GIT

OVERVIEW

Session 1

- What is version control?
- Why do we need version control?
- Version control basics
- Exercises with the basics
- How does it work beneath the surface?
- Final exercise

Session 2

- Ignore
- Stash / pop
- Rebase
- Rollback
- Merge conflicts
- Final exercise
- What else is possible… CI/CD

INTRODUCTION ROUND – SPEED DATE STYLE (30 SEC)

- Who are you?
- Background / position
- Ambitions
- Hobbies
- Something your colleagues don't know yet
- What do you want to learn?

WHAT IS SOURCE CONTROL MANAGEMENT?

- Keeps track of all the previous versions of the code
- Enables going back to previous versions of the code
- Helps merging different versions of the code

WHY DO WE NEED SOURCE CONTROL MANAGEMENT?

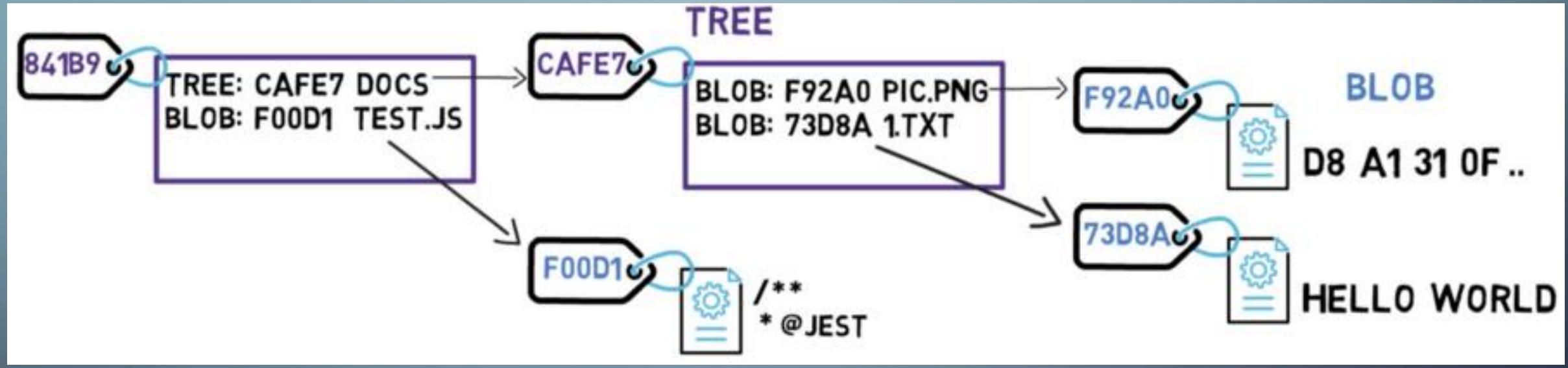
- Developers are working together on the same code base and will be editing the same files at the same time
- SCM avoids overriding each others work and breaking the software
- Whenever issues arise, you can easily go back to an older version that was stable

GIT

- Git is a version control system that tracks changes in computer files. It can coordinate changes to these files while multiple people are working on it. It is mainly used for source code files, but could be used for any set of files.

GIT INTERNALS - I

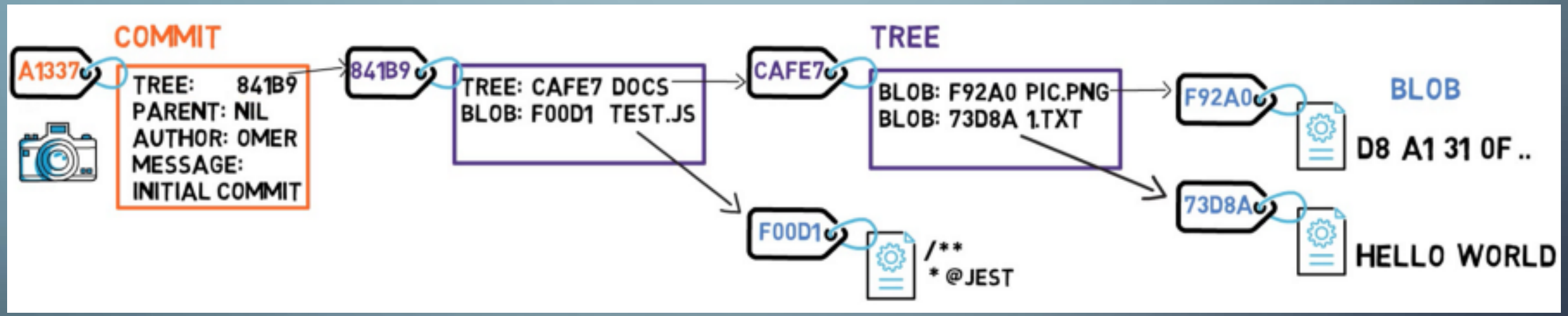
- Git can be thought of as a file system, it begins with a certain root directory
- Content of files are stored in blobs
- Blobs all get their own unique SHA-1 hash
- In git, directories can be thought of as trees: it refers to blobs and other trees (directories)



HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

GIT INTERNALS - II

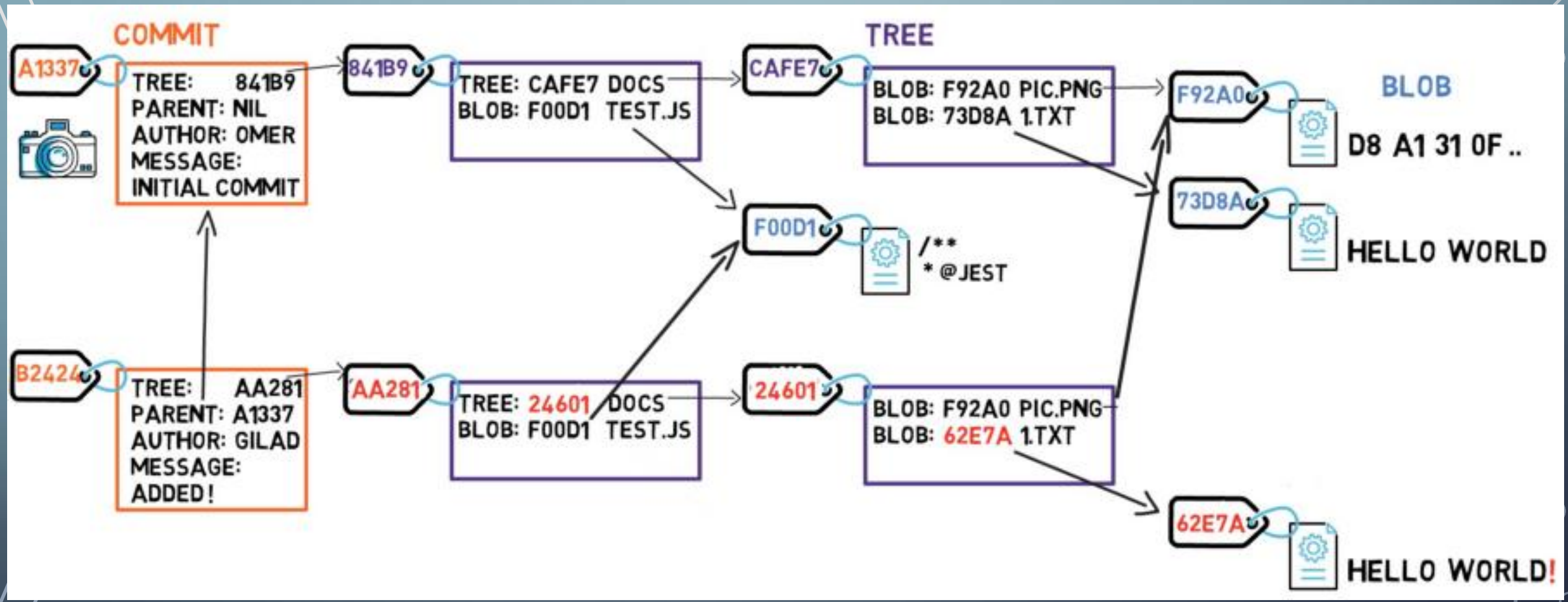
- Snapshot of the file system and all the files with their content = commit (so not just the delta)
- Commit points to the “root directory” and meta data of commit (who committed it, commit message and timestamp)
- Commits also point to parent commit (if not the first commit)



HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

GIT INTERNALS - III

- So how is it possible that this doesn't take huge amounts of data storage?
- Only the changed files get a new hash, and the commit object is usually pointing to a lot of the same blobs as the previous commits



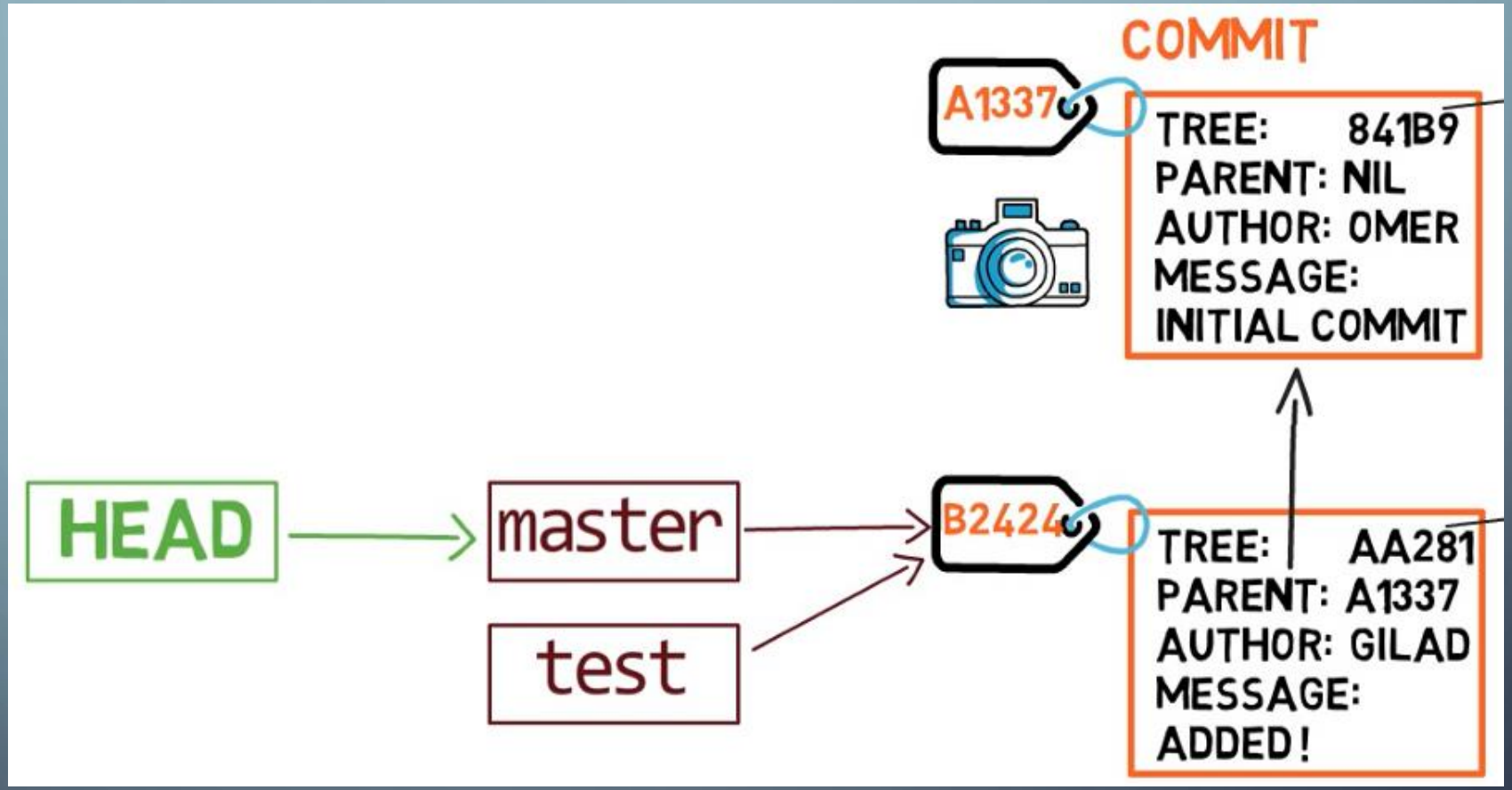
HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

GIT INTERNALS – OBJECT OVERVIEW

- Blob
- Tree
- Commit

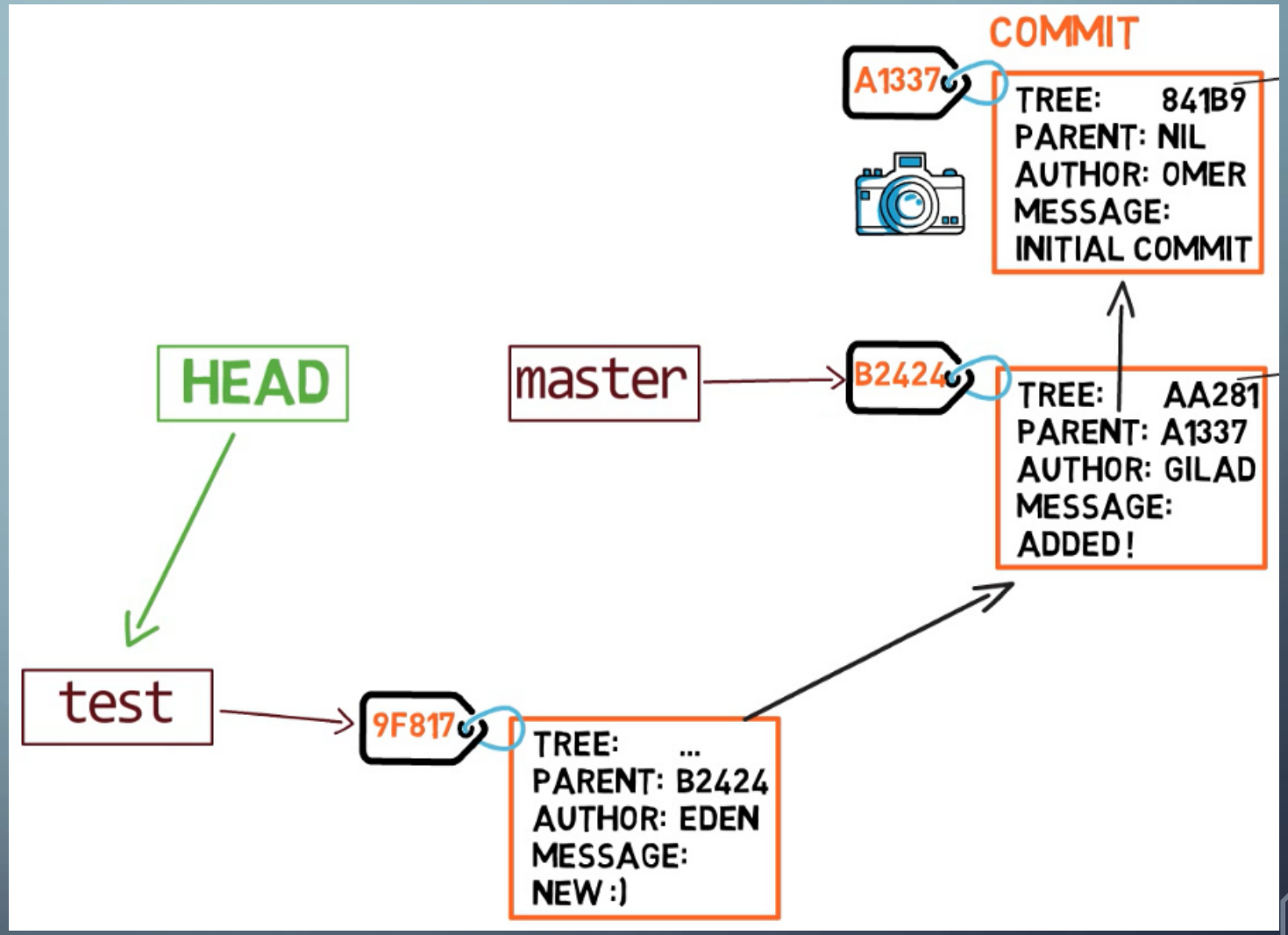
GIT INTERNALS - IV

- Branches are just pointers with a special name to a certain commit
- Git tracks the branch we are on using a special pointer “HEAD”
- HEAD points to a branch and the branch points to commit



HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

So if we make some changes and commit, this only gets added to the selected branch, like this:



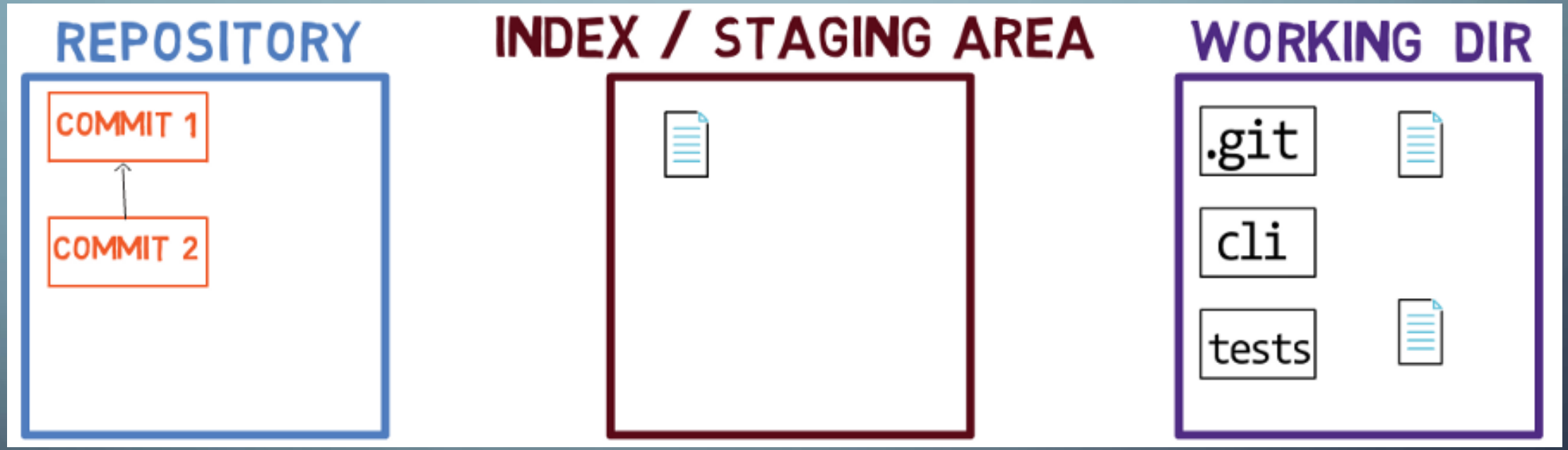
HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

GIT INTERNALS - V

- We have a copy of our source code in our working directory (local)
- This working directory is associated with a repository
- Repository is a set of commits (=snapshots of the file system) + branches and HEAD
- Working directory contains a .git file
- Changes we make in the working directory are captured in the remote repository

GIT INTERNALS - VI

- Git does not commit changes from the working dir to the repository immediately
- There is a inbetween area where changes are kept: staging area (also called index)
- Commits get created based on the staging area
- Changes have to be added to the staging area manually



HTTPS://WWW.FREECODECAMP.ORG/NEWS/GIT-INTERNALS-OBJECTS-BRANCHES-CREATE-REPO/

GIT INTERNALS - VII

- Files in the working tree can be tracked and untracked.
- Tracked: git is aware of these files and they were either in the last commit already or they are in staging area
- Untracked: everything that is not in last commit or staging area

SVN VS GIT

- SVN: centralized vs Git: distributed
- SVN: options to checkout sub-trees vs Git: entire repo or nothing
- SVN: deltas vs Git: blobs/hashes
- Git branch not automatically available to other people, because distributed and not centralized
- Git preserves branching history
- SVN branch = code copy, Git branch = named pointer

GIT FUNDAMENTALS

- Clone
- Fetch
- Pull
- Checkout
- Branch
- Add
- Status
- Commit
- Push
- Merge

REPOSITORIES (REPO)

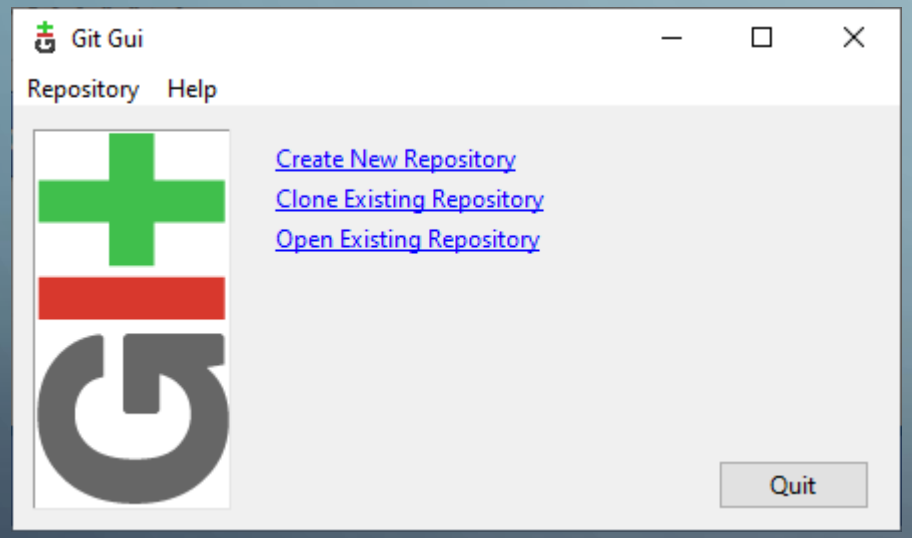
- Data structure that contains meta data and a set of files
- In this case we will use repo's to store our code projects

GIT FUNDAMENTALS I

- Clone
- Fetch
- Pull
- Checkout
- Branch
- Add
- Status
- Commit
- Push
- Merge

CREATE A NEW GIT RESPOSITORY

Command: git init



Mini exercise:

- Create a new git repo in two ways
  - Using the command line
  - With the GUI

CLONE

- Cloning / copying a local or remote repository to a new location
- Automatic connection from clone to original, called origin
- Command: git clone location
- Mini exercise:
  - Go online and find an open source project
  - Clone this project, you can choose to do it with the command line or the GUI

STATUS

- Inspect repository
- Displays the state of the working directory and whatever is being staged
- Command: git status

Mini exercise:

- Do git status
- Change something to the repository
- Check git status again

ADD

- Add change in working directory to staging area (ready for commit)
- Command: git add dir/dir/filename
- Command to add all: git add .

Mini exercise:

- Add previous change to staging area
- Check git status again

COMMIT

- Creates a snapshot of the working directory
- This commit can later be send to the remote repository
- Better: git commit –m “description message”

Mini exercise:

- Commit your changes and add a good message

FETCH

Command: git fetch

- Downloads new data from the remote repository, but it doesn't influence your local current directory
- Updates remote-tracking branches in repository

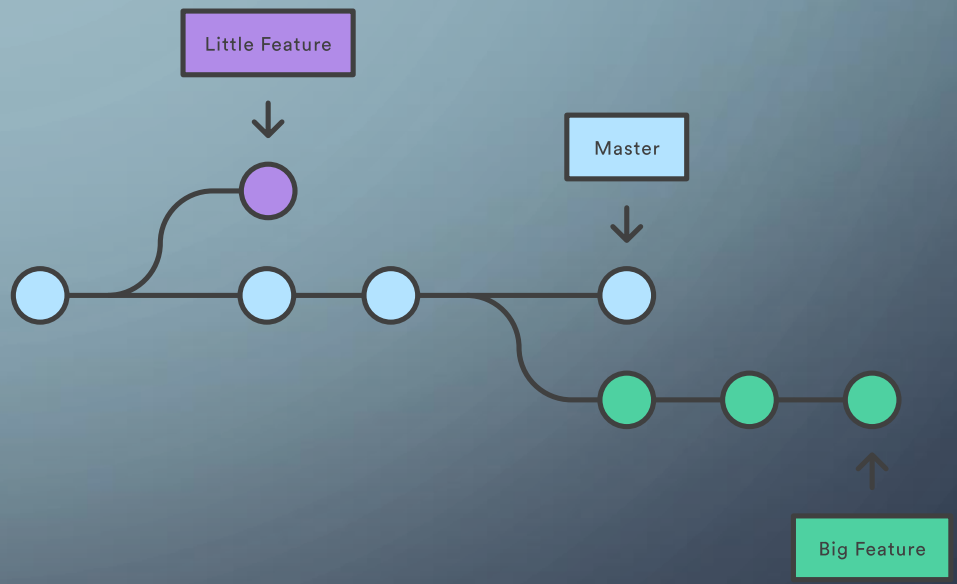
Mini exercise:

- Try git fetch (via GUI or command line)

BRANCH

- Branch is a line of development that diverges from main code line (main branch)

- command: git branch (show all branches)
- command: git branch name (create new branch)
- git branch –d branchname (delete branch)



CHECKOUT

- Can be used to checkout files, commits and branches
- Command: git checkout branchname
- Checkout and create new branch in one: git checkout –b newbranchname
- Mini exercise:
  - Create a new branch
  - Switch back to the main branch

DIFF

- Used to see the difference between two branches
- Command: git diff <branch-name> <other-branch-name>
- Mini exercise:
  - Create a new branch
  - Make some changes
  - Compare to main

PUSH

- Update remote repository with commit from local repo

- Command: git push (default remote and associated branch)
- More specific: git push remotename branchname (git push origin dev)

Mini exercise:

- Push your code to the remote repo

PULL

- Used to get code from the remote repository and update your local working directory

- Command: git pull remotename branchname (git pull origin master)

Mini exercise:

- Pull the code from a colleagues branch

EXERCISE

- See all existing branches
- Add a new one
- Add a file
- Get all remote branches
- Checkout a remote branch from your colleague
- Add something
- Get it on the remote

- Bonus: can you delete a remote branch?

END SESSION 1

RECAP

- What is version control?
- Why do we need version control?
- Version control basics
- Exercises with the basics
- How does it work beneath the surface?
- Final exercise

QUESTION:

WHAT IS GIT?

QUESTION:

WHY DO WE NEED GIT?

QUESTION:

WHAT ARE DIFFERENCES BETWEEN GIT AND SVN?

QUESTION:

WHAT STEPS DO YOU NEED TO GET YOUR CHANGED CODE TO THE REMOTE REPOSITORY?

QUESTION:

WHEN DO YOU USE A BRANCH?

OVERVIEW

Session 1

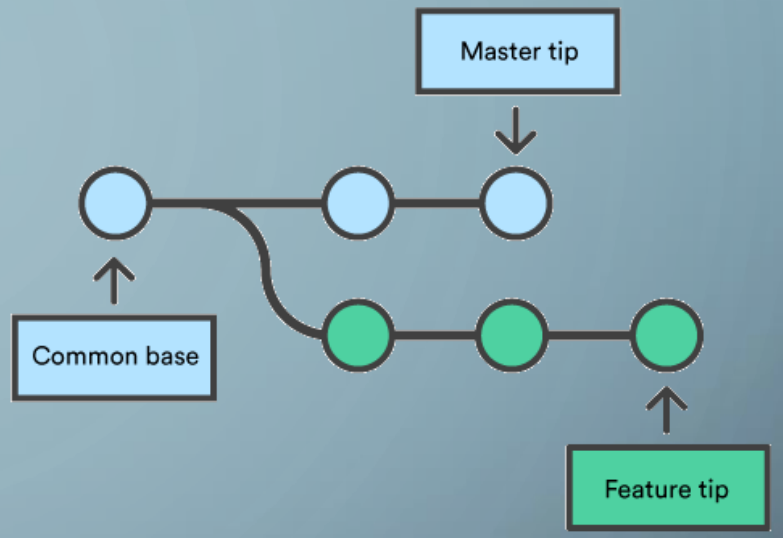
- ~~What is version control?~~
- ~~Why do we need version control?~~
- ~~Version control basics~~
- ~~Exercises with the basics~~
- ~~How does it work beneath the surface?~~
- ~~Final exercise~~

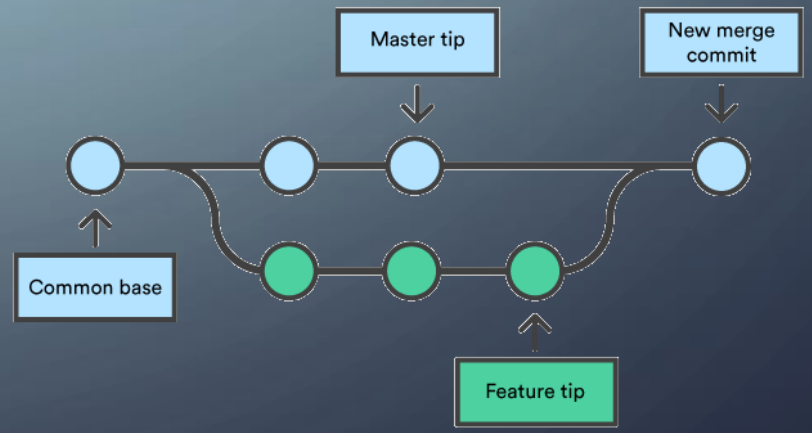
Session 2

- Merge & merge conflicts
- PR + issues
- Ignore
- Stash / pop
- Rebase
- Rollback
- Final exercise
- What else is possible… CI/CD
- Azure DevOps

MERGE

- Combining two branches
- This cannot always be done automatically, then you'll get a merge conflict
- Command: git merge name-of-branch-you-want-to-merge-into-current
- Mini exercise: merge your branch with a colleague's branch





MERGE CONFLICTS

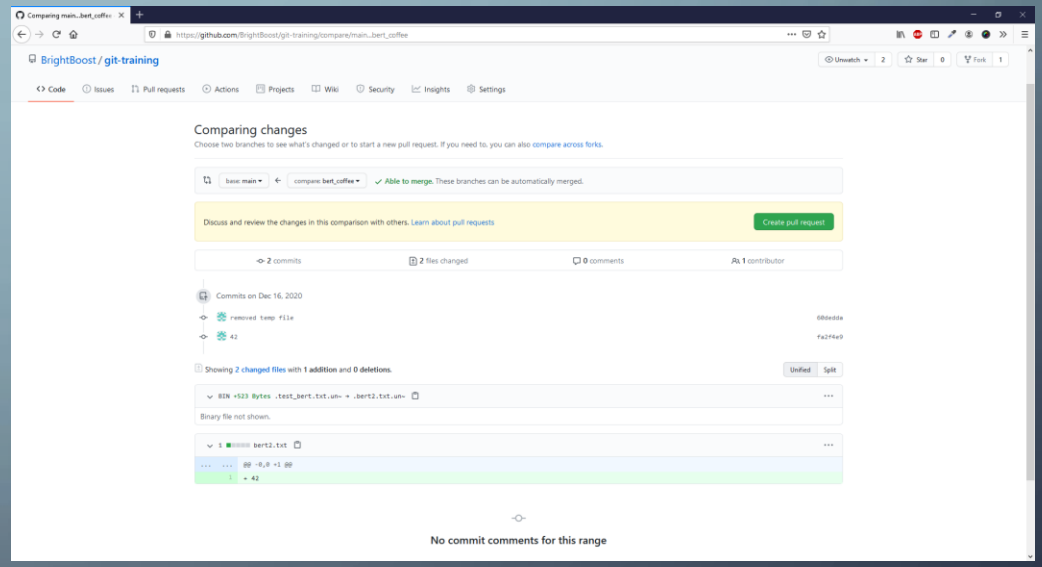
- Happen when git can't figure out how to auto merge
- You'll have to merge it for git:
  - Open file, fix the problem
  - Add fixed file

Mini exercise:

- Create a merge conflict by trying to merge two branches that have changed the same line of a file
- Solve the problem

PULL REQUEST

- Invite to a target repository to take your updates (please pull from my branch)
- Only useful when you're not the only developer
- Great opportunity to discuss and review changes



PULL REQUEST REVIEW

- Great to improve quality of the code
- Developers know what colleagues are working on
- Reviewing is a great learning opportunity for both parties

EXERCISE

- Create a new branch
- Add something new
- Push to remote repo
- Create a pull request and assign a colleague
- Review the pull request assigned to you

IGNORE

- Files that are in the .gitignore won't be tracked by git
- You can ignore complete directories
- You can ignore patterns of file names
- Added files need to be removed first, remove all from tracking: git rm --cached . -r
- Remove one: git rm --cached name.txt

- Add a git ignore file:

touch .gitignore

Examples:

https://github.com/github/gitignore

EXERCISE

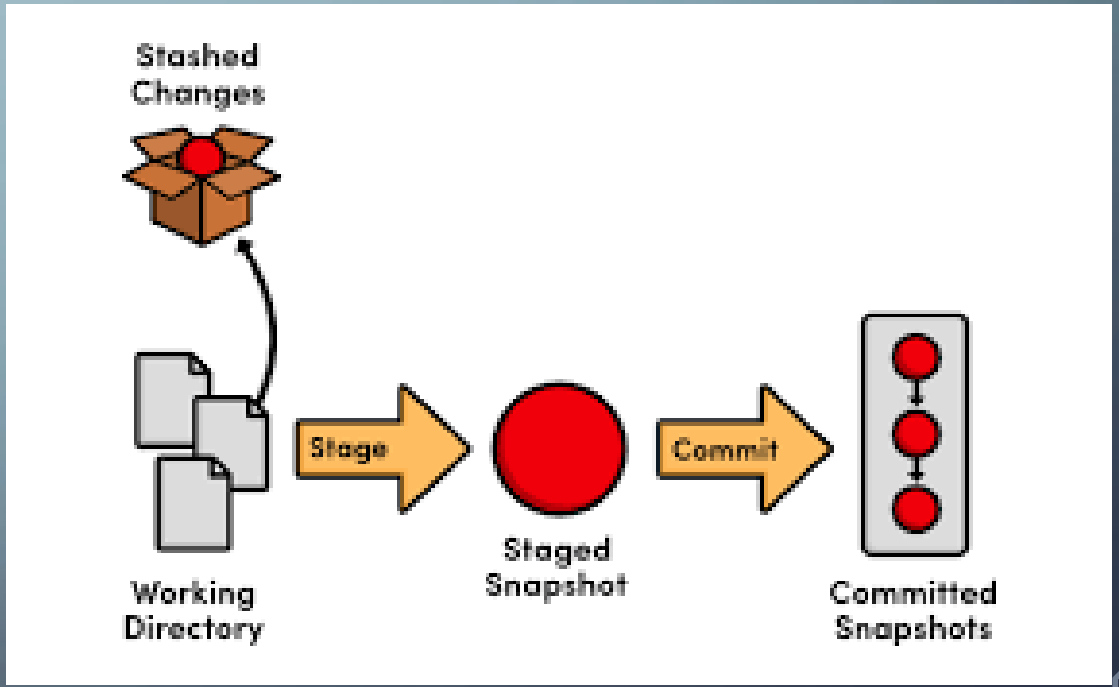
- Add a .gitignore file to the project you initiated earlier with git init
- Add a file to this .gitignore
- See if it gets ignored

- Bonus: Can you add a pattern for ignoring files?

STASH

- Takes uncommitted changes (the ones that are yet added and also the ones that weren't added yet) and shelves them for later
- Ignores new files that aren't added (staged)
- Ignores files in the .gitignore
- Very useful when you want made some changes and need to pull something

- Command: git stash
- Or better: git stash save useful-description



POP (APPLY)

- Unshelves your stashed work
- Command: git stash pop

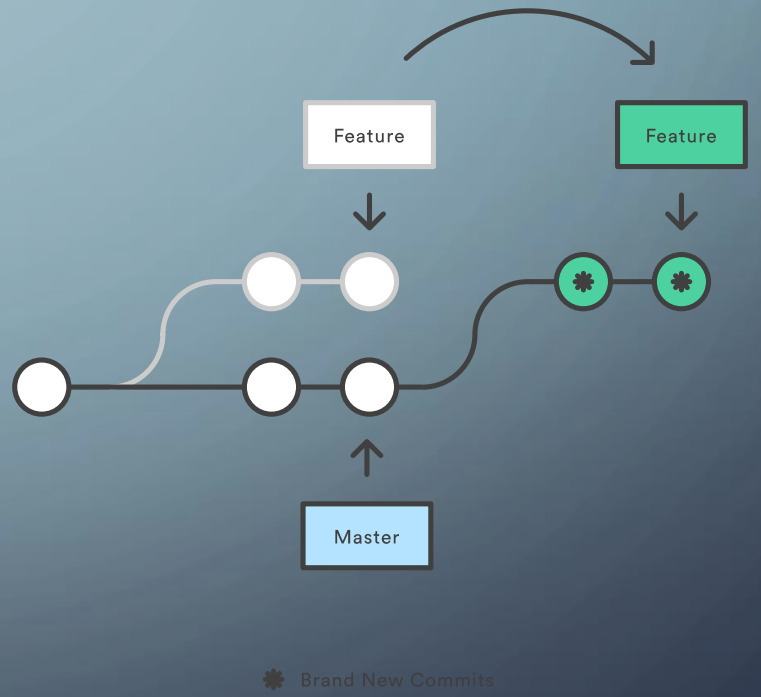
EXERCISE

- Make some changes to your branch
- Perform git status
- Switch to main
- Perform git status: what happened?
- Go back to your branch
- Stash your changes

- Git status
- Switch back to main
- Git status
- What's the difference?
- How to get your changes back if you go to your own branch?

REBASE

- Special way of combining branches
- It removes history of the branch that is being added into the target branch (it places the history on the target branch as if the work was done on the target branch)
- https://www.youtube.com/watch?v=dO9BtPDIHJ8



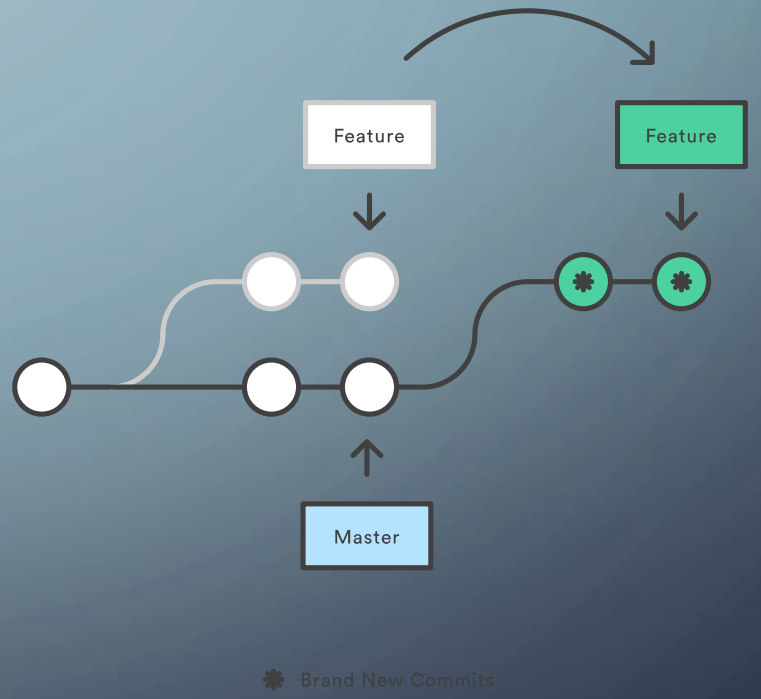
STEPS TO REBASE

git checkout yourbranch

git rebase main

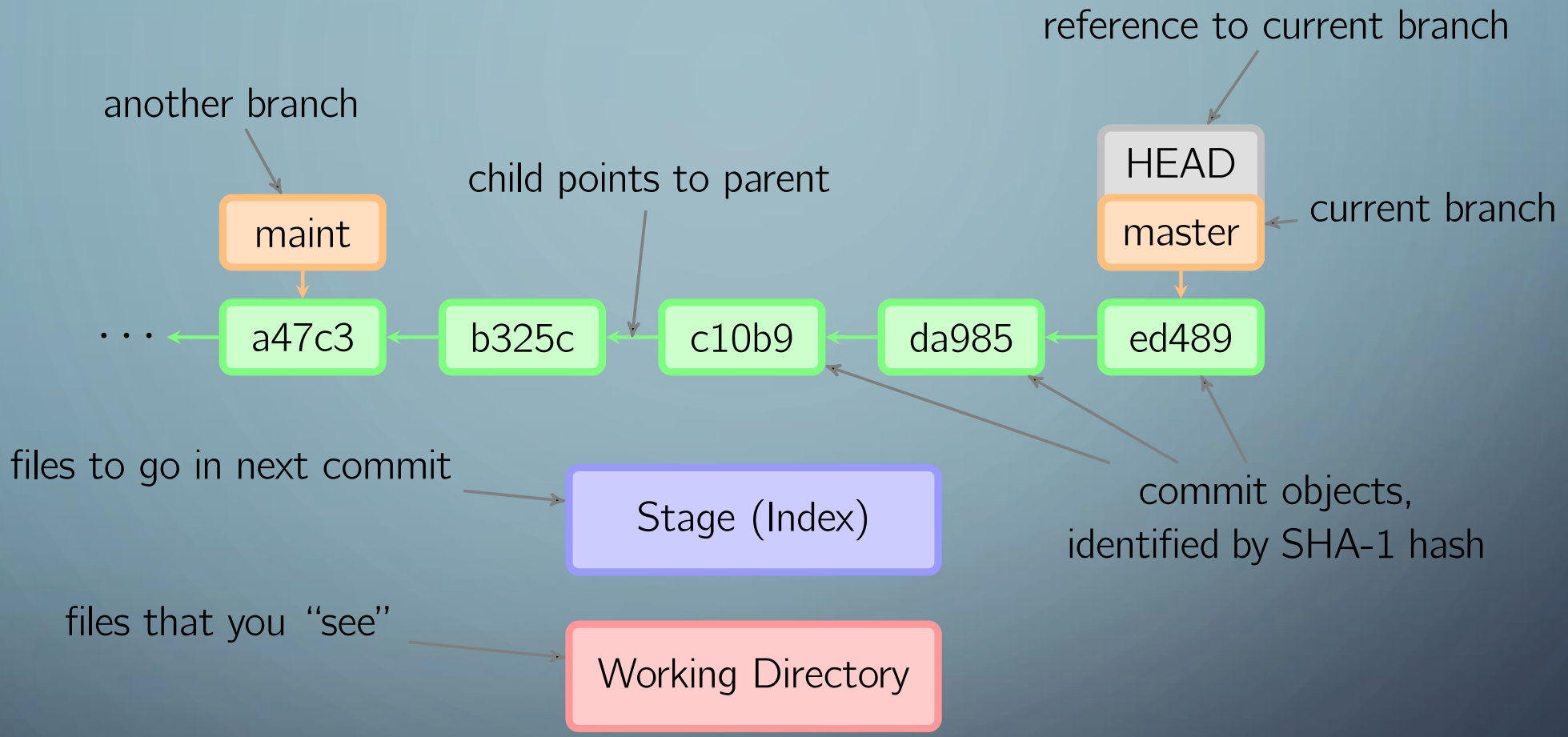
git checkout main

git merge yourbranch



EXERCISE

- Do three commits to your branch
- Use rebase to get these commits into main
- Look at the commit history


reference to current branch
another branch
HEAD
child points to parent
maint
master
current branch
. . .
a47c3
b325c
c10b9
da985
ed489
files to go in next commit
Stage (Index)
commit objects,
identified by SHA-1 hash
files that you "see"
Working Directory

RESET

- Rollback, losing commits
- Options:
  - Hard – reset files of working directory and index
  - Soft – won't alter working directory and index, stages the difference that head moved
- Command: git reset –hard commitcodename
- Command: git reset --hard HEAD^ (going back one commit)

Mini exercise:

- Go back one commit using a hard reset
- Push
- Can you still find the commit?

REVERT

- Rollback, keeping commits and adding undo/invert commit

- Command: git revert commitcodename
- Command: git revert HEAD^ (going back one commit)

Mini exercise:

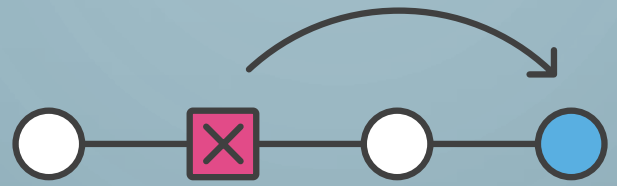
Go back one commit using revert

Push

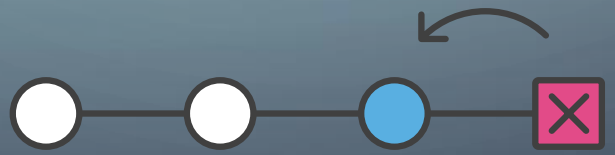
Can you still find the commit?

RESET VS REVERT

Reverting



Resetting



RESET

- Rollback, losing commits
- Options:
  - Hard – reset files of working directory and index
  - Soft – won't alter working directory and index, stages the difference that head moved
- Command: git reset –hard commitcodename
- Command: git reset --hard HEAD^ (going back one commit)

Mini exercise:

- Go back one commit using a hard reset
- Push
- Can you still find the commit?

PUTTING IT ALL TOGETHER EXERCISE

- Add a branch
- Add files
- Make a new file: new.txt with some text on the first line
- Push to remote
- Merge with branch conflict
- Merge with branch of your colleague
- Solve possible merge conflicts
- Add another addition to the file
- Push this addition
- Undo this addition
- Merge or rebase your branch with main

BEST PRACTICES SCM

COMMIT OFTEN

PULL OFTEN

WORK WITH BRANCHES

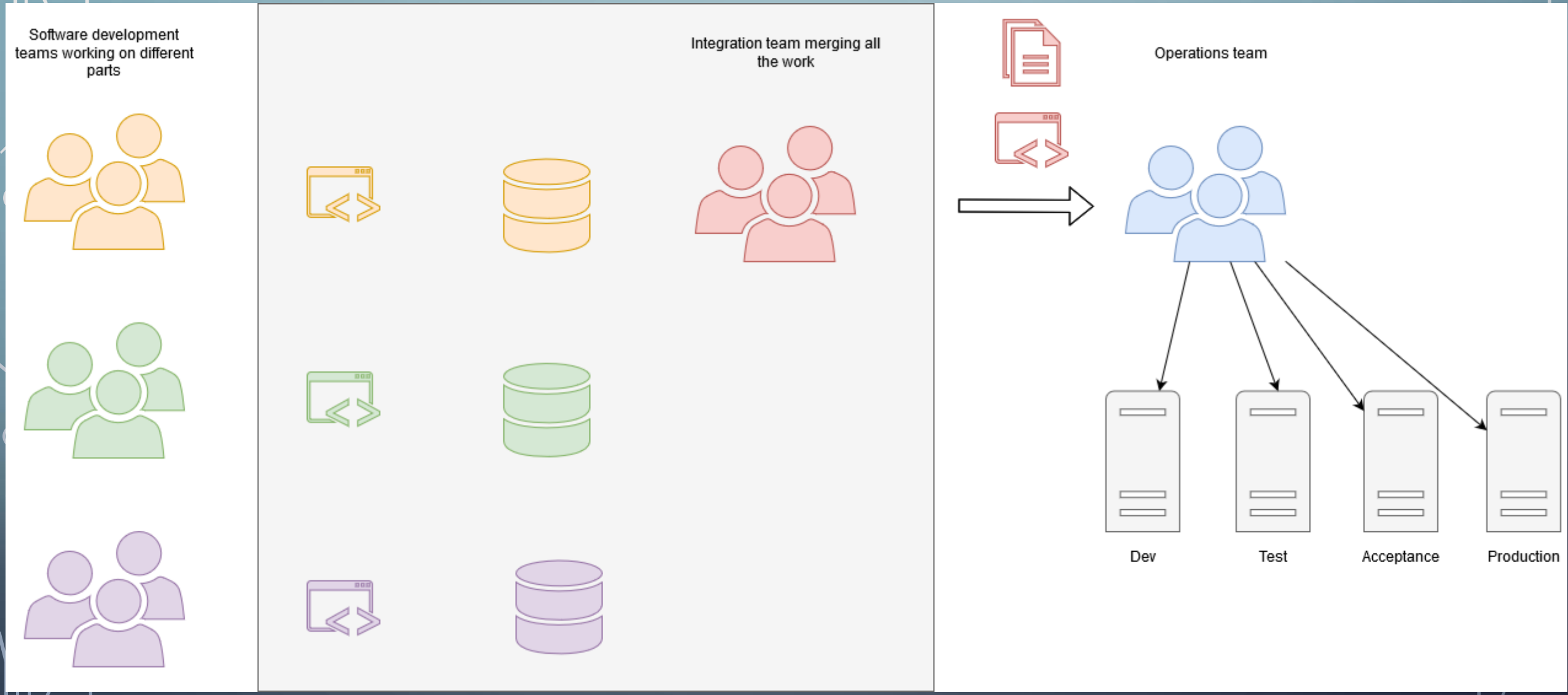
MERGE ASAP

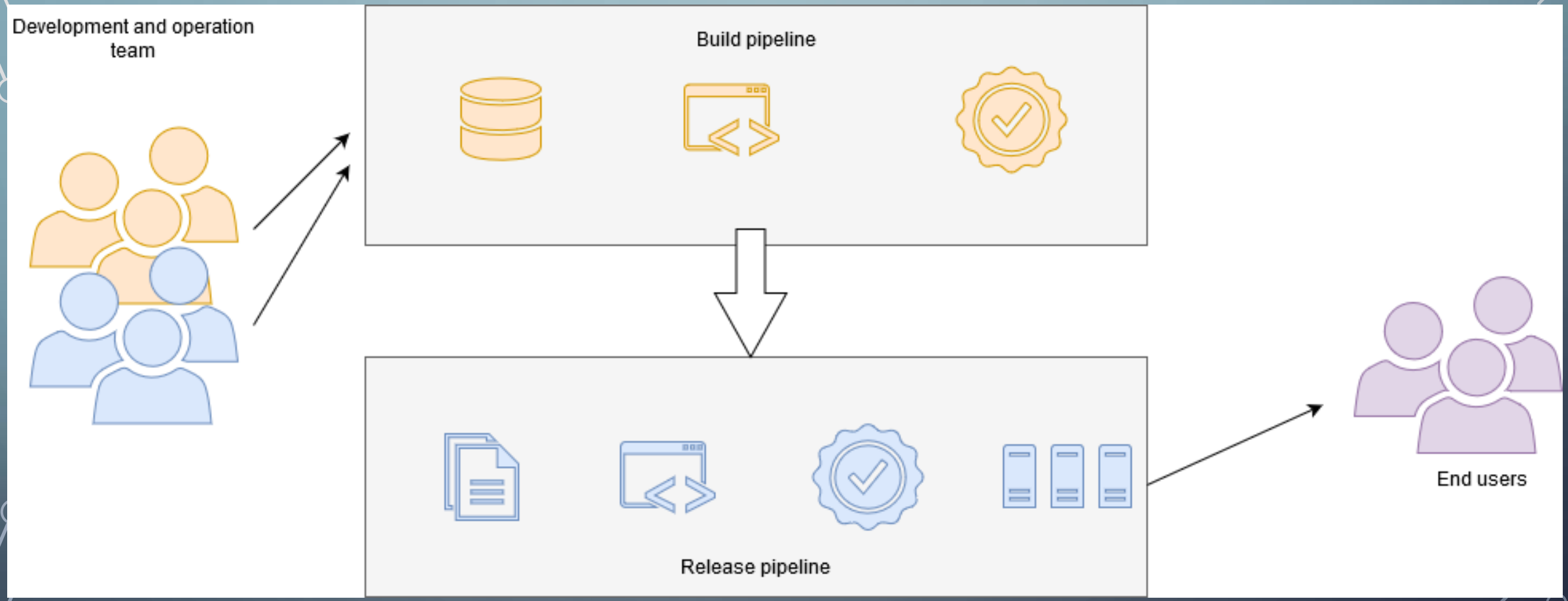
DO CODE REVIEWS

HAVE DETAILED AGREEMENTS ABOUT THE PROCESS

GIT BRANCHING STRATEGIES

- Many options, best one depends on:
  - Need of the team / company
  - Size of the application
  - Number of requests
  - Whether or not connected to CI/CD
- Example of a very light weight strategy:
  - Main branch
  - Dev branch
  - Every time you make a change, you create a new branch for that change based on dev. Remove branch after merging into dev.
  - Dev gets merged into master whenever needed / ready


Software development teams working on different parts
Integration team merging all the work
Operations team
Dev
Test
Acceptance
Production


Development and operation team
Build pipeline
Release pipeline
End users

RECAP

- Ignore
- Stash / pop
- Rebase
- Rollback
- Merge conflicten
- Final exercise
- What else is possible… CI/CD

QUESTION:

WHAT IS A PULL REQUEST?

QUESTION:

WHAT IS A MERGE CONFLICT? AND HOW TO DEAL WITH IT?

QUESTION:

WHAT ARE BEST PRACTICES OF VERSION CONTROL?

QUESTION:

WHY IS A PULL REQUEST NOT CALLED PUSH REQUEST?

QUESTION:

WHAT ARE STASH AND POP USED FOR?

QUESTION:

WHAT COMMAND SHOULD YOU USE TO GO BACK TO AN OLD COMMIT?

QUESTION:

WHAT IS THE PURPOSE OF CI/CD?

QUESTIONS?

- Maaike.vanputten@brightboost.nl