**Git Essentials: Conflict Resolution, Cherry-Pick, and Stash Commands**

**1. Git Conflicts**

**What is a Git Conflict?**

A **Git conflict** occurs when:

* Two branches have changes to the same line in a file.
* One branch has deleted a file while another has edited it.

Conflicts typically happen in team environments when multiple developers make changes to the same parts of a codebase.

**Resolving Conflicts**

You can resolve conflicts:

* **Manually**: By editing the conflicted files directly.
* **Using 3rd Party Tools**: Tools like TortoiseGit or IDEs such as VSCode provide interfaces to view and resolve conflicts.

To resolve conflicts:

1. Git will show conflict markers (<<<<<<, ======, >>>>>>) in the files with conflicts.
2. Edit the files to select the correct content.
3. Once resolved, stage the changes with git add <file>.
4. Complete the merge with git commit.

**2. Git Cherry-Pick**

**What is Git Cherry-Pick?**

git cherry-pick allows you to apply specific commits from one branch to another. This is useful for applying hotfixes or specific changes without merging entire branches.

**Example Scenario**

You deployed version 1.0.0 to production. Later, developers created version 1.0.1 with new changes. To apply just the hotfix to production, use cherry-pick to selectively apply commits.

**Cherry-Picking Commands**

1. **Apply a Specific Commit**:

git cherry-pick <commit-hash>

Example:

git cherry-pick 272c6130719dee4586046dce68014818246feef2

1. **Undo Cherry-Picked Changes** (if needed):

git revert <commit-hash>

**3. Git Stash**

git stash is used to temporarily save your changes without committing them. This is useful if you need to switch branches or handle an urgent task without losing your current work.

**Common Git Stash Commands**

1. **Stash Your Changes**:

git stash

This saves all current changes and reverts your working directory to match the last commit.

1. **View Stash List**:

git stash list

Each stash will be listed as stash@{N}, where N is the stash number.

1. **Apply the Most Recent Stash**:

git stash apply

This reapplies the most recent stash but keeps it in the stash list.

1. **Apply a Specific Stash**:

git stash apply stash@{1}

1. **Apply and Remove the Most Recent Stash**:

git stash pop

This applies the most recent stash and removes it from the stash list.

1. **Remove a Specific Stash**:

git stash drop stash@{1}

1. **Clear All Stashes**:

git stash clear

**Stash Workflow Example**

Imagine you’re working on a Dockerfile, but you need to handle a high-priority production issue:

1. **Stash Changes**:

git stash

1. **Handle the Production Issue** (e.g., switch branches, make changes).
2. **Return to Your Work**:

git stash pop

This reapplies your previous work on the Dockerfile.

**Summary**

| **Command** | **Description** |
| --- | --- |
| git cherry-pick <commit> | Apply a specific commit from one branch to another. |
| git stash | Save current changes without committing. |
| git stash list | View all stashed changes. |
| git stash apply | Reapply the latest stash without removing it. |
| git stash pop | Reapply and remove the latest stash. |
| git stash drop <stash@{N}> | Remove a specific stash. |
| git stash clear | Clear all stashes. |