Version Control with GIT

Key Takeaways

TECHWORLD WITH NANA

What is Version Control?

- Also known as "source control"
- Practice of **tracking and managing changes to software code**
- It enables multiple people to **simultaneously work on a single project**

- Code is **hosted centrally on the internet**
- Every developer has an **entire copy** of the code **locally**

Basic Concepts of Version Control

- Version Control keeps a **history of changes**

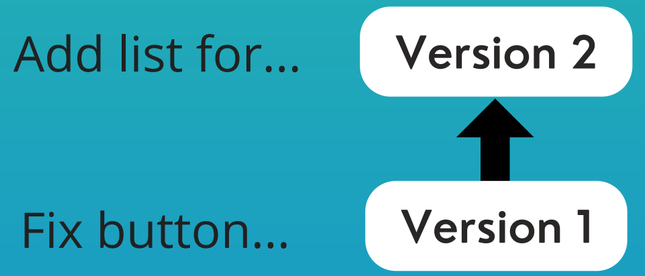
Every code change and file is tracked!

You can revert commits

Each change labelled with commit message

git

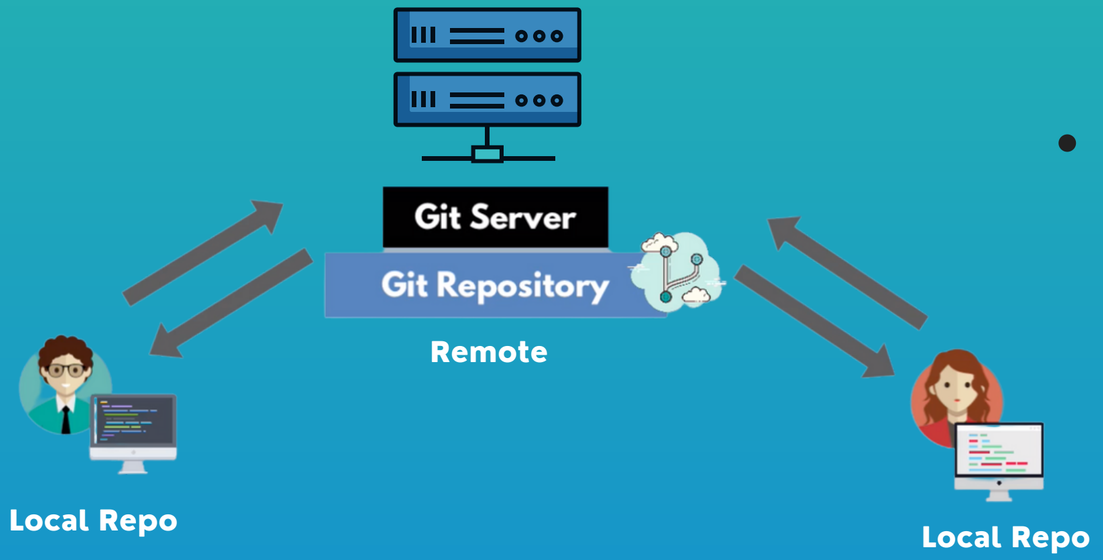
Git is the most used version control system



TECHWORLD WITH NANA

Basic Concepts of Git - 1

- **Remote Git Repository**: where the **code is hosted**, e.g. on Gitlab or GitHub
- **Local Git Repository**: **local copy** of the code on your machine
- **Git Client**: to connect and **execute git commands** can be UI or Command Line Tool



- Code is fetched ("pulled") from remote repo and "pushed" to it

TECHWORLD WITH NANA

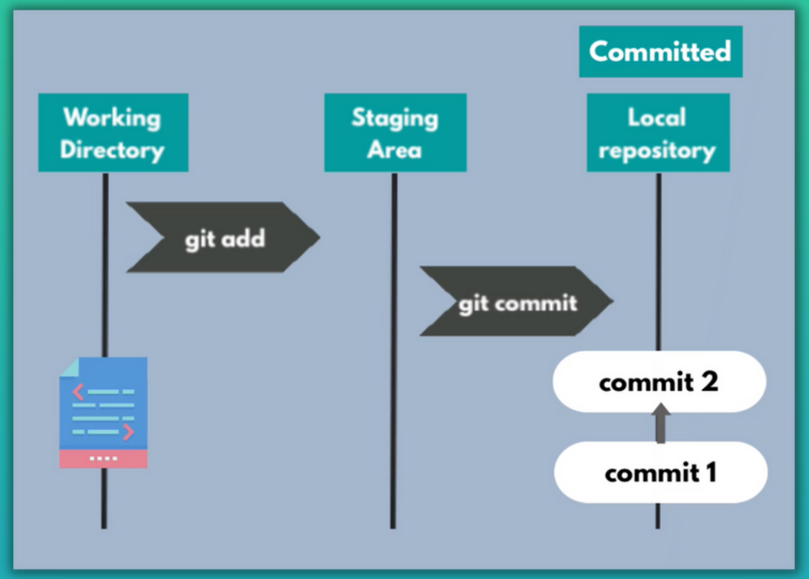
Basic Concepts of Git - 2

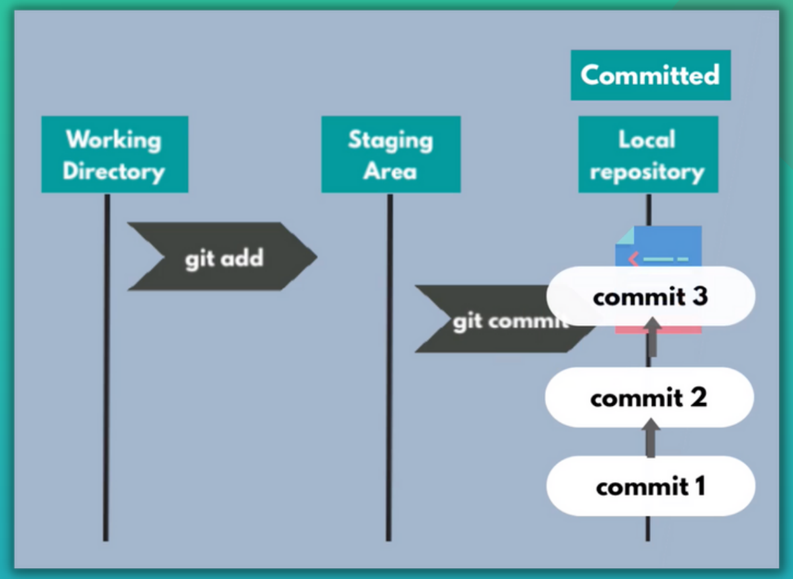
- Most of the time, Git knows how to **merge changes automatically**
- But you have a "**Merge Conflict**", when e.g. same line was changed. Then you need to resolve it manually

- To avoid merge conflicts: **Best Practice:** Push and Pull often from remote repository to stay in sync

- Note: Breaking changes doesn't affect you until you pulled the new code

Working with Git - 1





git add <file>:

- To include the changes of a file into the next commit
- Moves the changes from "working directory" to the "staging area"

git commit -m "commit message":

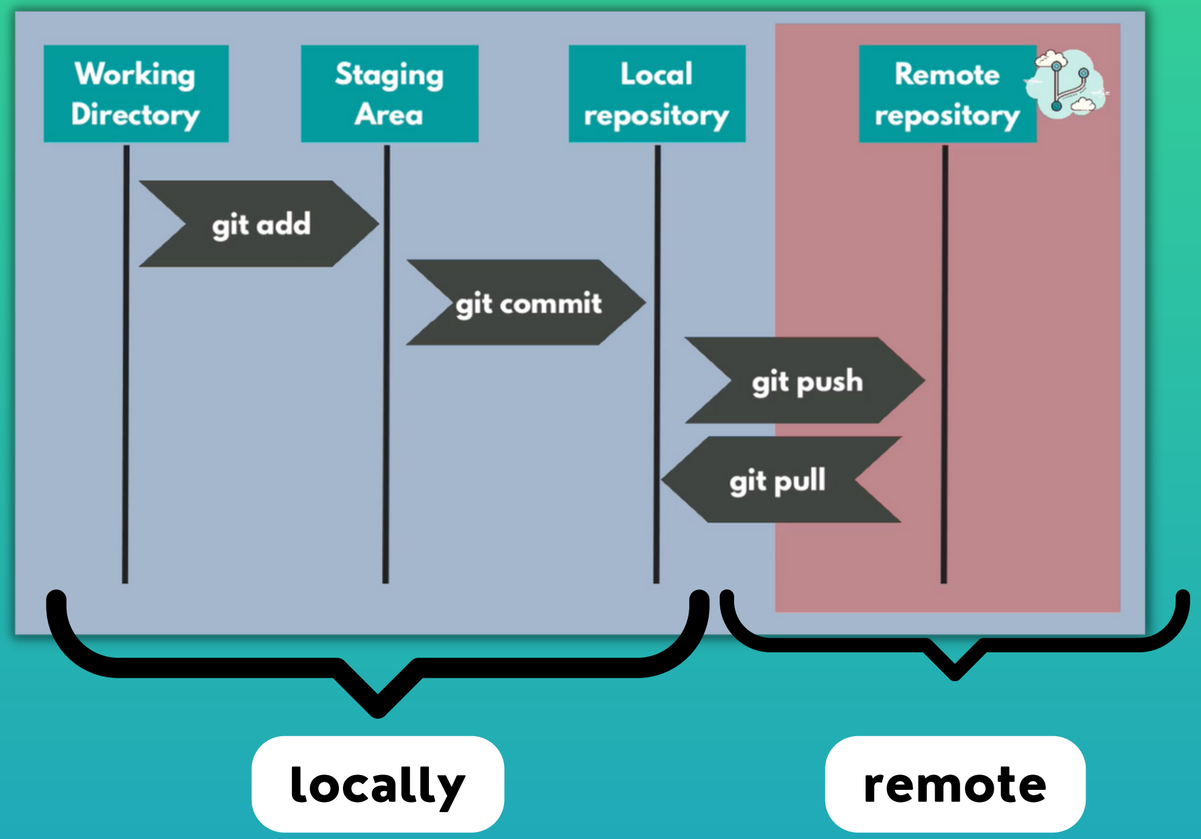
- To save your changes in your local repository
- Creates a new commit, which you can go back to later if needed

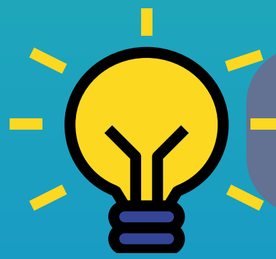
TECHWORLD WITH NANA

Working with Git - 2

git push <remote> <branch-name>:

- After committing your changes, you want to send your changes to the remote Git server
- Uploads your commits to the remote repo





At the end of the handout you can find a summary of most important git commands

TECHWORLD WITH NANA

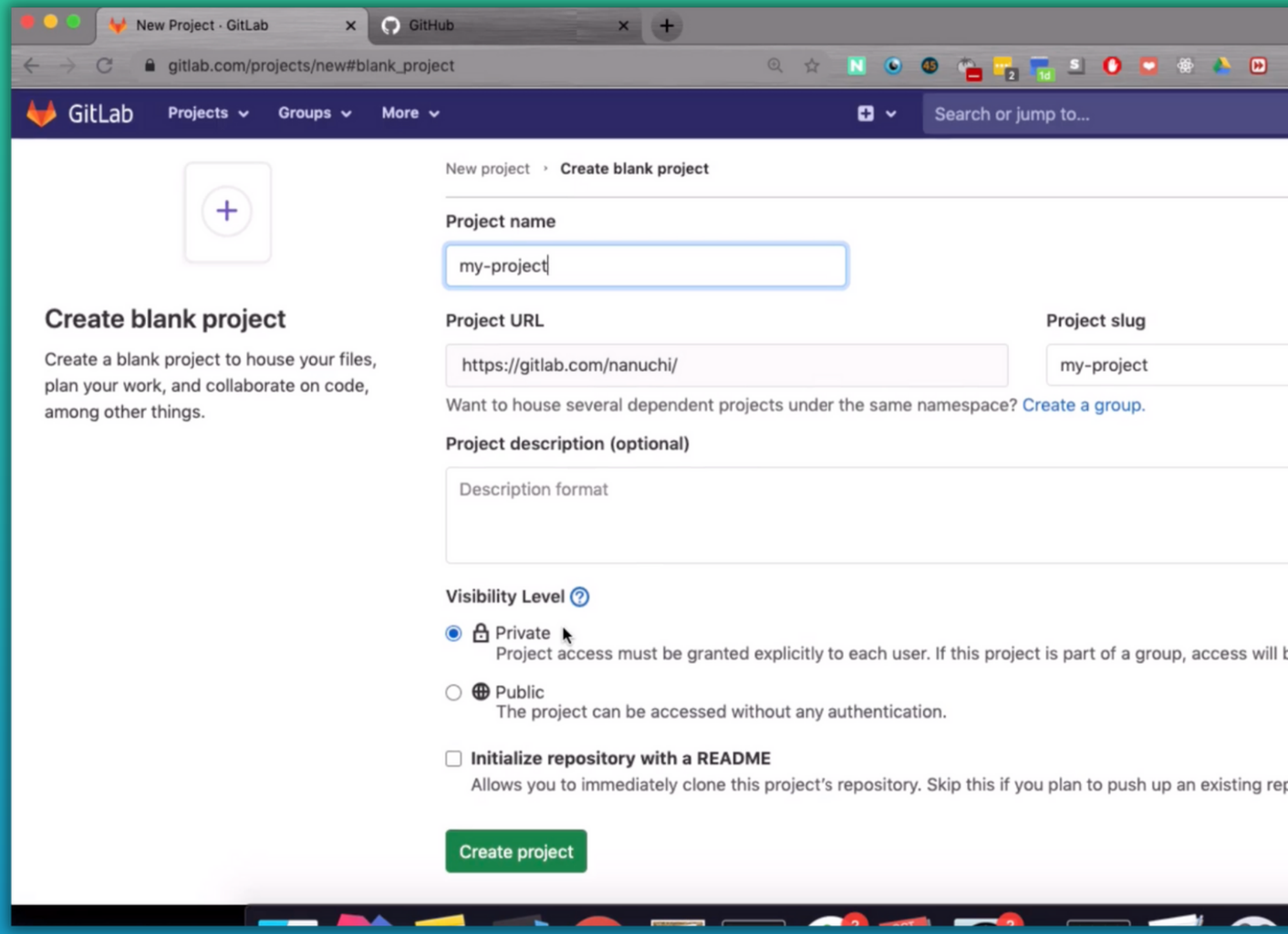
Setup Git Repository - 1

1) Remote Repository

- Different Git Repositories to register:

GitLab github

- These are platforms that **host your repository**
- Companies have own Git servers hosted by them
- Your repository can be **private or public**. E.g. Private for companies, Public for open source projects
- You can do a lot via the Platforms UI

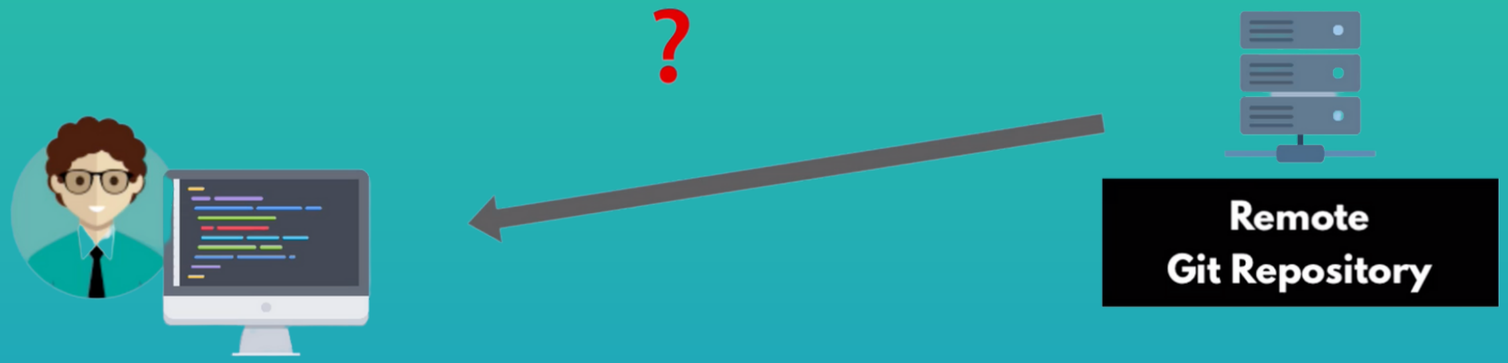


TECHWORLD WITH NANA

Setup Git Repository - 2

2) Local Repository

- Having the remote repository set up, you need a way to connect with the remote repository to copy or "**clone" git project to your local machine**



- **Git client** needs to be installed

UI client or **Git Command Line Tool**

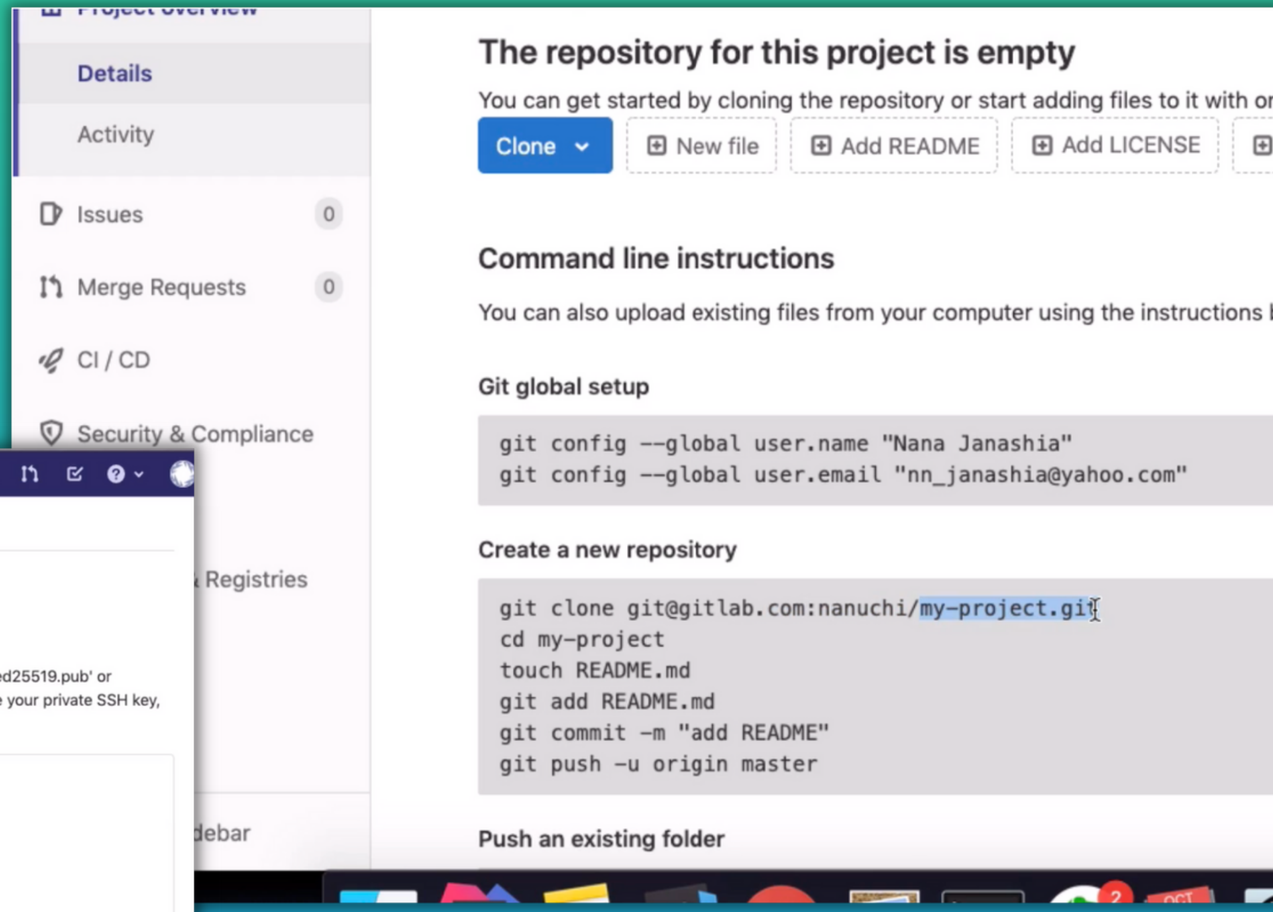
- https://git-scm.com/downloads/guis
- Installation Guide: https://git-scm.com/downloads

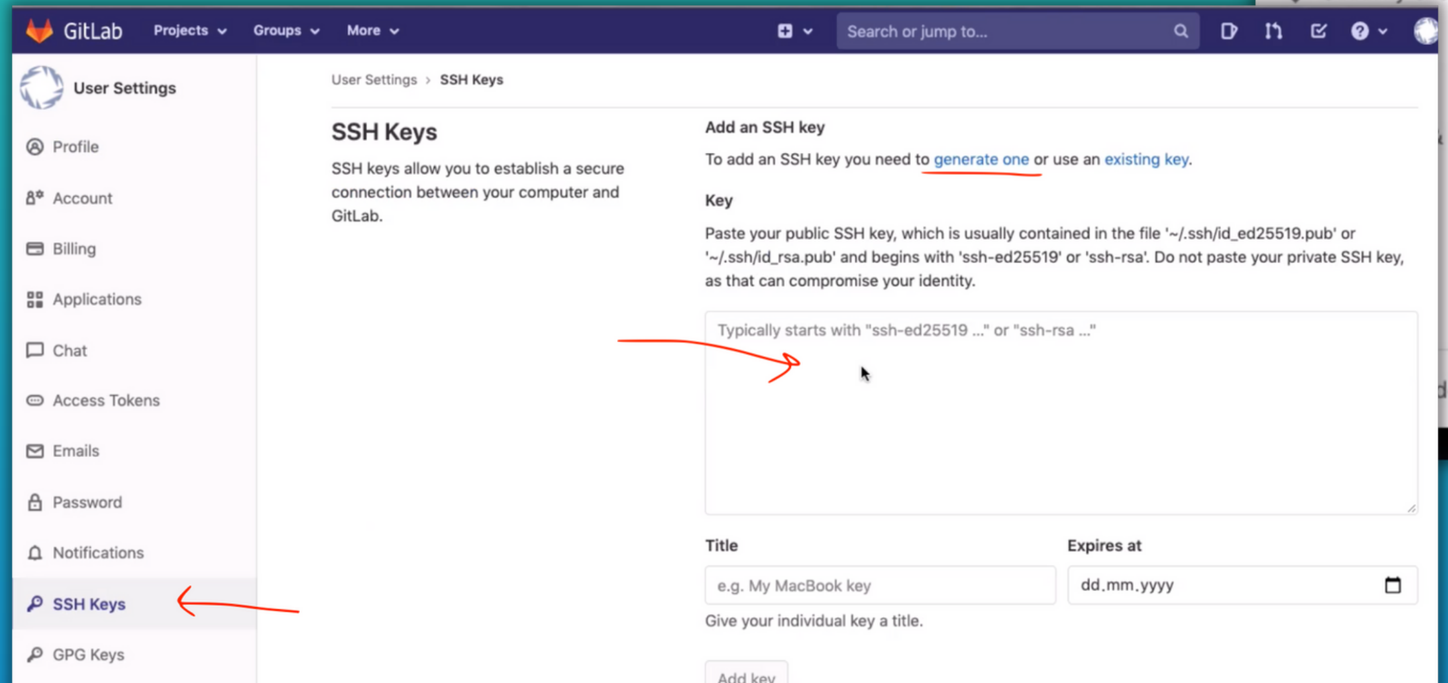
Setup Git Repository - 3

2) Local Repository

- You need to **authenticate** with GitHub/GitLab/...
- For that, your public SSH Key must be added to the remote platforms:

Getting started guide available:





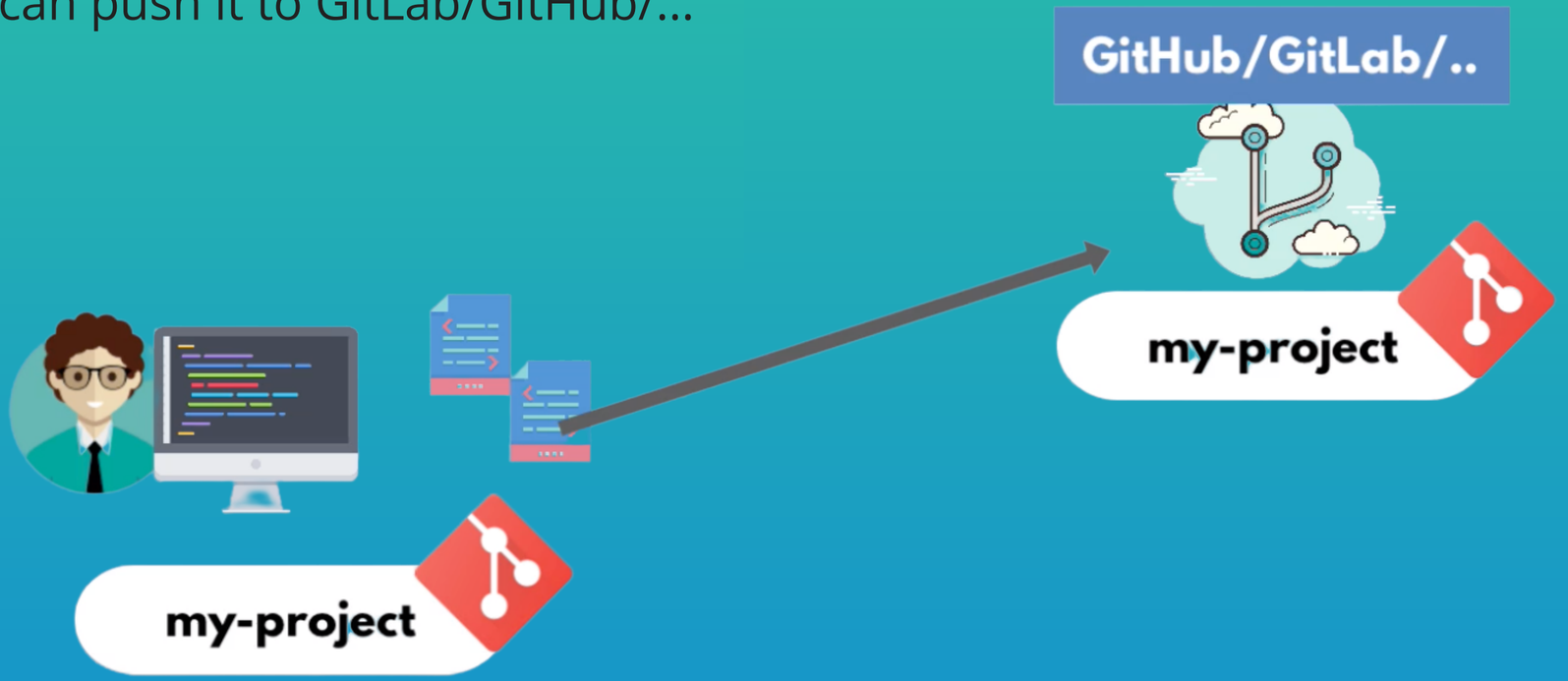
TECHWORLD WITH NANA

Setup Git Repository - 4

2) Local Repository

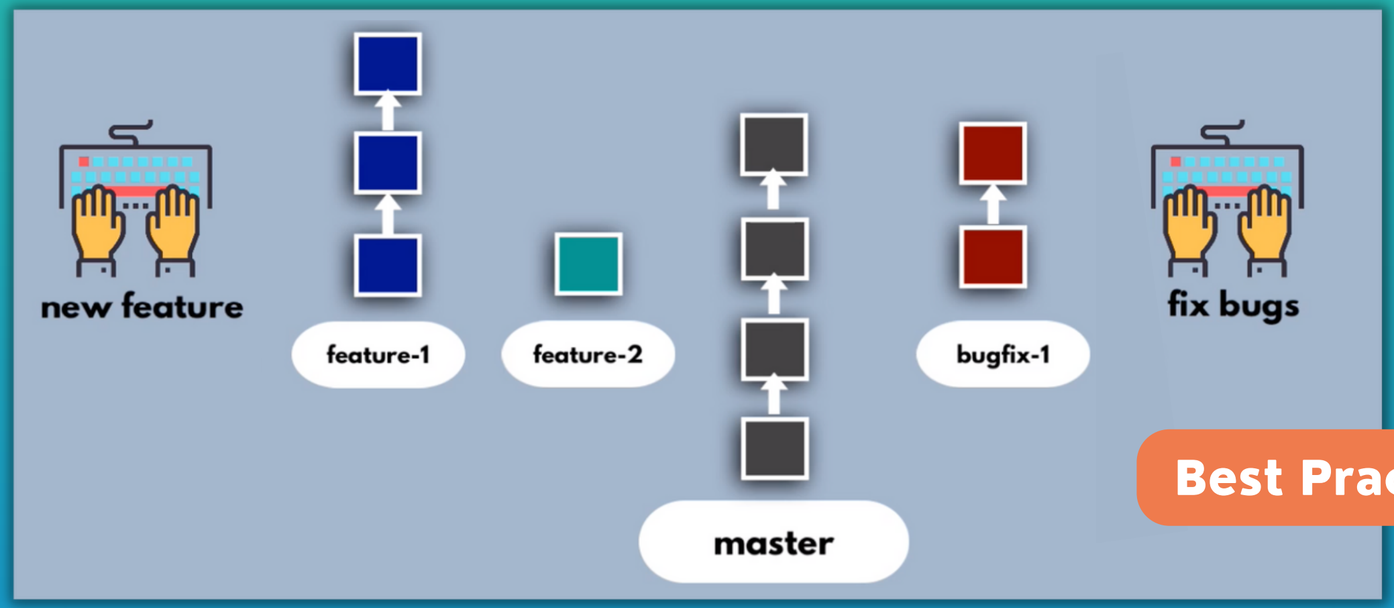
As an **alternative**:

- If you have already an existing project locally, you can **initialize a git repository with "git init"**
- Then you can push it to GitLab/GitHub/...



Concept of Branches - 1

- Branches are used for better collaboration
- A **"main" branch (also called "master")** is created by default when a new repository is initialized
- Each developer can then create temporary branches e.g. for a feature or bugfix and work on it without worrying to break the main branch



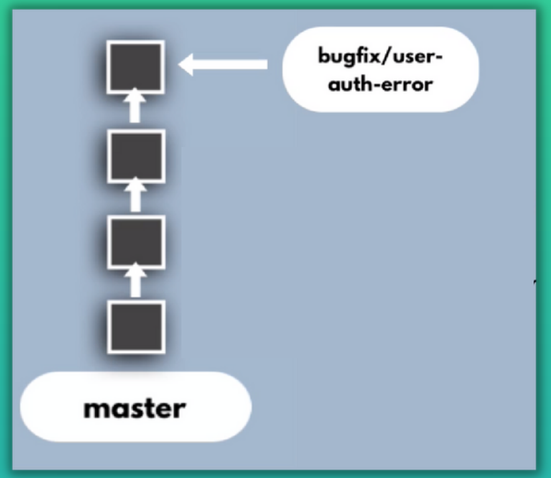
Here we have 4 different branches

- **Common branch names**: main, dev, feature, bugfix
  E.g. bugfix/ticket-2134

Best Practice: 1 branch per bugfix or feature

TECHWORLD WITH NANA

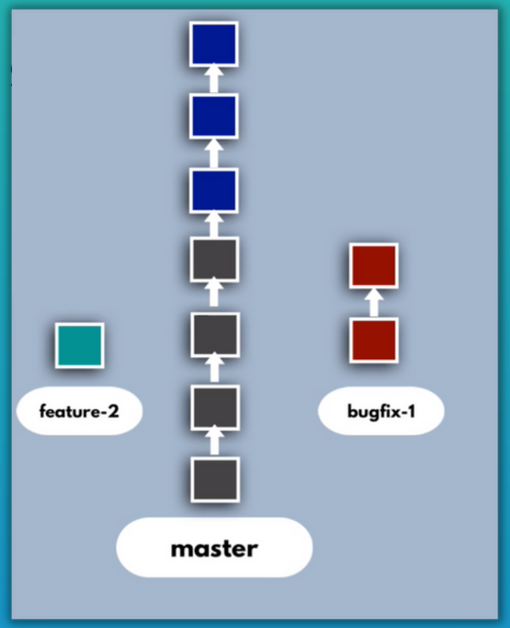
Concept of Branches - 2



- Branch is **based on main branch**. So, it starts from same codebase:

- When finished, the complete branch can be merged back to the main branch

Branch for "feature-1" was merged into the main branch:

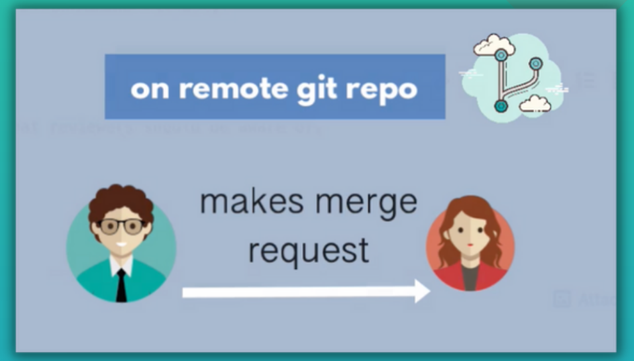


Goal is to have a stable main branch, ready for production deployment

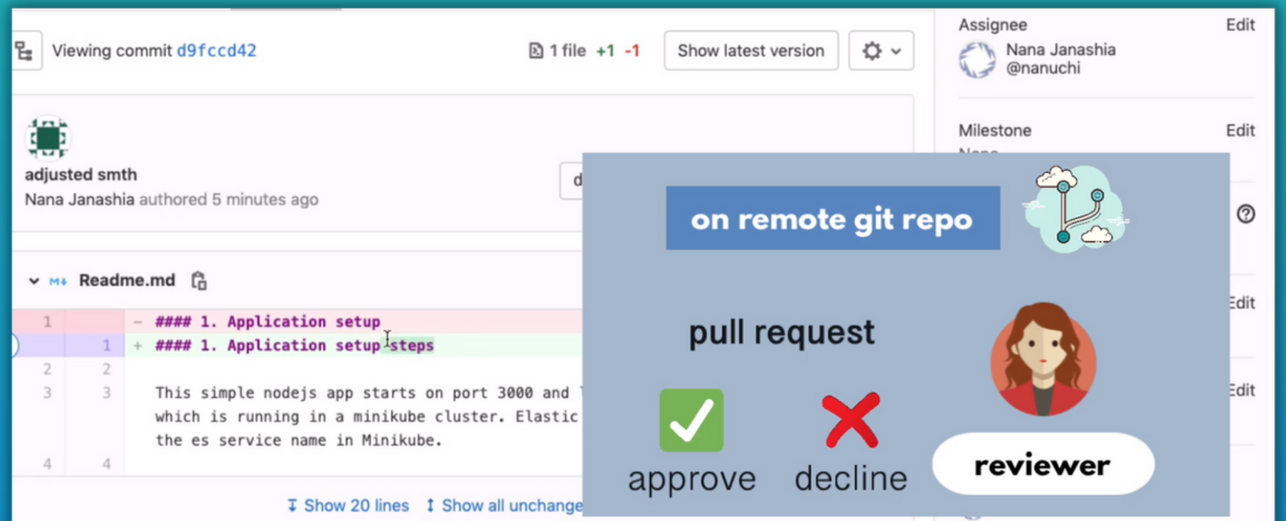
Merge Requests or Pull Requests

Best Practice: Other developer reviews code changes before merging

- For that we have "merge request" or also called "pull request"
- It's basically a ***request*** **to merge one branch into another** (usually in the main branch)



- Reviewer can see the changes made and either **approve or decline** the merge request



TECHWORLD WITH NANA

Why to know Git as DevOps Engineer? - 1

Use Case 1) Infrastructure as Code

- **As a DevOps engineer you write code** (configuration files and scripts) to create and provision infrastructure

Use Case 2) Automation Scripts

- **As a DevOps engineer you write automation scripts** e.g. with Python to automate different tasks

- So these files should also "live" in a remote repository and be **tracked and versioned by Git**

Just like software code, files should be:

- Tracked - history of changes
- Securely stored in one place
- Shareable to collaborate as team

TECHWORLD WITH NANA

Why to know Git as DevOps Engineer? - 2

Use Case 3)
CI/CD Pipeline and Build Automation

- CI means: On each merge, checkout code from repository, test and build application
- For that, you need integration for the build automation tool with application git repository
- You need to **setup integration with build automation tool and git repository**
- You need to **know git commands for example for:**

→ Getting commit hash of specific commit

→ Check if changes happened in frontend or backend code

Git Cheatsheet - 1

Source: https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

TECHWORLD WITH NANA

Git Cheatsheet - 2

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author="<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

TECHWORLD WITH NANA

Best Practices - 1

Commit-related best practices:

- Use descriptive and meaningful commit messages
- Commit in relatively small chunks
- Commit only related work
- Adequately configure the commit authorship (name and email address) with git config

Avoiding very large deviations between local and remote repository:

- Keep your feature/bugfix branch up-to-date with remote master and/or develop branch. So pull often from remote git repository
- Branches shouldn’t be open for too long or master branch should be merged into your feature/bugfix branch often

Best Practices - 2

Other:

- Don’t "git push" straight to main branch
- Use -force push carefully! Do NOT force push into master or develop branches or better only when working alone in a branch
- Create a separate branch for each feature or bugfix and name the branch with prefix “feature/xx” and “bugfix/xxx” respectively
- Doing Code Reviews via Merge Requests
- Use .gitignore file to ignore e.g. editor specific files, build folders