Git Stash and Cherry-pick: Resolving Conflicts with Branching

Introduction to Git Stash

Cherry-pick: Selective Committing

Resolving Conflicts

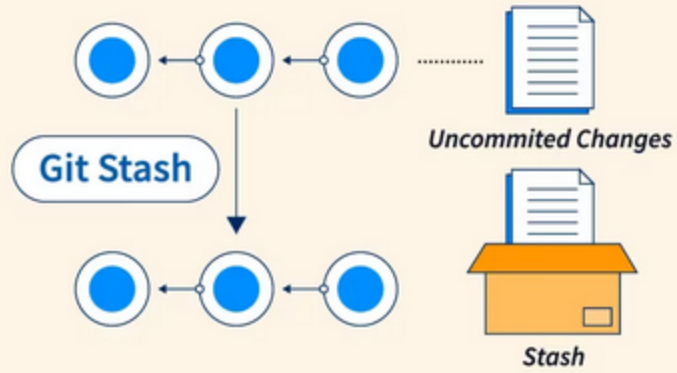
Example: git stash

Example: git cherry-pick

Introduction to Git Stash

Git stash is a useful feature in Git that allows you to temporarily save changes without committing them. This can be helpful when you need to switch to a different branch or work on a different task without losing your current progress.

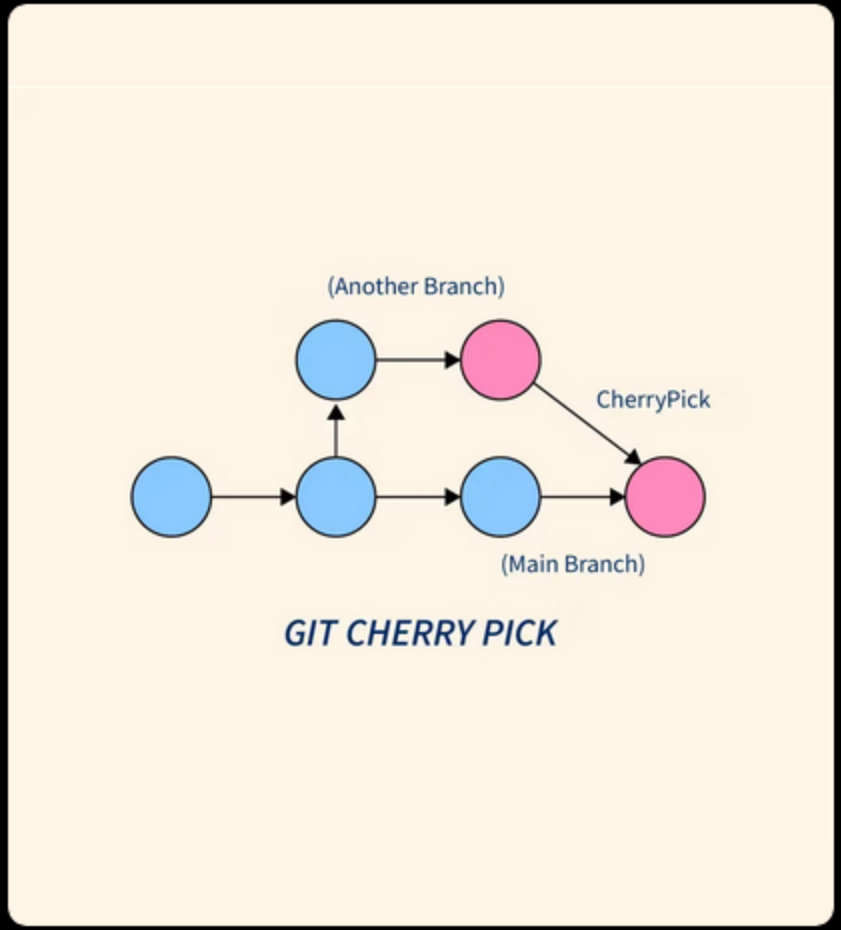
To use git stash, simply run the command `git stash` while you have uncommitted changes. This will save your changes and revert your working directory to the last commit.



Cherry-pick: Selective Committing

Cherry-pick is another useful feature in Git that allows you to choose specific commits from one branch and apply them to another branch. This can be helpful when you want to selectively merge changes from one branch to another.

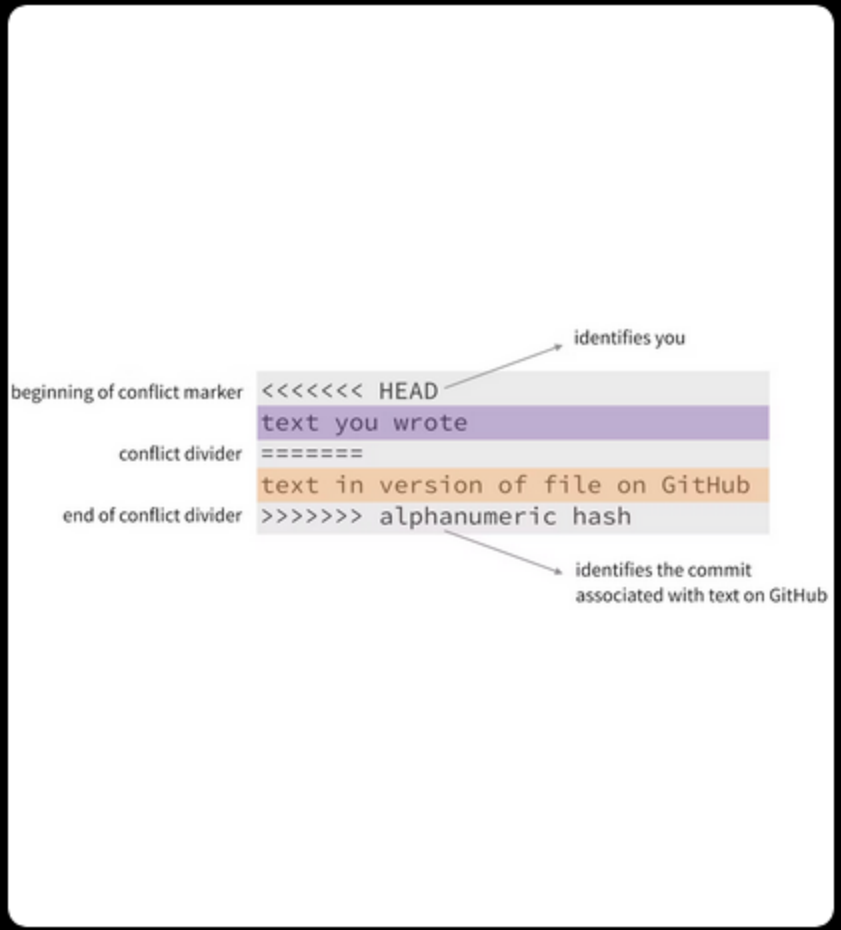
To use cherry-pick, simply run the command `git cherry-pick <commit>` while in the target branch. This will apply the specified commit to the target branch.



Resolving Conflicts

When merging branches or applying cherry-picked commits, conflicts may arise if the same lines of code were modified in both branches. Git provides tools to help resolve these conflicts.

To resolve conflicts, run the command `git mergetool` or manually edit the conflicting files. Once resolved, run `git add <file>` to stage the changes and 'git commit' to finalize the merge.



Example: git stash

1. Creating a new branch in Git allows you to work on a separate copy of the code without affecting the main branch. `git checkout -b <branch_name>`.
2. Make some changes to files like: `echo "new line" » file.txt`
3. Use `git stash` to save the changes without committing them.
4. Now, switch to a different branch and do your work.
5. Afterward, switch back to that branch and use `git stash pop` to bring the changes back and apply them on top of the new commits. You can then continue working on the branch and commit your changes as usual.

Example: git cherry-pick

1. Add some changes to the branch.
2. Commit the changes: `git commit -m "commit message"` .
3. Now get the commit id using `git log –oneline` .
4. After that switch to the branch where you want to apply this commit: `git checkout <branch>` .
5. Use the cherry-pic command to apply the commit : `git cherry-pic <commit id>`
6. Stage the file : `git add .` .
7. Now, continue to cherry-pic: `git cherry-pick --continue` .