Lesson Plan

Git Workflows and Lifecycle

PW SKILLS

Git Workflows and Lifecycle

Introduction to Git

Git is a distributed version control system that helps developers manage and track changes to their codebase. Understanding Git workflows and the lifecycle of a Git repository is crucial for effective collaboration and version control.

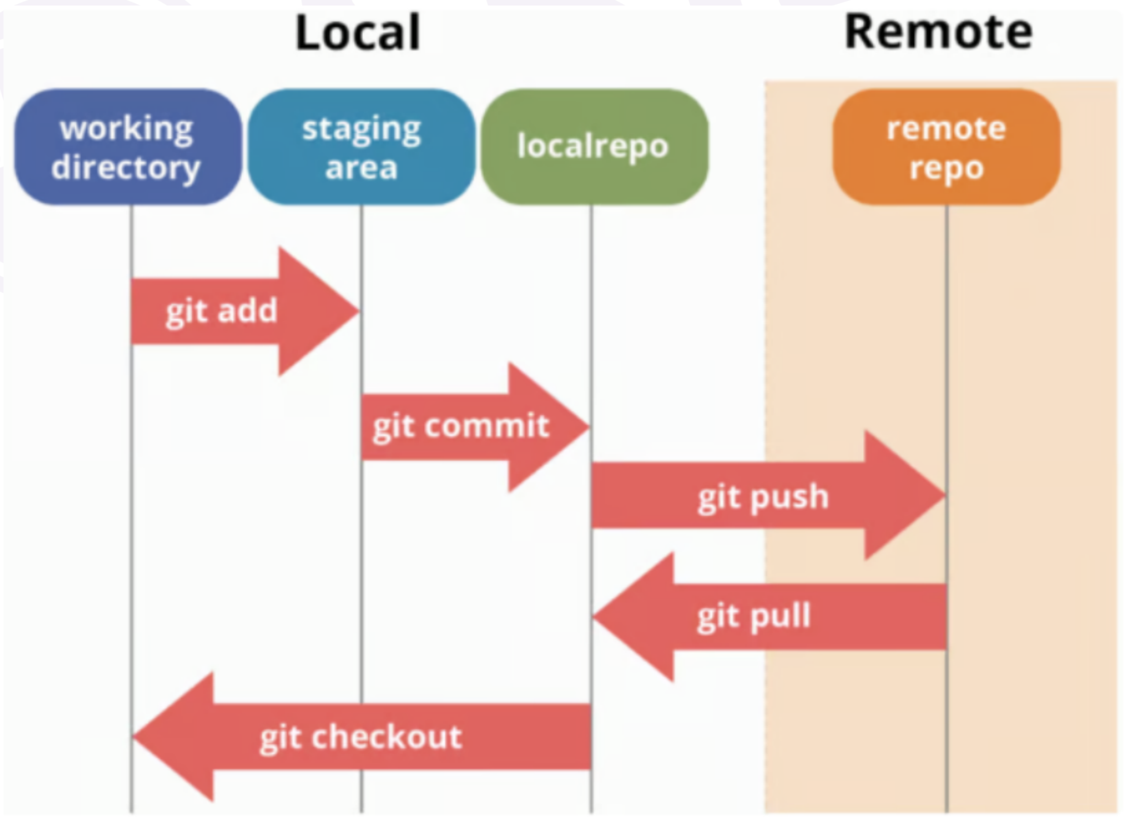
Basic Concepts

Repository
A Git repository (repo) is a collection of files and folders along with their revision history. It contains everything needed to maintain the history of changes to your project.

Commit
A commit is a snapshot of your repository at a specific point in time. It represents a saved change and includes a commit message that describes the modifications made.

Branch
A branch is a parallel version of a repository that diverges from the main line of development (often called the master branch). It allows multiple developers to work on different features simultaneously without affecting the main codebase.

Git Workflows



Git Workflows
Git workflows are designed to help manage the development process and collaboration between team members. There are several common workflows, each with its own advantages and suited to different types of projects and team dynamics. Here are some of the most popular Git workflows:

1. **Centralized Workflow**
2. **Feature Branch Workflow**
3. **Gitflow Workflow**
4. **Forking Workflow**

1. Centralized Workflow
The Centralized Workflow uses a central repository as the single point of collaboration. It is similar to the workflows used by traditional version control systems like Subversion (SVN).

Example:

Clone the central repository:
```
git clone https://github.com/yourorg/repo.git
cd repo
```

Create a new branch to work on a feature:

```
git checkout -b feature_branch
```

Make changes and commit locally:

```
git add .
git commit -m "Add new feature"
```

Push changes to the central repository:

```
git push origin feature_branch
```

Merge changes into the main branch:

```
git checkout main
git pull origin main
git merge feature_branch
git push origin main
```

2. Feature Branch Workflow
The Feature Branch Workflow isolates work on a specific feature in a dedicated branch. This allows multiple developers to work on different features simultaneously without interfering with the main codebase.

Example:

Clone the central repository:

```
git clone https://github.com/yourorg/repo.git
cd repo
```

Create a new branch for the feature:

```
git checkout -b feature_branch
```

Develop the feature and commit changes locally:

```
git add .
git commit -m "Develop new feature"
```

Push the feature branch to the central repository:

```
git push origin feature_branch
```

1. **Create a pull request to merge the feature branch into the main branch:**
   - Go to the repository on GitHub.
   - Click on "Pull requests" and then "New pull request".
   - Select the feature branch as the source and the main branch as the destination.
   - Click "Create pull request".

2. **Review and merge the pull request:**
   - Team members review the code.
   - Once approved, merge the pull request.

3. Gitflow Workflow

The Gitflow Workflow is a branching model designed for managing releases. It introduces two main branches: main for production-ready code and develop for integration. Feature branches, release branches, and hotfix branches are used to manage different types of work.

Example:
Clone the central repository:

```
git clone https://github.com/yourorg/repo.git
cd repo
```

Create a new branch for the feature from develop:

```
git checkout develop
git checkout -b feature_branch
```

Develop the feature and commit changes locally:

```
git add .
git commit -m "Develop new feature"
```

Push the feature branch to the central repository:

```
git push origin feature_branch
git merge release_branch
git push origin main

git checkout develop
git merge release_branch
git push origin develop
```

Create a hotfix branch from main if needed, merge back into main and develop:

```
git checkout main
git checkout -b hotfix_branch
git add .
git commit -m "Fix critical bug"
git push origin hotfix_branch

git checkout main
git merge hotfix_branch
git push origin main

git checkout develop
git merge hotfix_branch
git push origin develop
```

4. Forking Workflow
The Forking Workflow is often used in open-source projects. Developers fork the central repository, make changes in their own fork, and then create a pull request to merge changes back into the central repository.

Example:

1. **Fork the central repository:**
    - Go to the repository on GitHub and click "Fork".

Clone your fork:

```
git clone https://github.com/yourusername/repo.git
cd repo
```

Create a new branch for your changes:
```
git checkout -b my_feature_branch
```

Develop your changes and commit them locally:
```
git add .
git commit -m "Develop new feature"
```

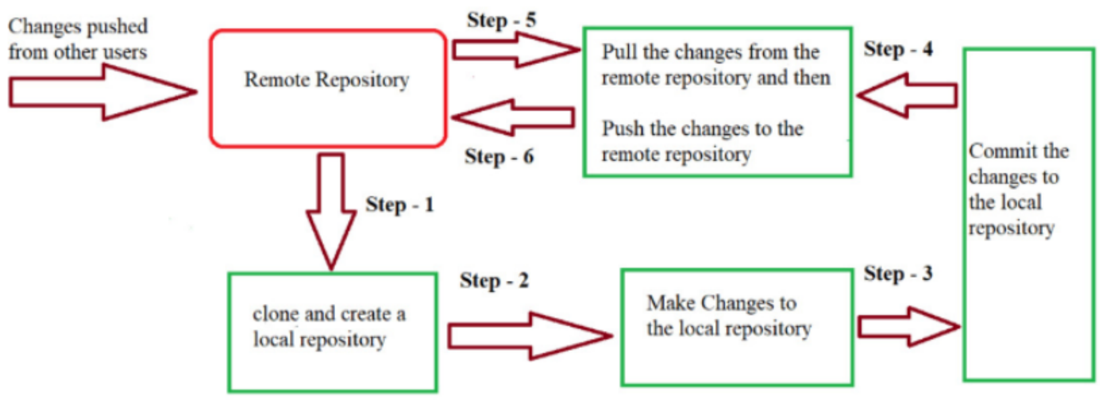
Push your branch to your fork:
```
git push origin my_feature_branch
```

1. **Create a pull request to the central repository:**
   - Go to your fork on GitHub.
   - Click on "Pull requests" and then "New pull request".
   - Select your branch as the source and the central repository's main branch as the destination.
   - Click "Create pull request".

2. **Review and merge the pull request:**
   - Project maintainers review the code.
   - Once approved, they merge the pull request into the central repository.

Lifecycle of a Git

Git is used in our day-to-day work, we use git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Life Cycle that git has and understand more about its life cycle. Let us see some of the basic steps that we follow while working with Git –


Changes pushed from other users
Remote Repository
Step - 5
Pull the changes from the remote repository and then
Push the changes to the remote repository
Step - 6
Step - 4
Commit the changes to the local repository
Step - 1
clone and create a local repository
Step - 2
Make Changes to the local repository
Step - 3

- **In Step – 1**, We first clone any of the code residing in the remote repository to make our own local repository.
- **In Step-2** we edit the files that we have cloned in our local repository and make the necessary changes in it.
- **In Step-3** we commit our changes by first adding them to our staging area and committing them with a commit message.
- **In Step – 4** and Step-5 we first check whether there are any of the changes done in the remote repository by some other users and we first pull that changes.
- If there are no changes we directly proceed with **Step – 6** in which we push our changes to the remote repository and we are done with our work.

Initialization

To start using Git in a project, initialize a Git repository in the project's root directory using git init.

Adding and Committing Changes

Use git add <filename> to stage changes and git commit -m "Commit message" to save them to the repository.

Branching and Merging

Create branches with git branch <branchname> and switch between them using git checkout <branchname>. Merge branches using git merge <branchname>.

Remote Repositories
Collaborate with others by connecting your local repository to remote repositories (git remote add origin <remote-url>) and pushing changes (git push origin <branch>).

Resolving Conflicts
Conflicts may arise when merging branches with conflicting changes. Resolve conflicts by editing the conflicting files and committing the resolved changes.

Tagging Releases
Tag specific commits as releases (git tag -a <tagname> -m "Release message").

Undoing Changes
Use git reset, git revert, or git checkout to undo changes depending on whether the changes have been committed or not.

Conclusion

Understanding Git workflows and the lifecycle of a Git repository is essential for efficient collaboration and version control in software development. Choose a workflow that best suits your project's size, complexity, and team structure to maximize productivity and code quality.