Lesson Plan

Git Hooks and Workflows, Submodules, and Subtrees

PW SKILLS

Git Hooks and Workflows, Submodules, and Subtrees

Introduction
Git is a distributed version control system widely used for source code management (SCM). It allows multiple developers to work on a project simultaneously without interfering with each other. To enhance its capabilities, Git offers several advanced features like hooks, submodules, and subtrees. This document explores these features in detail, providing examples to illustrate their use in real-world scenarios.

Git Hooks
Git hooks are scripts that run automatically at specific points in the Git lifecycle. They allow you to customize and automate Git's internal behavior and trigger customizable actions at key points in your development process.

Types of Git Hooks

1. **Client-side hooks:** These hooks are triggered by operations such as committing and merging. Examples include pre-commit, prepare-commit-msg, commit-msg, post-commit, post-checkout, post-merge, and post-rewrite.
2. **Server-side hooks:** These hooks are triggered by network operations such as receiving pushed commits. Examples include pre-receive, update, and post-receive.

Git Workflows
Git workflows are methodologies for using Git in a collaborative environment. They define how developers interact with the repository and each other.

Common Git Workflows

1. **Centralized Workflow:** All changes are committed to a central repository. This is similar to the traditional SVN workflow.
2. **Feature Branch Workflow:** Developers create separate branches for each feature or task. Once the feature is complete, the branch is merged back into the main branch.
3. **Gitflow Workflow:** This workflow uses feature branches, release branches, and a well-defined branching model to manage releases.
4. **Forking Workflow:** Developers fork the main repository, make changes in their forks, and then create pull requests to merge changes back into the main repository.

Example: Feature Branch Workflow
Create a feature branch:

```
git checkout -b feature/new-feature
```

Make changes and commit them:

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

Push the feature branch to the remote repository:

```
git push origin feature/new-feature
```

1. **Create a pull request:** Once the feature is complete, create a pull request to merge the feature branch into the main branch.
2. **Merge the pull request:** After code review, merge the pull request and delete the feature branch.

Git Submodules
Git submodules allow you to keep a Git repository as a subdirectory of another Git repository. This can be useful for including external libraries or dependencies.

Example: Adding a Submodule
Add a submodule:

```
git submodule add https://github.com/example/library.git path/to/library
```

Initialize and update submodules:

```
git submodule update --init --recursive
```

Clone a repository with submodules:

```
git clone --recursive https://github.com/your/repo.git
```

Git Subtrees
Managing large codebases with multiple dependencies or sub-projects can be a challenging task. Git, the popular version control system, provides several strategies for handling such cases. One effective solution is the use of Git Subtree. Git Subtree allows you to include a repository as a subdirectory of another repository, simplifying the management and integration of external projects. This article will explain what Git Subtree is, its benefits, and how to use it effectively.

Benefits of Using Git Subtree

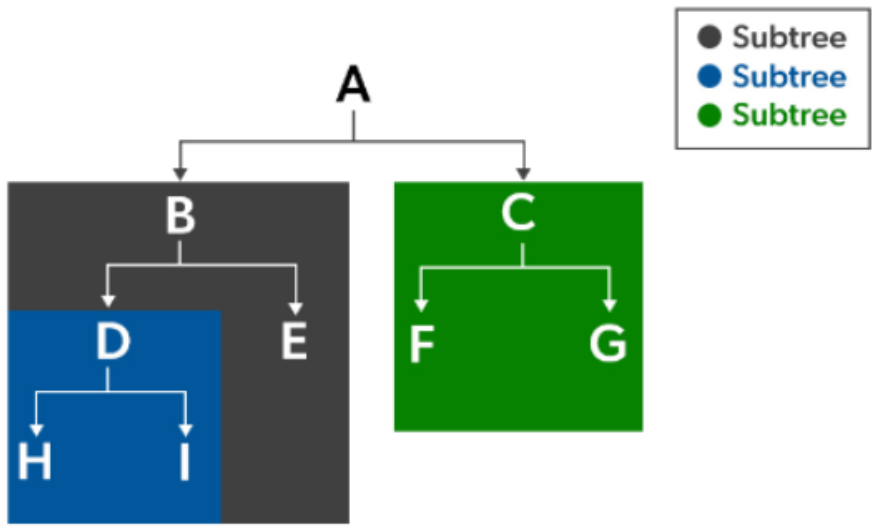
- **Integration:** Seamlessly integrates external repositories into your main project.
- **Simplicity:** Simplifies the workflow compared to Git Submodules by avoiding additional metadata and simplifying commands.
- **Autonomy:** Each subtree can be managed independently, making it easier to track changes and updates.
- **Flexibility:** Allows for merging and splitting subtrees without affecting the main repository.

Advantages of Subtrees

1. Supported by Git's previous version. It supports versions older than 1.5 as well.
2. Workflow management is simple.
3. After the super-project is completed, there will be an available sub-project code.
4. You don't need newer Git knowledge.
5. Content modification without the need for a different repo of the dependence.

Disadvantages of Subtrees

1. It's not immediately obvious that a subtree is used to build the main repo.
2. Your project's subtrees are difficult to list.
3. You can't, at least not simply, list the subtrees' remote repositories.
4. When you change the main repository with subtree commits, then submit the subtree to its main server, and then pull the subtree, the logs can be a little misleading.



Example: Adding a Subtree
Add a subtree:

git subtree add --prefix=path/to/library https://github.com/example/library.git master --squash

Pull updates from the subtree repository:

git subtree pull --prefix=path/to/library https://github.com/example/library.git master --squash

Push changes to the subtree repository:

git subtree push --prefix=path/to/library https://github.com/example/library.git master

Conclusion

Git hooks, workflows, submodules, and subtrees are powerful features that enhance Git's functionality, making it suitable for complex and collaborative development environments. Hooks automate tasks, workflows streamline development processes, submodules manage dependencies, and subtrees integrate external repositories seamlessly. Understanding and utilizing these features can significantly improve your development workflow and productivity.