Git vs GitLabs vs Github vs Version Control

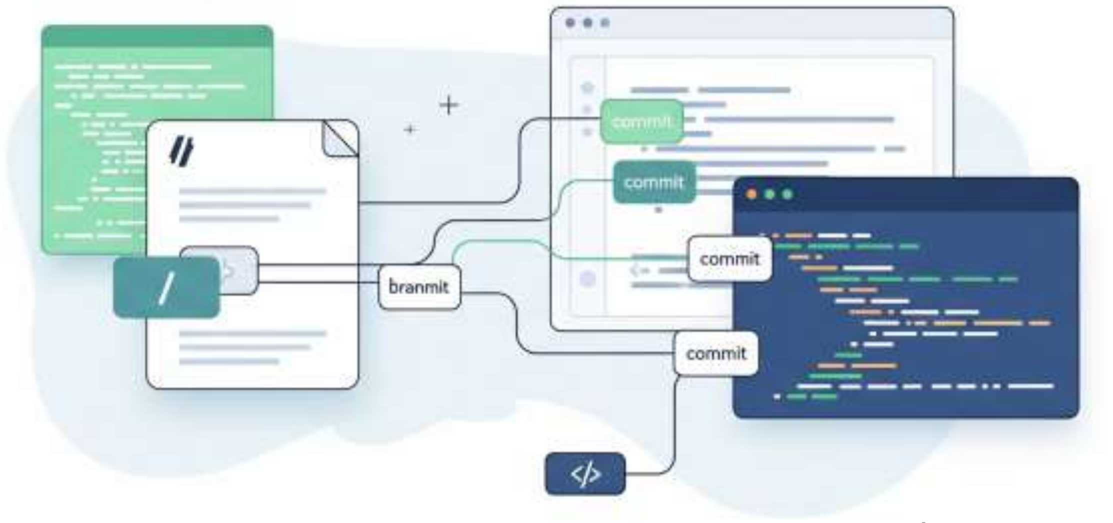
What is Git?

- Distributed Version Control System (VCS)
- aka Software Configuration Management (SCM) system
- Used by developers to manage changes to their source code
- "Distributed" means every developer has a full copy of the project's history on their local machine
- Fast, versatile, highly scalable and open-source
- Originally developed by Linus Torvalds, the creator of the Linux operating system kernel

Official (Closed) - Non Sensitive

Purpose of Git

- Version control to track, manage and collaborate
- Save snapshots of your source code
- Roll back to previous versions if needed
- Collaborate with others without overwriting each other’s changes
- Prevents “lost work”, and helps teams work together smoothly, even across the globe



Git Terminology

- **Repository (repo)** - the entire project history and metadata, including all commits and branches, stored in the *.git* folder
- **Working tree** - the directory on your system where you edit files - it reflects the current checked-out version of the project
- **Branch** - a lightweight, movable pointer to a commit, used to isolate different lines of development
- **Commit** - a snapshot of changes in the project, recorded in the repository with a message and metadata
- **Hash** - a unique 40-character SHA-1 identifier that Git uses to reference commits and other objects

Common Git Commands

`$ git init`	*Create a Git repository in the current folder*
`$ git status`	*View the status of each file in the repository*
`$ git add <file>`	*Stage a file for the next commit*
`$ git commit`	*Commit the staged file(s) with a descriptive message*
`$ git log`	*View the repository's commit history*
`$ git pull`	*Fetch changes from remote repository and merges them into current branch*
`$ git push`	*Upload local commits to a remote repository branch*

Local vs Remote Repository

- **Local Repo:** Lives on your computer. You control commits and branches locally
- **Remote Repo:** Stored on a server in a network (e.g. GitHub, GitLab, Bitbucket) so that others can access it
- **Why both?:** You can work offline locally, then sync changes with the remote repo when ready

Essential Git Command: git init

- This command initializes a new, empty Git repository in the current directory.
- It creates a hidden *.git* folder that tracks all the changes.
- **Command:** git init

```
$ git init
```

Essential Git Command: git clone

- This command copies an existing repository from GitHub to your local machine
- It's how you get a copy of a project to work on
- **Command:** git clone <repository-url>
- **Example:** git clone https://github.com/octocat/Spoon-Knife.git

```
$ git clone https://github.com/octocat/Spoon-
Knife.git
```

Essential Git Command: git status

- This is your best friend! It tells you the current state of your working directory and staging area
- It shows which files have been modified, staged, or are untracked
- **Command:** `git status`

```
$ git status
```

Essential Git Command: git add

- This command moves changes from the working directory to the staging area
- You must add a file before you can commit it
- To stage a single file: git add index.html
- To stage all changes: git add .

```
$ git add index.html
```

```
$ git add .
```

Essential Git Command: git commit

- This command takes the staged changes and saves them permanently in your local repository
- Every commit must have a message explaining what you changed
- **Command:** `git commit -m "Your commit message here"`
- **Example:** `git commit -m "Added header to homepage"`

```
$ git commit -m “Added header to homepage”
```

Essential Git Command: git push

- This command uploads your local commits to a remote repository (like GitHub)
- It "pushes" your changes to the server
- **Command:** git push

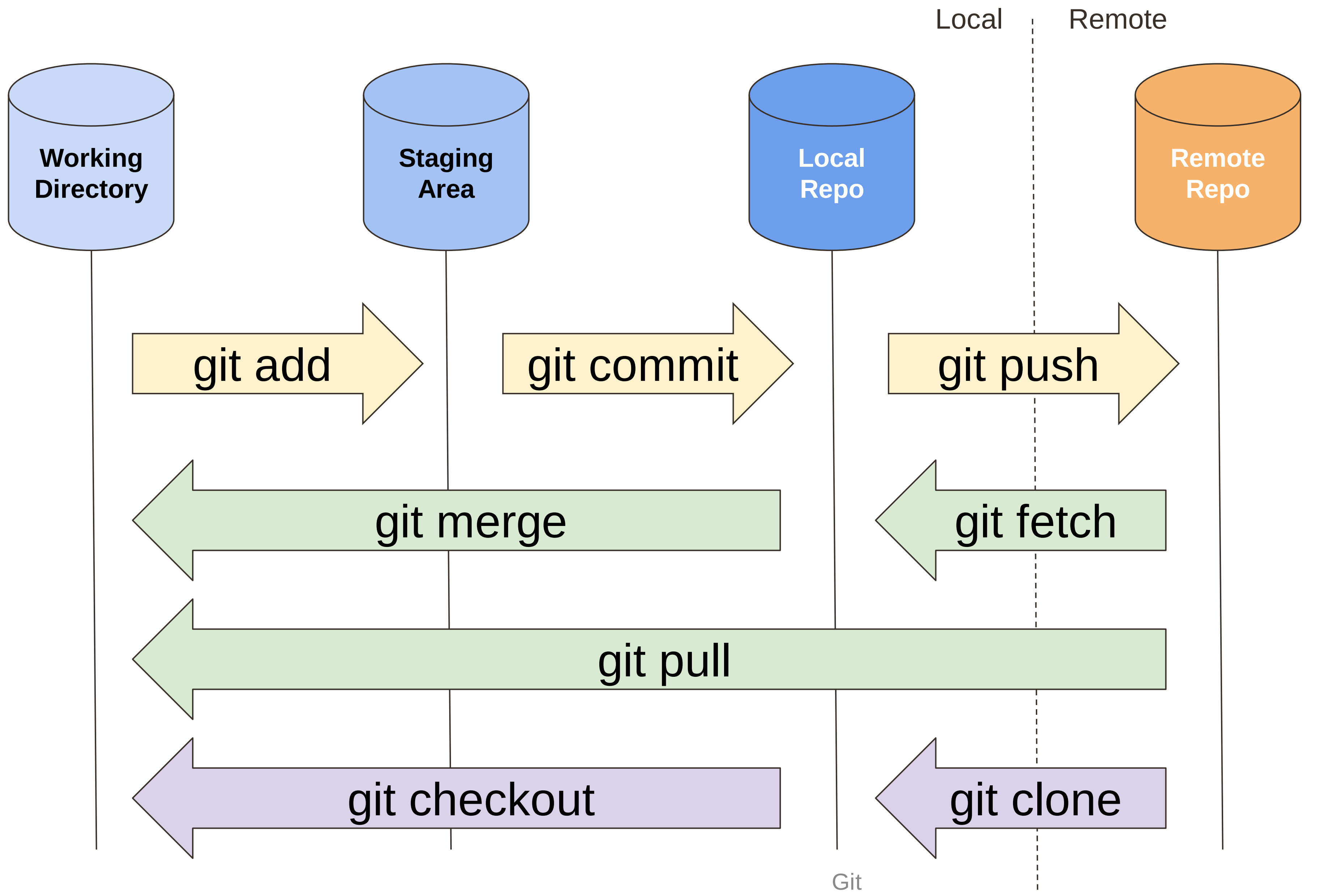
```
$ git push
```

Essential Git Command: git pull

- This command downloads changes from the remote repository to your local machine
- It "pulls" new work from other collaborators
- Command: git pull

```
$ git pull
```

Git Workflow



Branching and Merging

- **Branching**
  - Lets you work on features, experiments, or fixes without touching the main branch
  - Commands
    - `git branch <name>` - Create a branch
    - `git checkout <name>` - Switch to a branch
- **Merging**
  - Combines changes from one branch into another
  - `git merge <branch>` - Merge the branch into your current branch

Essential Command: git branch

- This command lets you manage your branches
- To list all branches: git branch
- To create a new branch: git branch <new-branch-name>

```
$ git branch
```

```
$ git branch my-new-branch
```

Essential Command: git checkout

- This command lets you switch between branches
- It updates your working directory to match the code in that branch
- **Command:** git checkout <branch-name>
- To create and switch to a new branch in one step:
  git checkout -b <new-branch-name>

```
$ git checkout my-new-branch
```

```
$ git checkout -b my-newest-branch
```

Essential Command: git merge

- This command combines a branch into your current branch
- After you finish working on a feature branch, you merge it back into main
- **Command:** git merge <branch-to-merge>
- **Example:** From the main branch, run git merge feature-a to bring the changes from feature-a into main

```
$ git merge my-new-branch
```

Essential Command: git log

- This command shows a log of all the commits in your repository
- It displays the commit hash, author, date, and commit message
- **Command:** git log
- **To see a simplified log:** git log --oneline

```
$ git log
```

```
$ git log --oneline
```

Essential Command: git diff

- This command shows the difference between your working directory and the staging area, or between two commits
- It's a great way to review your changes before you commit them
- **Command:** git diff

```
$ git diff
```

Installing Git on your local machine

- **Windows**
  - Go to https://git-scm.com
  - Download the Windows installer
  - Run installer with default options
- **MacOs**
  - Install via Homebrew - `brew install git`
  - Or download from https://git-scm.com
- **Linux (Debian/Ubuntu)**
  - `sudo apt install git`

Signing Up for GitHub

1. Go to https://github.com
2. Click **Sign Up**
3. Enter
   a. Username
   b. Email
   c. Password
4. Choose a free plan
5. Verify your email address
6. Login and set up a profile picture (optional)

GitHub Workflow

- Clone a repository from GitHub
- Create a new branch to work on your feature
- Make changes, git add, and git commit them locally
- git push your new branch to GitHub
- Create a Pull Request to merge your changes

Creating a New Repository on GitHub

- Go to github.com and log in
- Click the + button in the top right corner and select "New repository.”
- Give it a name and a description
- Choose if it's public or private
- Click "Create repository."

Pushing to a New Repository

- After creating the repo on GitHub, you can connect your local project to it
- **Command:** git remote add origin <repository-url>
- **Command:** git push -u origin main (This pushes your main branch and sets it to track the remote origin branch)

```
$ git push -u origin main
```

Pull Requests (PRs)

- A Pull Request is how you propose changes to a project on GitHub
- It's a way to get feedback and have other people review your code before it's merged
- When you push a new branch, GitHub will often show a button to create a PR

Resolving Merge Conflicts

- A merge conflict happens when Git can't automatically combine two different changes
- This usually happens when two people edit the same line of code
- Git will mark the conflict in the file
- You must manually choose which version to keep, save the file, git add, and git commit to resolve it

Git in Collaboration

- **Team workflow:**
  - Each member clones the remote repository
  - Work happens in separate branches
  - Changes are pushed to the remote repository
- **Pull Requests / Merge Requests**
  - A way to propose changes before merging them into the main branch
  - Allows for code review and discussion

Best Practices

- Commit often, with clear, descriptive messages
- Always *pull before pushing* to reduce conflicts
- Use branches for every new feature or bug fix
- Keep the *main* (or *master*) branch clean and deploy-ready
- Don’t commit sensitive files like passwords or API keys