you can use the command below:

```
-> git branch <branch name>
```

- But this command won't take the HEAD on the branch then you should use the command below:

```
-> git checkout <branch name>
```

- Use -b flag to create the branch and get the HEAD on it:

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

# Naming branches

- feature/: For new features or enhancements
- bugfix/: For fixing bugs
- hotfix/: For urgent fixes to production
- chore/: For maintenance or minor improvements (e.g., chore/update-dependencies)
- release/: For release branches (e.g., release/v2.0)

Example:

- `feature/user-authentication`
- `bugfix/fix-login-issue`
- `hotfix/crash-on-startup`

# Avoid this namings:

```
-> git branch mammad
```

```
-> git branch asghar
```

```
-> git branch code-ke-reza-zade
```

# Let’s practice!

Let’s help mammad and reza to contribute in a project that they are developing with each other.

- Create each file and commit them on a branch that is named in this pattern:

feature/<file name>

- By this naming you should have 4 different branches

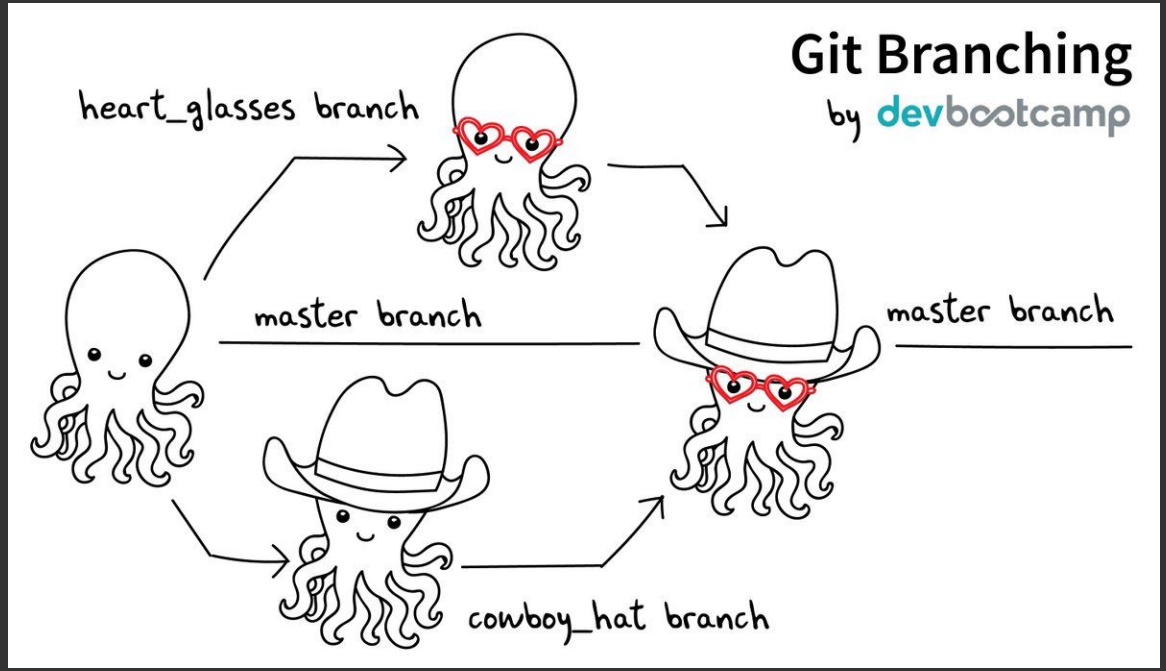
# Merging in git

We use merge in git to merge some files in a particular branch to another one:

```
-> git merge <branch name>
```

- We use this command to merge `<branch name>` to the branch that the HEAD is on.

# Merge in git




- By default git repositories have a main branch named (Master/main)

# Let’s practice!

Now on, Mammad and reza want to merge the branches that they have created. Help them in this progress with the new thing that you have learned:

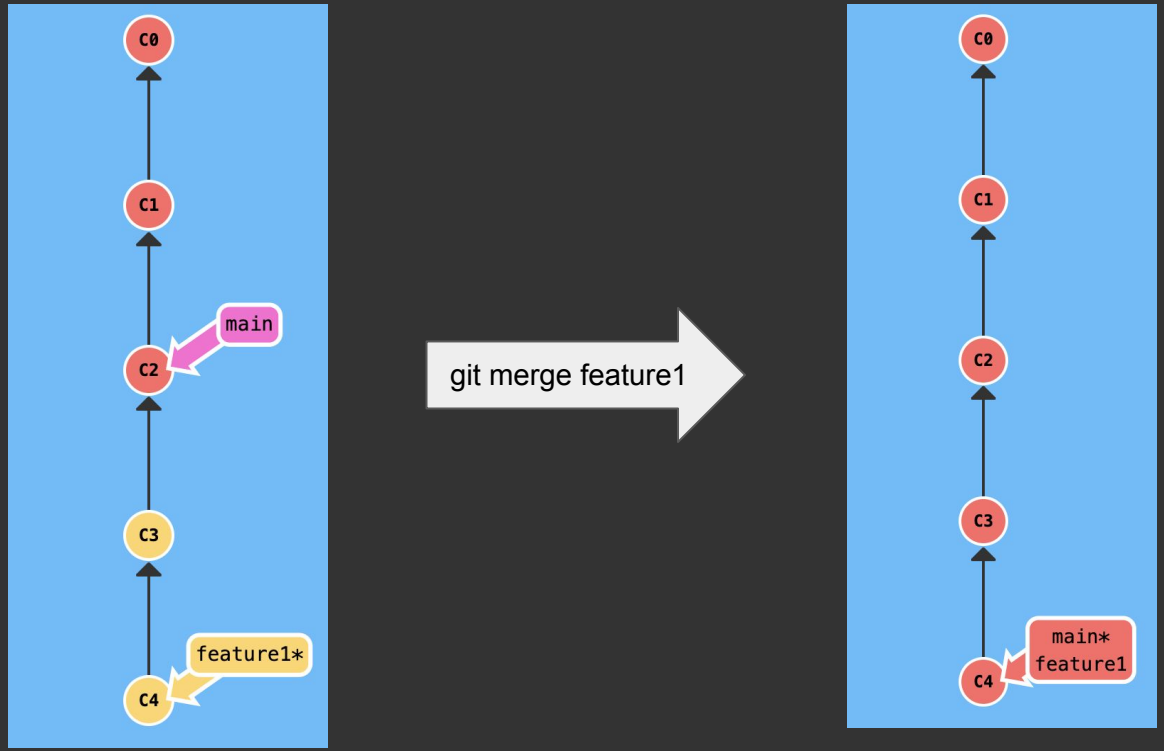
- Merge all the files in the main branch

# Kinds of merging

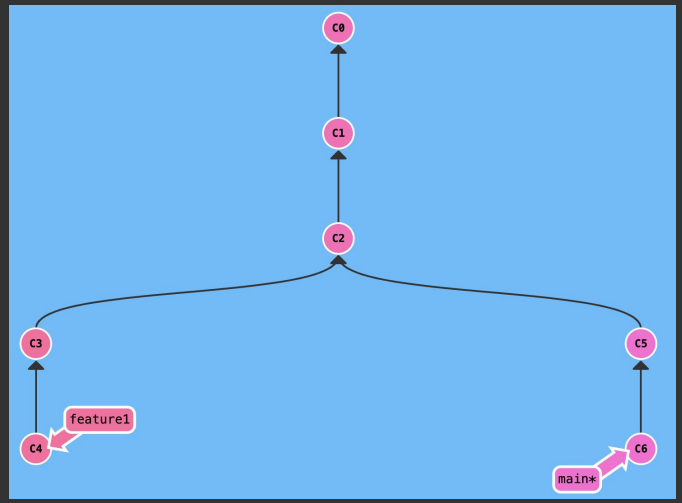
In git there are 2 kinds of merging:

1. Fast Forward merging
2. 3-way merging

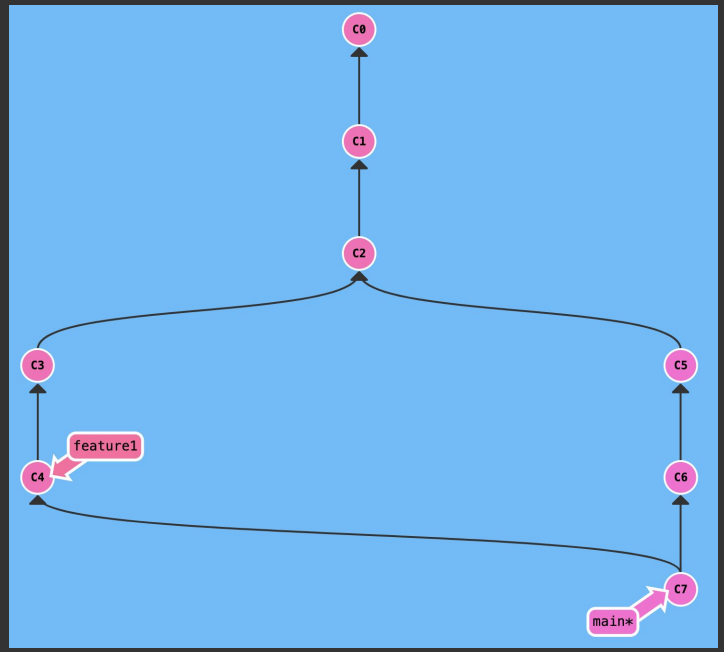
# Fast Forward merge



# 3-way merge




git merge feature1



# Difference between them

- When there is a commit after creating a branch from the parent commit, git has to check there different commits so it can merge the files.For this purpose its named 3-way branch

- On the other hand when there is no commit after creating the branch there is nothing to compare then git will fast forward it.

# Quiz

Which kind of merging will happened is this scenario ?

# Conflict

When a file(files) is retrieved in two different branches:

- Which one should git keep

In this case, Something happens that is named conflict.

Lets see an example of conflict.

# Example

Let’s make a situation that can happen a conflict

# Time machine in conflict!

```
git merge --abort
```

```
git diff <branch-name>
```

# Solving conflict

For solving conflict we have different ways:

1. Keeping the current branch modifications
2. Keeping the other branch modifications
3. Combining both of  them

```
-> git checkout --ours /path/to/file
```

```
-> git checkout --theirs /path/to/file
```

# Rebase

replays the changes from your current branch into another base branch. Instead of creating a new merge commit to combine histories, “git rebase” applies each commit individually on top of the target branch, creating a linear history.

```
-> git checkout feature
```

```
-> git rebase main
```

# Error handling in projects using git

When an error occurs in a project it’s time to use the power of git.

First of all let’s have overview on some basic commands of this section.

# Seeing the commit history

By using the command below you can check the commit history and the commits IDs:

```
git log
```

# Making git not to track a file

The command below can be used to remove a file from out directory and repo at the same time:

```
git rm <file-name>
```

To remove a file only from the Repo (just untracking) we use:

```
git rm --cached <file-name>
```

# Making git not to track a file

The command below can be used to remove a file from out directory and repo at the same time:

```
git rm <file-name>
```

To remove a file only from the Repo (just untracking) we use:

```
git rm --cached <file-name>
```

# Ignore the changes!

To ignore the changes after the commit, We can use the commands below:

```
git restore <File-name>
```

If you accidently added a file to the staging state, Use the command below:

```
git restore --staged <File-name>
```

# Ignore the changes!

To ignore the changes after the commit, We can use the commands below:

```
git restore <File-name>
```

If you accidently added a file to the staging state, Use the command below:

```
git restore --staged <File-name>
```

You can also use restore to restore a file(s) from a commit:

```
git restore --source=<commit-id> <File-name>
```

# Git HEAD

The Head can be moved by the command below:

```
git checkout <commit-id>
```

# Quiz!

Then what is the difference between changing the HEAD location and restoring a file(s)?

# It's time to accept the mistakes

After firing the one who made a change in code that created a bug, you have to revert the changes.

For this purpose we use the command below:

```
git revert <commit-id>
```

- For going back to more than 1 commit do this process commit by commit

# It's time to accept the mistakes

After firing the one who made a change in code that created a bug, you have to revert the changes.

For this purpose we use the command below:

```
git revert <commit-id>
```

- For going back to more than 1 commit do this process commit by commit
- Attention: THIS COMMAND ONLY REVERTS THE CHANGES IN THE COMMIT NOT THE OTHER ONES

# It's time to accept the mistakes

If you want to undo all the mistakes and changes and get back to a specific commit you can use reset command:

```
git reset <commit-id>
```

- All the files will get back to that commit states
- You can change the behavior by using tags with this command

# Reset command flags

Example for flags:

```
git reset -hard <commit-id>
```

## Comparison Table

| Tag | Commit History | Staging Area | Working Directory |
|---|---|---|---|
| `--soft` | Rewind | Kept | Kept |
| `--mixed` | Rewind | Cleared | Kept |
| `--hard` | Rewind | Cleared | Cleared |

# Visualizing It with a Document Example

- --soft:
  Undo the save, but keep the text in your clipboard (staged).
- --mixed:
  Undo the save, but leave the text on the page (working directory) for editing.
- --hard:
  Undo the save and erase all your text (discard changes completely).