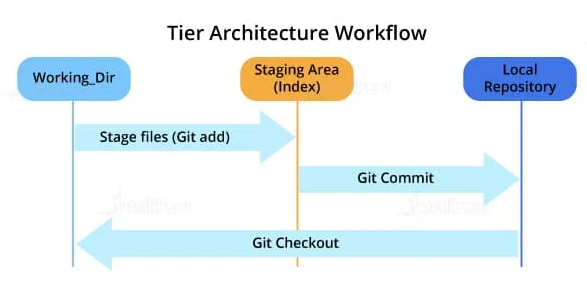
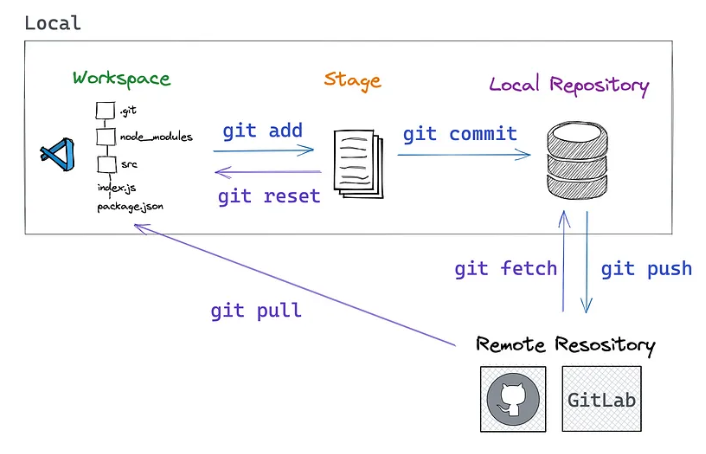
****

**What is Git?**

Sure! Git is a version control system that helps you manage and track changes to your code (or any files, really) over time. It's like a time machine that allows you to revisit different versions of your work, collaborate with others seamlessly, and revert to previous states if needed.

**Git uses a three-stage architecture: -** working directory, staging area, and local repository to optimize change tracking.





## **Init**

git init

Creates a new empty repo in your current directory.

## **Clone**

git clone <https://github.com/facebook/react.git>

Copies a remote repository into your current directory.

## **add**

The git add command adds new or changed files in your working directory to the Git staging area.

git add <file1> <file2>

Adds <file1> and <file2> to the staging area.

git add \*.js

Adds all JavaScript files in the current directory to the staging area.

## **Reset**

git reset

Removes all files from the staging area. (Opposite of git add .)

git reset <filename>

Removes <filename> from the staging area.

## **commit**

Save prepared changes in the staging area to the local repository.

git commit -m "update the README.md with link to contributing guide"

Records everything in the staging area to your repository with the commit message.

To create an Azure DevOps Git account, follow these steps:

1. **Sign Up for Azure DevOps:**
   * Go to the Azure DevOps website (dev.azure.com).
   * Click on "Start free" or "Sign up for free" to create an account if you don't already have one.
2. **Create a New Project:**
   * Once logged in, click on "Create project" to start a new project.
   * Give your project a name and choose the visibility (public or private).
3. **Set Up Your Git Repository:**
   * Inside your project, navigate to the Repos area (Repositories).
   * Click on "New repository" to create a new Git repository.
4. **Clone Your Repository:**
   * To start working with Git locally on your computer, you need to clone the repository.
   * Click on "Clone" and copy the repository URL provided.
5. **Configure Git on Your Local Machine:**
   * Install Git on your computer if you haven't already (you can download it from git-scm.com).
   * Open a terminal or command prompt and navigate to the directory where you want to clone your repository.
   * Use the command git clone <repository URL> to clone the repository to your local machine.
6. **Start Using Git:**
   * Now you can start adding files, making changes, committing those changes locally with git add and git commit, and pushing them to your Azure DevOps repository with git push.

Azure DevOps offers a lot more features beyond just Git repositories, such as project management tools, CI/CD pipelines, and more, so it's a versatile platform for software development and collaboration.

**Git Commands:**

Certainly! Git commands are the set of instructions you use in the Git version control system to manage your source code and collaborate with others. Here's an overview of some common Git commands and their explanations:

**1. git init**

* **Explanation:** Initializes a new Git repository in the current directory.
* **Usage:** git init

**2. git clone**

* **Explanation:** Copies a remote repository (like on GitHub or Azure DevOps) to your local machine.
* **Usage:** git clone <repository URL>

**3. git add**

* **Explanation:** Adds changes in the current directory to the staging area.
* **Usage:**
  + git add <file> (adds a specific file)
  + git add . (adds all files and directories in the current directory)

**4. git commit**

* **Explanation:** Records the changes added to the staging area as a new commit in the local repository.
* **Usage:**
  + git commit -m "Commit message" (commits with a message)
  + git commit -a -m "Commit message" (adds all modified files and commits)

**5. git status**

* **Explanation:** Shows the current status of the repository, including tracked, untracked, and staged files.
* **Usage:** git status
  1. **git push**
* **Explanation:** Uploads local repository content to a remote repository.
* **Usage:** git push <remote> <branch>
  + Example: git push origin main

1. **git pull**

* **Explanation:** Fetches and integrates changes from a remote repository to the current branch.
* **Usage:** git pull <remote> <branch>
  + Example: git pull origin main

1. **git branch**

* **Explanation:** Lists existing branches or creates a new branch.
* **Usage:**
  + git branch (lists all branches)
  + git branch <branch-name> (creates a new branch)

1. **git checkout**

* **Explanation:** Switches branches or restores working tree files.
* **Usage:**
  + git checkout <branch-name> (switches to an existing branch)
  + git checkout -b <new-branch> (creates and switches to a new branch)

1. **git merge**

* **Explanation:** Integrates changes from another branch into the current branch.
* **Usage:**
  + Switch to the branch you want to merge into (e.g., git checkout main)
  + git merge <branch-to-merge>

1. **git log**

* **Explanation:** Shows the commit logs.
* **Usage:** git log

1. **git remote**

* **Explanation:** Manages connections to remote repositories.
* **Usage:**
  + git remote add <name> <url> (adds a new remote repository)
  + git remote -v (lists all remotes)

1. **git fetch**

* **Explanation:** Downloads objects and refs from another repository.
* **Usage:** git fetch <remote>

1. **git reset**

* **Explanation:** Resets the current HEAD to a specified state.
* **Usage:**
  + git reset <commit> (unstages changes and moves HEAD to a specified commit)
  + git reset --hard <commit> (discards all changes and moves HEAD to a specified commit)

1. **git revert**

* **Explanation:** Reverts a commit by creating a new commit.
* **Usage:** git revert <commit>

These are some fundamental Git commands that cover most version control tasks. Mastering these commands will enable you to effectively manage your codebase, collaborate with others, and navigate through different versions of your project.

#### **Basic Git Commands:**

* **Initialization**: git init - Initializes a new Git repository.
* **Cloning**: git clone <repository\_url> - Creates a copy of a remote repository on your local machine.
* **Checking Status**: git status - Shows the state of the working directory and staging area.
* **Adding Changes**: git add <file> - Stages changes to be committed.
* **Committing Changes**: git commit -m "message" - Records changes to the repository with a message.
* **Pushing Changes**: git push origin <branch> - Uploads local commits to a remote repository.
* **Pulling Changes**: git pull origin <branch> - Fetches and integrates changes from a remote repository.
* **Branching**: git branch <branch\_name> - Creates a new branch; git checkout <branch\_name> - Switches to a branch.
* **Merging**: git merge <branch\_name> - Merges changes from another branch into the current branch.

**Steps:**

1. **Clone Repository:**
   * Command: git clone <repository\_url>
   * Creates a local copy of the remote repository.
2. **Make Changes:**
   * Modify files in your local working directory.
3. **Check Status:**
   * Command: git status
   * Displays the status of changes in the working directory and staging area.
4. **Stage Changes:**
   * Command: git add <file> or git add . (to stage all changes)
   * Adds changes to the staging area.
   * The staging area, also known as the **index**, is a file (usually located in .git/index) that acts as a middleman between the working directory and the repository. It holds a snapshot of the changes you plan to commit.
5. **Commit Changes:**
   * Command: git commit -m "commit message"
   * Records the staged changes to the repository with a descriptive message.
6. **Push Changes:**
   * Command: git push origin <branch\_name>
   * Uploads local commits to the remote repository.
7. **Pull Changes:**
   * Command: git pull origin <branch\_name>
   * Fetches and integrates changes from the remote repository.

### Typical Workflow:

1. **Edit Files**: Make changes in the working directory.
2. **Stage Changes**: Use git add to move changes to the staging area.
3. **Commit Changes**: Use git commit to save the staged changes to the local repository.
4. **Push Changes**: Use git push to send your commits to the remote repository (if collaborating).

### Summary:

**Git stages help manage and track changes through several phases:**

The working directory (where changes are made).

The staging area (where changes are prepared for committing).

The local repository (where commits are stored).

The remote repository (where changes are shared).

Understanding these stages and how to work with them effectively is crucial for managing your project’s history and collaborating with others.

Git is a distributed version control system widely used for managing source code in software development. Its core concepts are designed to help developers track changes, collaborate effectively, and maintain a history of their codebase. Here’s a detailed breakdown of Git's key concepts:

**1.** **Repository (Repo)**

A repository is the central place where Git stores all the files and the history of changes. It can be local (on your own computer) or remote (on a server, such as GitHub, GitLab, or Bitbucket).

* Local Repository: The version of the repository on your local machine. It contains all the files and history relevant to your work.
* Remote Repository: A version of the repository hosted on a server. It allows multiple collaborators to access and contribute to the project.

**2. Working Directory**

The working directory is where you have your files and make changes. It represents the files in your project as they are on your local filesystem.

**3. Staging Area (Index)**

The staging area is a file, also known as the index, that acts as a middle ground between the working directory and the repository. When you add changes to the staging area, you are preparing them to be committed.

**4. Commit**

A commit is a snapshot of your changes. Each commit has a unique identifier (SHA-1 hash) and includes:

* The changes made to the files.
* A commit message describing the changes.
* Metadata, including the author and timestamp.

Commits form a linked list of changes, creating a history of the project.

**5. Branch**

A branch is a separate line of development. It allows you to work on different features or fixes independently from the main codebase (usually the main or master branch). Branches help manage development workflows and enable collaboration.

* Master/Main Branch: The default branch where the code is generally stable and production-ready.
* Feature Branches: Used for developing new features or fixes. They are eventually merged into the main branch.

**6. Merge**

Merging is the process of combining changes from one branch into another. Git automatically handles the merging of changes, but sometimes conflicts may arise if changes overlap. In such cases, manual resolution is required.

**7. Rebase**

Rebasing is an alternative to merging that involves moving or combining a sequence of commits to a new base commit. This can simplify the history but requires careful handling to avoid rewriting shared history.

**8. Clone**

Cloning creates a copy of a remote repository on your local machine. This allows you to work on the project offline and push changes back to the remote repository when you're ready.

**9. Pull**

Pulling fetches changes from a remote repository and merges them into your local repository. It combines git fetch (which retrieves updates from the remote) with git merge (which integrates those updates).

**10. Push**

Pushing updates your remote repository with commits from your local repository. It sends your changes to the remote, making them available to others.

**11. Fetch**

Fetching retrieves updates from a remote repository without merging them into your local branch. This allows you to see what changes are available before integrating them.

**12. Tag**

Tags are used to mark specific points in history, usually to denote releases or milestones. They are like bookmarks in the commit history and can be used to easily reference a particular state of the repository.

**13. Conflict**

A conflict occurs when Git is unable to automatically reconcile differences between commits or branches. Manual resolution is needed to determine which changes should be kept.

**14. Stash**

Stashing temporarily shelves changes that you don’t want to commit right away. This is useful when you need to switch branches but want to keep your current work.

**15. Checkout**

Checking out a branch or commit means switching to that branch or specific snapshot of the codebase. This updates your working directory to reflect the state of the chosen branch or commit.

**Workflow Example**

A typical workflow using Git might look like this:

1. Clone a remote repository to your local machine.
2. Create a new branch for a feature or bug fix.
3. Make changes in the working directory.
4. Stage the changes by adding them to the staging area.
5. Commit the staged changes with a descriptive message.
6. Push the commits to the remote repository.
7. Create a pull request or merge the feature branch into the main branch when the work is complete.

Git's distributed nature, along with these concepts, allows for a flexible and robust version control system that supports both individual and collaborative development.