Day 0.1
21 September 2023
09:57

Git prune

-the git fetch command is used to retrieve changes from a remote repository to your local repository.

-In simple terms, when you use Git to work on a project, you often connect to a central online place where all the project files are stored. This central place is called a "remote repository."

-When you want to get the latest updates from this central place, you use the "git fetch" command. It's like checking if there are any new changes made by your team members.

-Now, sometimes your team members might delete some of their work from this central place. But when you use "git fetch," your computer doesn't always realize that things have been deleted on the central place. It just adds new stuff.

-Here's where "pruning" comes in. When you use "git fetch" with the "prune" option (like saying "git fetch --prune"), it not only gets new stuff but also checks if anything has been deleted on the central place. If it finds things deleted there, it cleans up your computer and removes any leftovers that are no longer needed.

-So, "git fetch --prune" is like making sure your computer stays clean and up-to-date with the central place, even if some things were deleted there.
-Command for git Prune:-**git fetch --prune** or u can use **git fetch -p**

Squash

- "squash" refers to the process of combining multiple commits into a single, consolidated commit.

-Squashing commits can help make your commit history more readable and easier to understand for yourself and your team members. It's particularly useful when you're working on a feature or bug fix and have made several commits during the development process, but you want to present a cleaner history before merging your changes into the main branch.

1.**Start an Interactive Rebase**: To begin squashing commits, you'll typically use an interactive rebase. Open your terminal, navigate to your Git repository, and run the following command:

Command-git rebase -i HEAD~n
Replace **n** with the number of commits you want to squash. For **example**, if **you want to squash the last three commits**, use **HEAD~3**.
2.**Interactive Rebase Editor:** The above command will open an interactive rebase editor, displaying a list of commits in your default text editor (e.g., Vim, Nano, or a specified editor).
The list will look something like this:

```
pick abc123 First Commit
pick def456 Second Commit
 pick ghi789 Third Commit
```

3.**Squash Commits:** To squash commits, change the word "pick" to "squash" (or simply "s" for short) for all but the first commit. The first commit will be the one that the others are squashed into.

```
pick abc123 First Commit
 squash def456 Second Commit
 squash ghi789 Third Commit
```

Save and close the editor.
4.Edit the Squash Message: Git will prompt you to edit the commit message for the new, consolidated commit. The text editor will display something like this:

```
# This is a combination of 3 commits.
 # The first commit's message is: First Commit
 # This is the 2nd commit message: Second Commit
# This is the 3rd commit message: Third Commit
# Please enter the commit message for your changes. Lines starting # with '#' will be ignored, and an empty message aborts the commit.
#
 # Date: …
 #
 # squashed commit message here
```

Modify the commit message to provide a concise and meaningful summary of the changes introduced by the squashed commits. Delete any unwanted lines, and save and close the file.

5.Complete the Rebase: Git will finalize the rebase process by squashing the selected commits into one and applying the new commit message. If there are any conflicts during the rebase, Git will pause the process and allow you to resolve the conflicts before proceeding.

6.Push Changes (if needed): If you've already pushed the original commits to a remote repository, you may need to force push **(git push --force)** the branch to update the remote branch with the squashed commit.