UNDERSTANDING GIT REVERT AND GIT RESET: MASTERING VERSION CONTROL TECHNIQUES

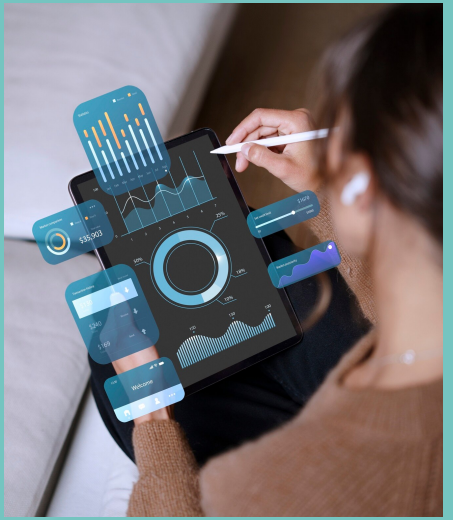
Topics Covered

- Git revert
- Git reset
- Git Cherry pick

GIT REVERT: UNDOING CHANGES

When a mistake is made in a project, **Git Revert** is used to undo specific changes without altering the project's history. This command creates a new commit that undoes the specified changes, making it a safe way to fix errors while preserving the project's integrity.

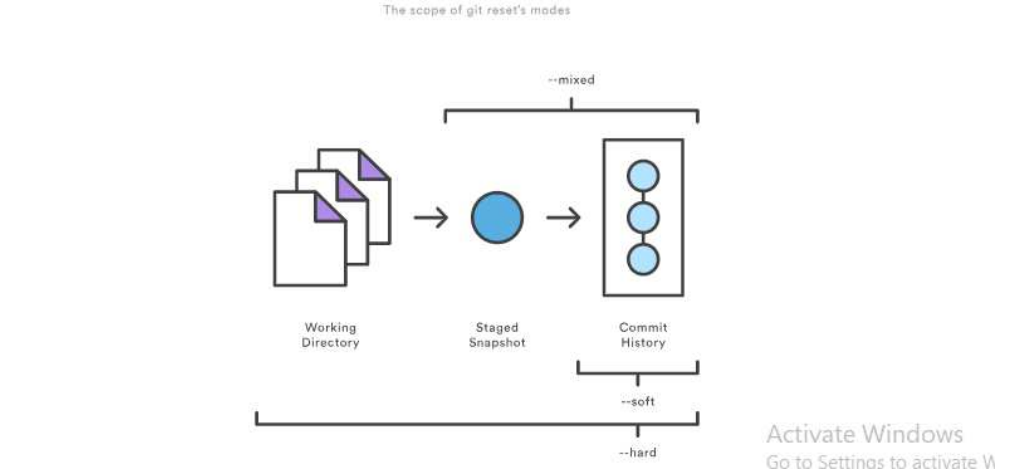
- $ git log --oneline
  86bb32e prepend content to demo file
  3602d88 add new content to demo file
  299b15f initial commit
- $ git revert HEAD
  [main b9cd081] Revert "prepend content to de
  mo file" 1 file changed, 1 deletion(-)
- $ git log --
  oneline 1061e79 Revert "prepend content to de
  mo file" 86bb32e prepend content to demo file
  3602d88 add new content to demo file 299b15f
  initial commit



GIT RESET: REWRITING HISTORY

Git Reset is a powerful command that allows developers to rewrite the project's history. By using different options such as *soft*, *mixed*, or *hard*, developers can reset the project to a specific commit, removing unwanted changes and restoring the project to a previous state.

The default invocation of `git reset` has implicit arguments of `--mixed` and `HEAD`. This means executing `git reset` is equivalent to executing `git reset --mixed HEAD`. In this form `HEAD` is the specified commit. Instead of `HEAD` any Git SHA-1 commit hash can be used.

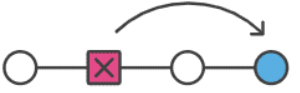


```
$ git reset --hard
HEAD is now at dc67808 update content of reset_lifecycle_file
$ git status
On branch main
nothing to commit, working tree clean
$ git ls-files -s
100644 d7d77c1b04b5edd5acfc85de0b592449e5303770 0 reset_lifecycle_file
```

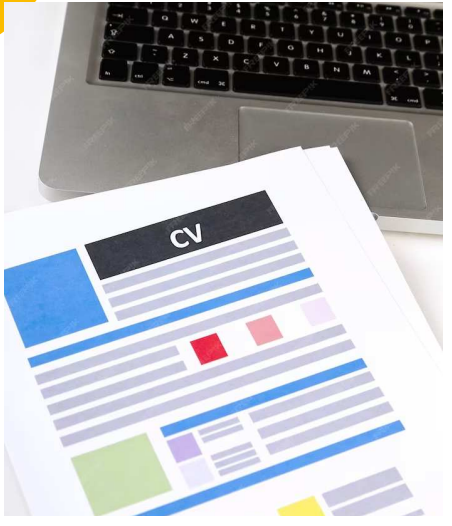
```
$ git reset --soft
$ git status
On branch main
Changes to be committed:
    (use "git reset HEAD ..." to unstage)

modified: reset_lifecycle_file
$ git ls-files -s
100644 67cc52710639e5da6b515416fd779d0741e3762e 0 reset_lifecycle_file
```

```
$ git reset --mixed
$ git status
On branch main
Changes not staged for commit:
    (use "git add ..." to update what will be committed)
    (use "git checkout -- ..." to discard changes in working directory)

modified: reset_lifecycle_file

Untracked files:
    (use "git add ..." to include in what will be committed)

new_file


no changes added to commit (use "git add" and/or "git commit -a")
$ git ls-files -s
100644 d7d77c1b04b5edd5acfc85de0b592449e5303770 0 reset_lifecycle_file
```

Reverting



Resetting



BEST PRACTICES FOR USING GIT REVERT AND GIT RESET

To master version control techniques, it's important to follow best practices when using **Git Revert** and **Git Reset**. Always create a backup branch before making significant changes, and communicate with the team to ensure everyone is aware of the modifications.

Git cherry pick

- git cherry-pick is a powerful command that enables arbitrary Git commits to be picked by reference and appended to the current working HEAD.
- Cherry picking is the act of picking a commit from a branch and applying it to another. git cherry-pick can be useful for undoing changes.
- For example, say a commit is accidently made to the wrong branch. You can switch to the correct branch and cherry-pick the commit to where it should belong.

Bug hot fixes:

- When a bug is discovered it is important to deliver a fix to end users as quickly as possible. For an example scenario,say a developer has started work on a new feature.
- During that new feature development they identify a pre-existing bug. The developer creates an explicit commit patching this bug. This new patch commit can be cherry-picked directly to the main branch to fix the bug before it effects more users.

To demonstrate how to use `git cherry-pick` let us assume we have a repository with the following branch state:

```
a - b - c - d   Main
     \
      e - f - g Feature
```

`git cherry-pick` usage is straight forward and can be executed like:

```
git cherry-pick commitSha
```

In this example `commit` Sha is a commit reference. You can find a commit reference by using `git log`. In this example we have constructed lets say we wanted to use commit `f` in `main`. First we ensure that we are working on the `main` branch.


```
git checkout main
```

Then we execute the cherry-pick with the following command:

```
git cherry-pick f
```

Once executed our Git history will look like:

```
a - b - c - d - f   Main
     \
      e - f - g Feature
```

The f commit has been successfully picked into the main branch

CONCLUSION: MASTERING VERSION CONTROL WITH GIT

Mastering Git Revert and Git Reset is essential for effectively managing code changes and maintaining a clean project history. By understanding the nuances of these version control techniques, developers can confidently navigate through complex development scenarios and collaborate seamlessly with their teams.

Thanks!