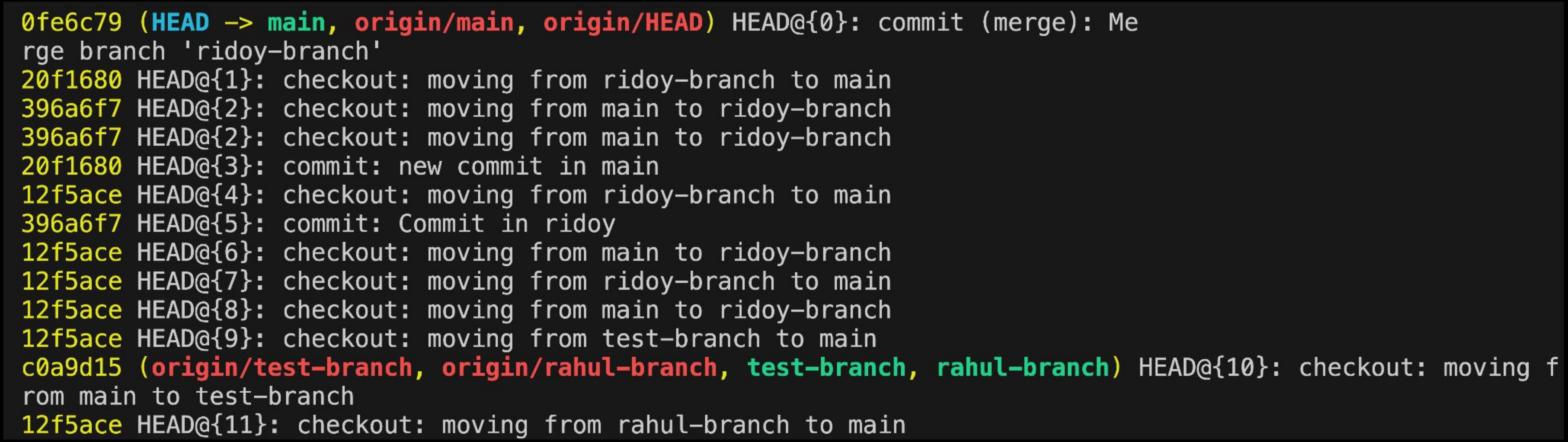
6. Advance Git Operations

1. git reflog
2. git blame
3. git stash
4. git revert
5. git reset
6. git commit --amend
7. git cherry-pick
8. git rebase
9. Git rebase interactive
10. Keeping a clean history

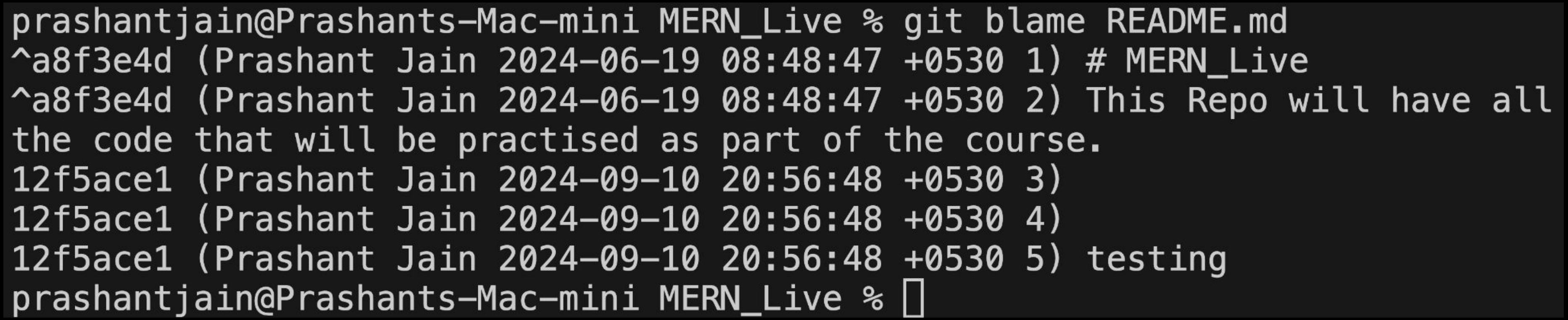
6.1 git reflog



```
0fe6c79 (HEAD -> main, origin/main, origin/HEAD) HEAD@{0}: commit (merge): Me
rge branch 'ridoy-branch'
20f1680 HEAD@{1}: checkout: moving from ridoy-branch to main
396a6f7 HEAD@{2}: checkout: moving from main to ridoy-branch
396a6f7 HEAD@{2}: checkout: moving from main to ridoy-branch
20f1680 HEAD@{3}: commit: new commit in main
12f5ace HEAD@{4}: checkout: moving from ridoy-branch to main
396a6f7 HEAD@{5}: commit: Commit in ridoy
12f5ace HEAD@{6}: checkout: moving from main to ridoy-branch
12f5ace HEAD@{7}: checkout: moving from ridoy-branch to main
12f5ace HEAD@{8}: checkout: moving from main to ridoy-branch
12f5ace HEAD@{9}: checkout: moving from test-branch to main
c0a9d15 (origin/test-branch, origin/rahul-branch, test-branch, rahul-branch) HEAD@{10}: checkout: moving f
rom main to test-branch
12f5ace HEAD@{11}: checkout: moving from rahul-branch to main
```

- Purpose: Shows the history of where HEAD has pointed.
- Usage: Ideal for undoing resets or finding lost commits.

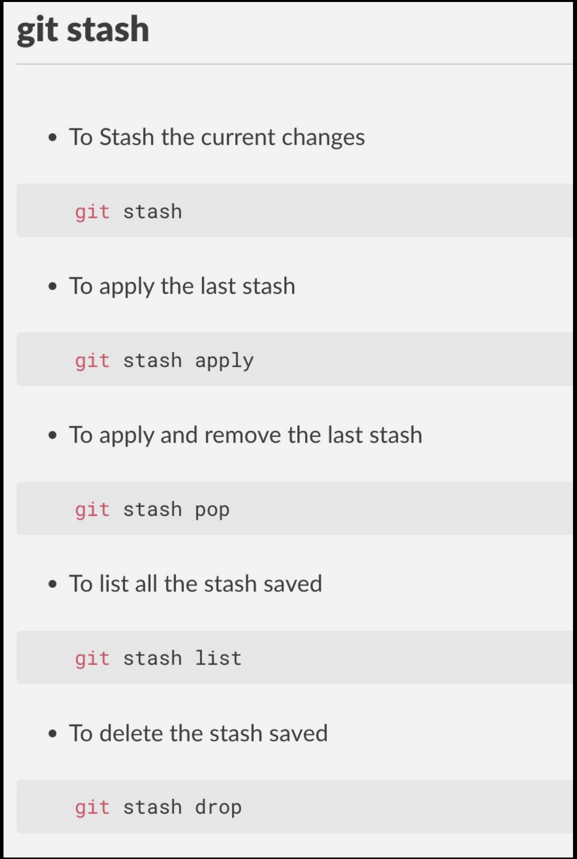
6.2 git blame



```
prashantjain@Prashants-Mac-mini MERN_Live % git blame README.md
^a8f3e4d (Prashant Jain 2024-06-19 08:48:47 +0530 1) # MERN_Live
^a8f3e4d (Prashant Jain 2024-06-19 08:48:47 +0530 2) This Repo will have all
the code that will be practised as part of the course.
12f5ace1 (Prashant Jain 2024-09-10 20:56:48 +0530 3)
12f5ace1 (Prashant Jain 2024-09-10 20:56:48 +0530 4)
12f5ace1 (Prashant Jain 2024-09-10 20:56:48 +0530 5) testing
prashantjain@Prashants-Mac-mini MERN_Live %
```

- Purpose: Shows who last modified each line in a file, along with the commit details.
- Usage: Useful for identifying responsibility for code changes.

6.3 git stash



git stash

- To Stash the current changes

```
git stash
```

- To apply the last stash

```
git stash apply
```

- To apply and remove the last stash

```
git stash pop
```

- To list all the stash saved

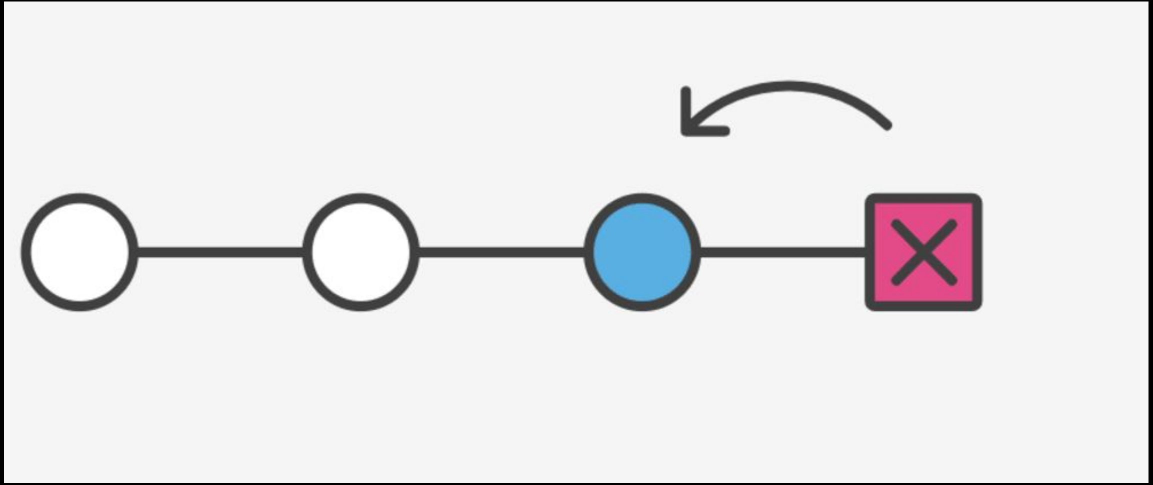
```
git stash list
```

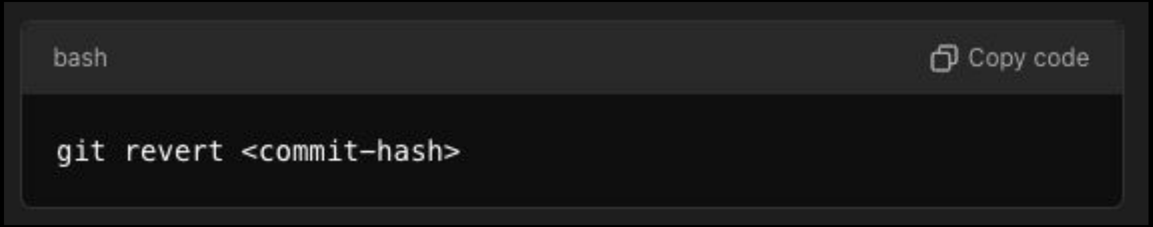
- To delete the stash saved

```
git stash drop
```

- Purpose: Temporarily saves your changes (working directory and index) and returns your workspace to a clean state.
- Useful when switching branches.
- Ideal for saving incomplete work without committing it.

6.4 git revert

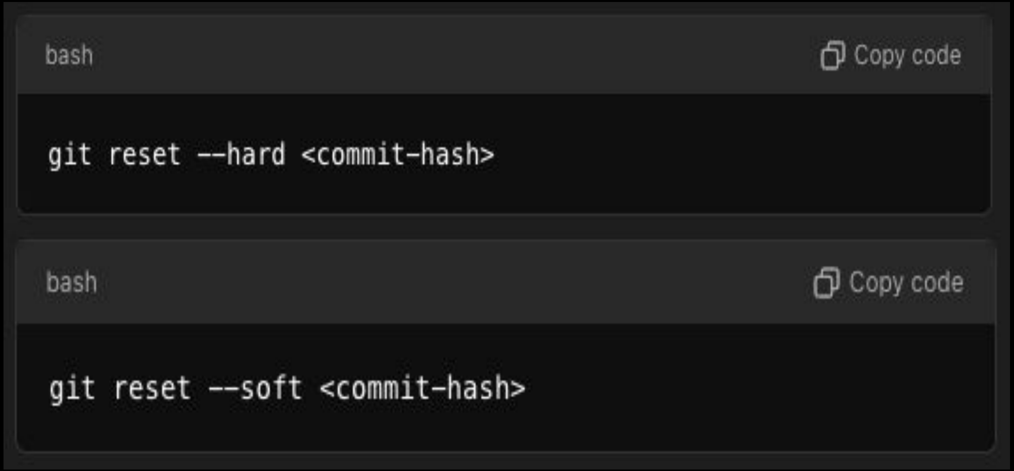




```bash
git revert <commit-hash>
```

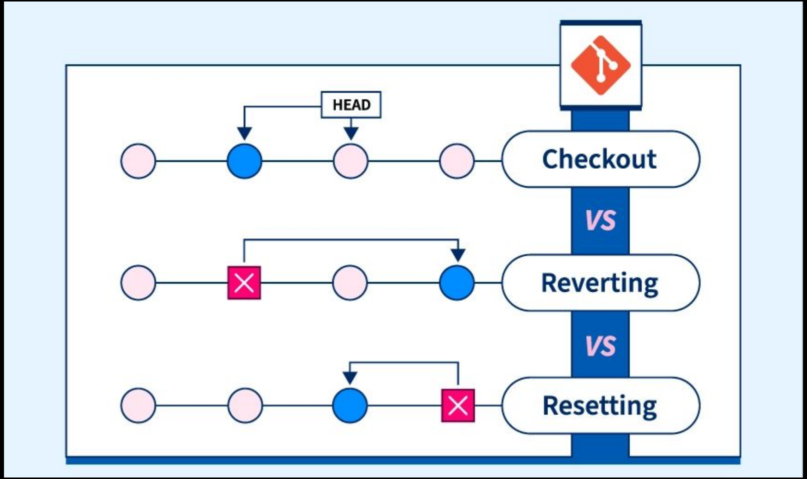
Reverts a commit by creating a new commit that undoes the changes introduced by a specific commit, without altering the commit history.

6.5 git reset



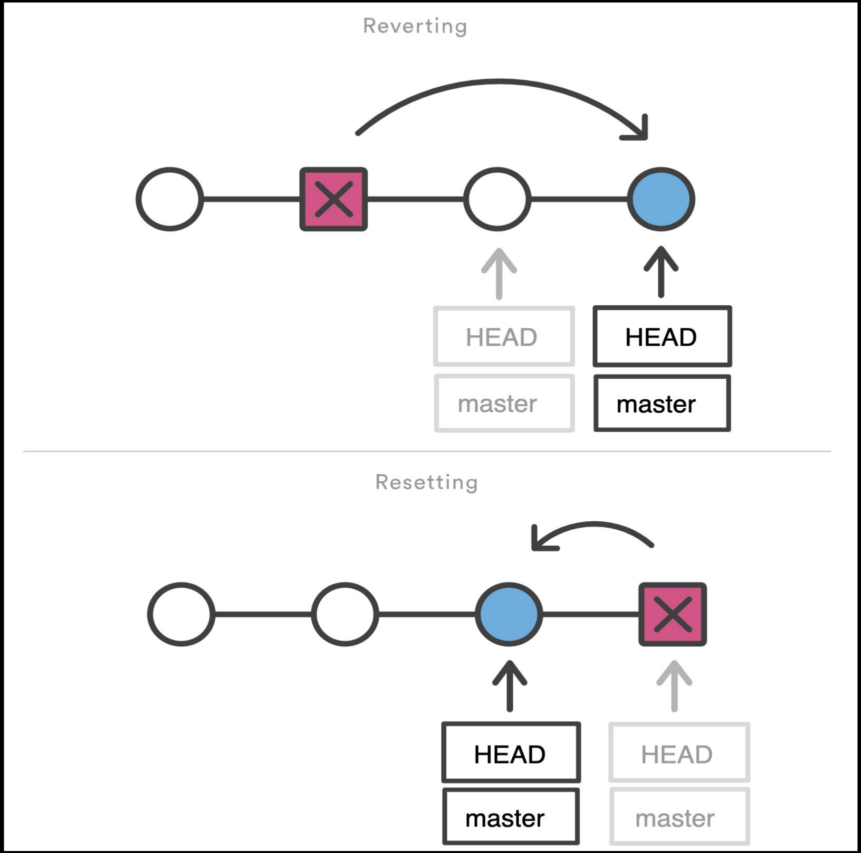
```bash
git reset --hard <commit-hash>
```

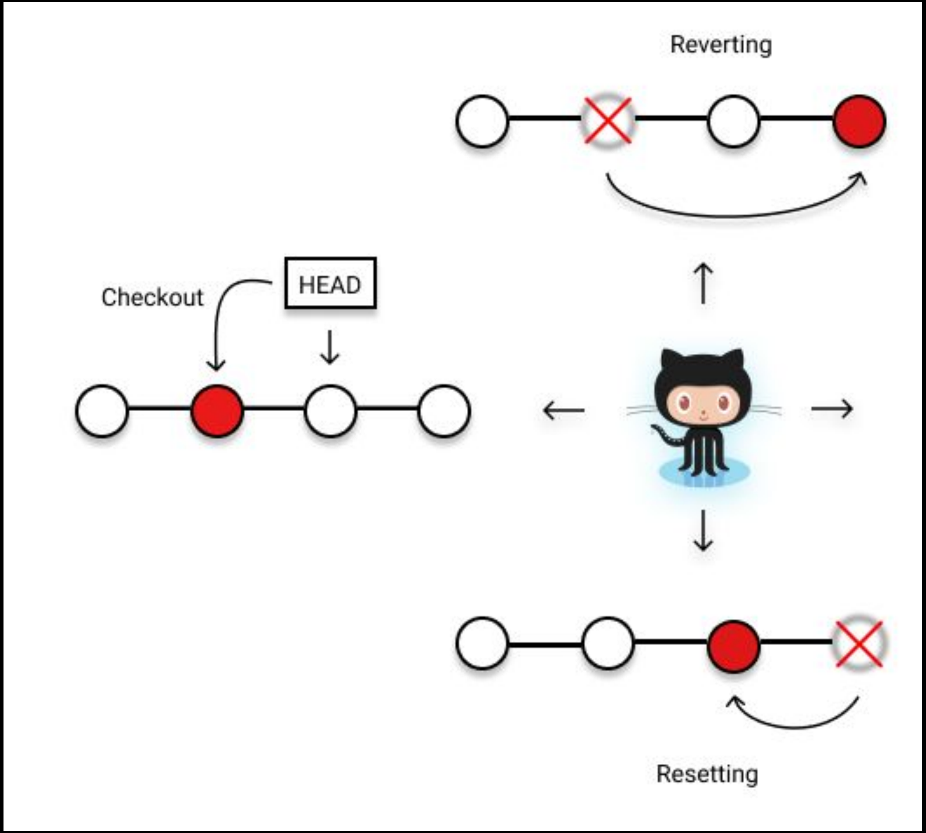
```bash
git reset --soft <commit-hash>
```



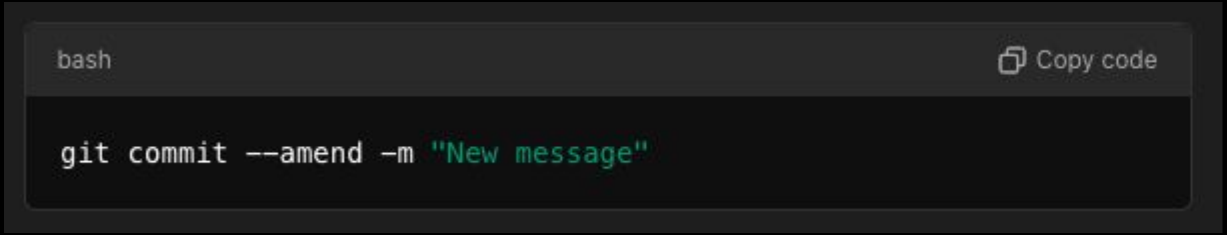
- Purpose: Moves the current branch HEAD to a specific commit.
- Mode supported (--soft, --mixed, --hard).
- Usage: Used to undo changes in commits, index, or working directory.

6.5 git reset vs revert





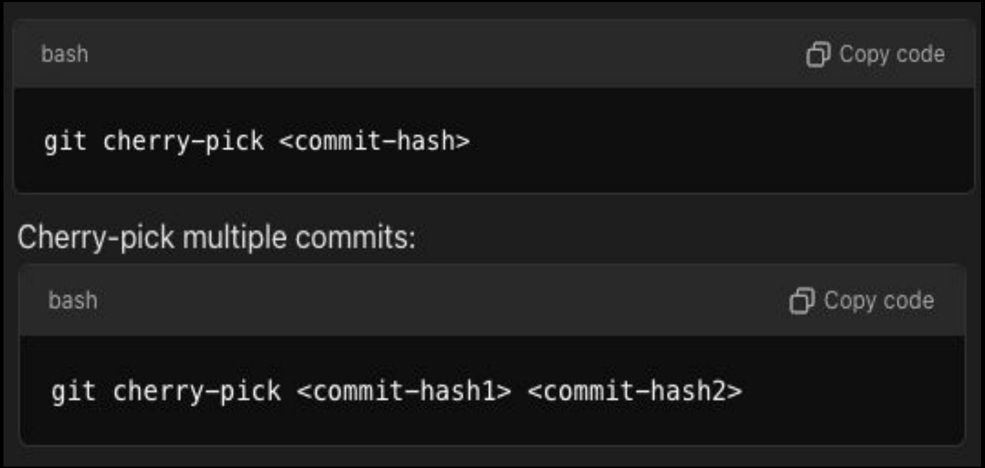
6.6 git commit --amend



```bash
git commit --amend -m "New message"
```

- While git commit --amend does change history, it only changes the most recent commit on your current branch.
- Modify the Last Commit: Edit the last commit's message or content.
- Fix Mistakes: Useful for correcting typos or adding forgotten changes.
- Replaces the Commit: Does not create a new commit, just replaces the previous one.
- Avoid with Pushed Commits: Should only be used for local commits to avoid history conflicts.

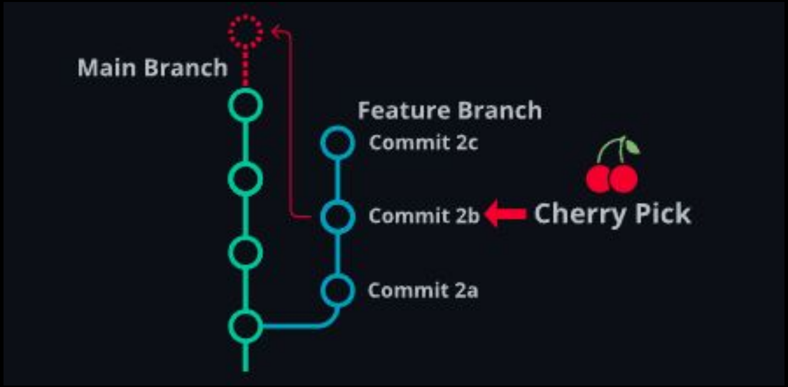
6.7 git cherry-pick



```bash
git cherry-pick <commit-hash>
```

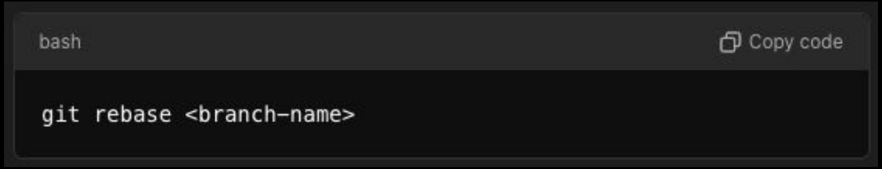
Cherry-pick multiple commits:

```bash
git cherry-pick <commit-hash1> <commit-hash2>
```



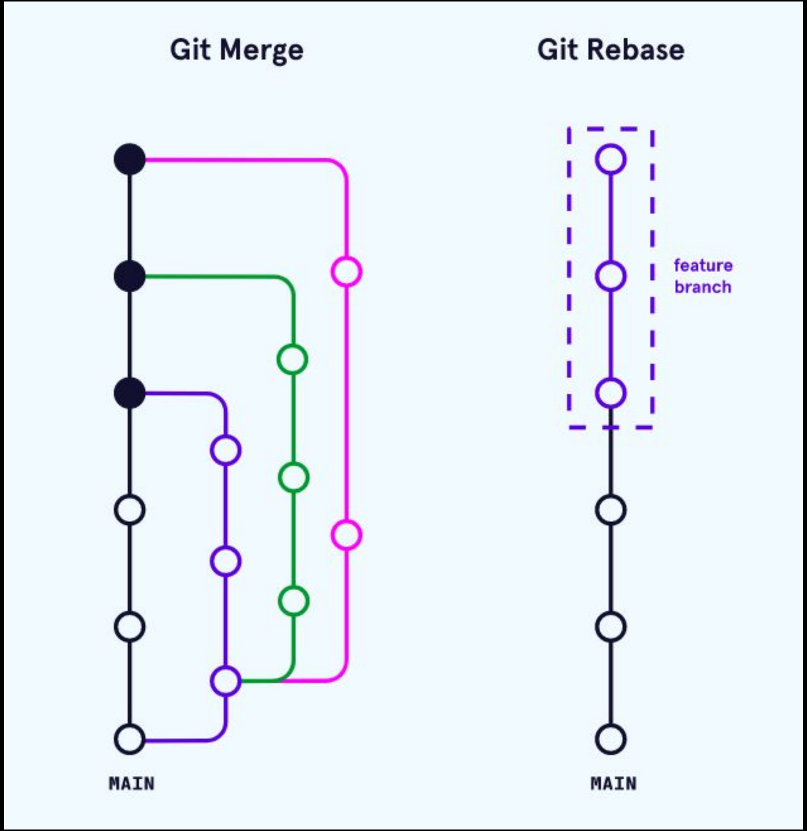
- Purpose: Applies changes from specific commits in one branch to another branch.
- Useful when you want to move specific features without merging everything.
- Usage: Great for bringing in specific features from another branch.

6.8 git rebase

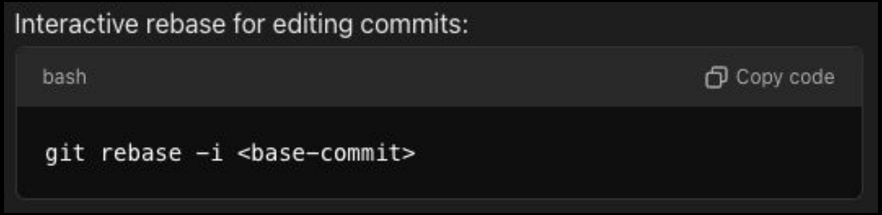


```bash
git rebase <branch-name>
```

- Purpose: Reapplies commits on top of another base commit.
- Can be used to clean up commit history and incorporate changes from one branch into another.



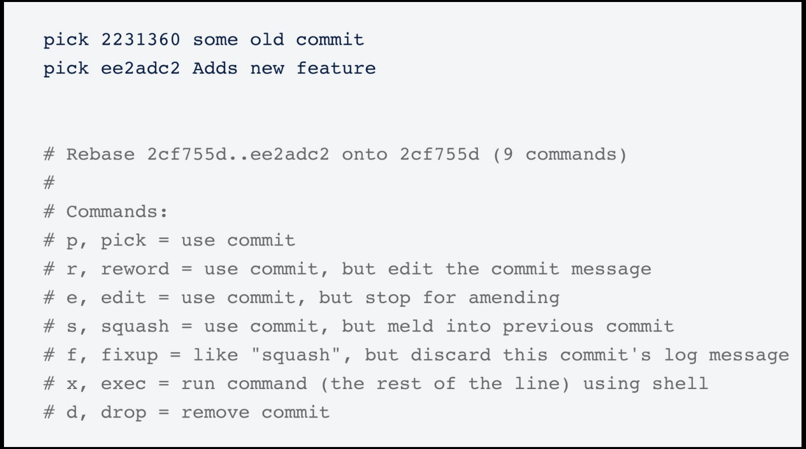
6.9 git rebase -i


Interactive rebase for editing commits:

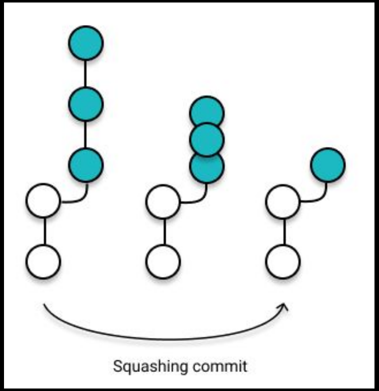
bash Copy code

```bash
git rebase -i <base-commit>
```

- Rearrange or Edit Commits: Allows you to modify, reorder, squash, or remove commits in a branch to clean up the commit history.
- Useful for History Cleanup: Ideal for refining commits before sharing or pushing, creating a cleaner and more understandable commit history.
- This changes history: Do not do this after pushing.

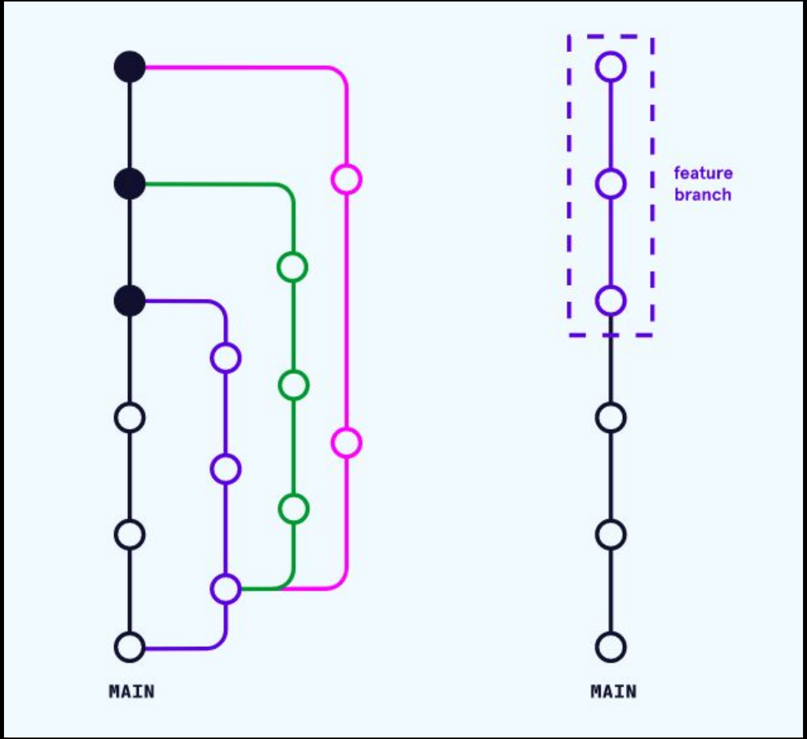

```
pick 2231360 some old commit
pick ee2adc2 Adds new feature


# Rebase 2cf755d..ee2adc2 onto 2cf755d (9 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
```



6.10 Keeping a clean history

- Avoid messy, redundant commits; use tools like git rebase to clean up history.
- Squash multiple small or meaningless commits into one meaningful commit.
- Use descriptive commit messages to keep the history readable and easy to follow.
- Regularly review and tidy up branches before merging into the main branch.


feature branch
MAIN
MAIN