Assignment Concepts - GIT

Using git reset and git revert with Git Bash Terminal

Example Scenario: Creating a File and Observing Changes

Setup Repository:

```
$ mkdir example-repo
$ cd example-repo
$ git init
```

Create a File and Make Initial Commit:

```
$ echo "Hello, World!" > example.txt
$ git add example.txt
$ git commit -m "Initial commit"
```

****

Using git reset:

****

Make Changes and Commit:

```
$ echo "This is the second line." >> example.txt
$ git add example.txt
$ git commit -m "Added second line"
```

Check Commit History:

```
$ git log --oneline
```

Example output:

```
2f3e4d5 Added second line
1a2b3c4 Initial commit
```

Reset to Previous Commit:

```
$ git reset --hard 1a2b3c4
```

The --hard option resets the index and working directory to match the specified commit.

Observe Changes:

Check the commit history.

```
$ git log --oneline
```

Check the contents of the file.

```
$ cat example.txt
```

The file should only contain "Hello, World!" since the reset removed the last commit and its changes.

****

Using git revert

****

Make Changes and Commit Again:

```
$ echo "This is the second line." >> example.txt
$ git add example.txt
$ git commit -m "Added second line"
```

Check Commit History:

```
$ git log --oneline
```

Example output:

```
3e4f5g6 Added second line
1a2b3c4 Initial commit
```

Revert the Last Commit:

```
$ git revert 3e4f5g6
```

This will create a new commit that undoes the changes from the specified commit.

Observe Changes:

Check the commit history.

```
$ git log --oneline
```

Example output:

```
7h8i9j0 Revert "Added second line"
3e4f5g6 Added second line
1a2b3c4 Initial commit
```

Check the contents of the file.

```
$ cat example.txt
```

The file should only contain "Hello, World!" since the revert commit undid the changes from the "Added second line" commit.

Differences Between git reset and git revert

History Alteration:

git reset changes the commit history by removing commits. It can reset the working directory to a previous state.

git revert preserves the commit history by creating a new commit that undoes the changes of a specified commit.

Usage Context:

git reset is typically used when you want to discard commits and start over, usually in a private branch or local repository.

git revert is safer for shared repositories as it doesn't remove commits but rather creates new commits to negate the changes.

Undoing Changes:

git reset --hard discards all changes in the working directory and index to match the specified commit.

git revert keeps the commit history intact by creating a new commit that negates a previous commit's changes.

Collaboration:

git reset can disrupt other collaborators by rewriting commit history.

git revert is collaboration-friendly as it maintains the complete history, making it easy to track changes and understand the repository's evolution.

Git squash is a technique used to combine multiple commits into a single commit. This is useful for tidying up commit history before merging a feature branch into the main branch. The git rebase command is typically used to squash commits.

Example Scenario: Creating and Squashing Commits

Setup Repository:

```
$ mkdir squash-repo

$ cd squash-repo

$ git init
```

Create a File and Make Initial Commit:

```
$ echo "Hello, World!" > example.txt

$ git add example.txt

$ git commit -m "Initial commit"
```

Make Additional Commits:

```
$ echo "First change." >> example.txt

$ git add example.txt

$ git commit -m "First change"


$ echo "Second change." >> example.txt

$ git add example.txt
$ git commit -m "Second change"
```

```
$ echo "Third change." >> example.txt
$ git add example.txt
$ git commit -m "Third change"
```

Check Commit History:

```
$ git log --oneline
```

Example output:

```
4d3e2f1 Third change
3c2b1a0 Second change
2b1a0f9 First change
1a2b3c4 Initial commit
```

Using git rebase -i to Squash Commits

Start Interactive Rebase:

To squash the last three commits into one, start an interactive rebase from the parent of the first commit you want to squash.

```
$ git rebase -i HEAD~3
```

Interactive Rebase Interface:

This will open an editor with the following content:

plaintext

Copy code

```
pick 2b1a0f9 First change
pick 3c2b1a0 Second change
pick 4d3e2f1 Third change
```

Edit the Commit Actions:

Change the second and third pick to squash (or s):

plaintext

Copy code

```
pick 2b1a0f9 First change

squash 3c2b1a0 Second change

squash 4d3e2f1 Third change
```

Save and Close the Editor:

Save the file and close the editor to continue the rebase process. Git will combine the commits and prompt you to edit the commit message.

Edit the Commit Message:

An editor will open to combine commit messages. Edit it to a single message, if needed:

```
First change


# This is a combination of 3 commits.

# The first commit's message is:

#

# First change

#

# The second commit's message is:

#

# Second change

#

# The third commit's message is:

#

# Third change
```

Save and Close the Editor:

Save the file and close the editor to complete the rebase.

Complete the Rebase:

If there are no conflicts, the rebase will complete. If there are conflicts, resolve them and continue the rebase.

```
$ git rebase --continue
```

Check the Squashed Commit:

```
$ git log --oneline
```

Example output:

```
9e8f7g6 First change
1a2b3c4 Initial commit
```

The last three commits have been squashed into one.

Differences Between git squash, git reset, and git revert

History Alteration:

git reset changes the commit history by removing commits. It can reset the working directory to a previous state.

git revert preserves the commit history by creating a new commit that undoes the changes of a specified commit.

git squash combines multiple commits into a single commit, making the commit history cleaner.

Usage Context:

git reset is typically used to discard commits and start over, usually in a private branch or local repository.

git revert is safer for shared repositories as it doesn’t remove commits but rather creates new commits to negate the changes.

git squash is used during interactive rebase to clean up commit history before merging a feature branch.

Undoing Changes:

git reset --hard discards all changes in the working directory and index to match the specified commit.

git revert keeps the commit history intact by creating a new commit that negates a previous commit’s changes.

git squash merges the changes from multiple commits into one, simplifying the commit history.

Collaboration:

git reset can disrupt other collaborators by rewriting commit history.

git revert is collaboration-friendly as it maintains the complete history, making it easy to track changes and understand the repository's evolution.

git squash is useful for making a series of changes look like a single commit, which is helpful before merging branches, but it should be done carefully in a collaborative environment to avoid conflicts.

By
Kastro Kiran V

LinkedIn: https://www.linkedin.com/in/kastro-kiran/

YouTube: https://www.youtube.com/playlist?list=PLs-PsDpuAuTdOcZa-DDgG8KRbtMI_XRrC