KOSS PROJECT

NEW GIT CONCEPTS

WHY DO WE NEED GIT !

GIT

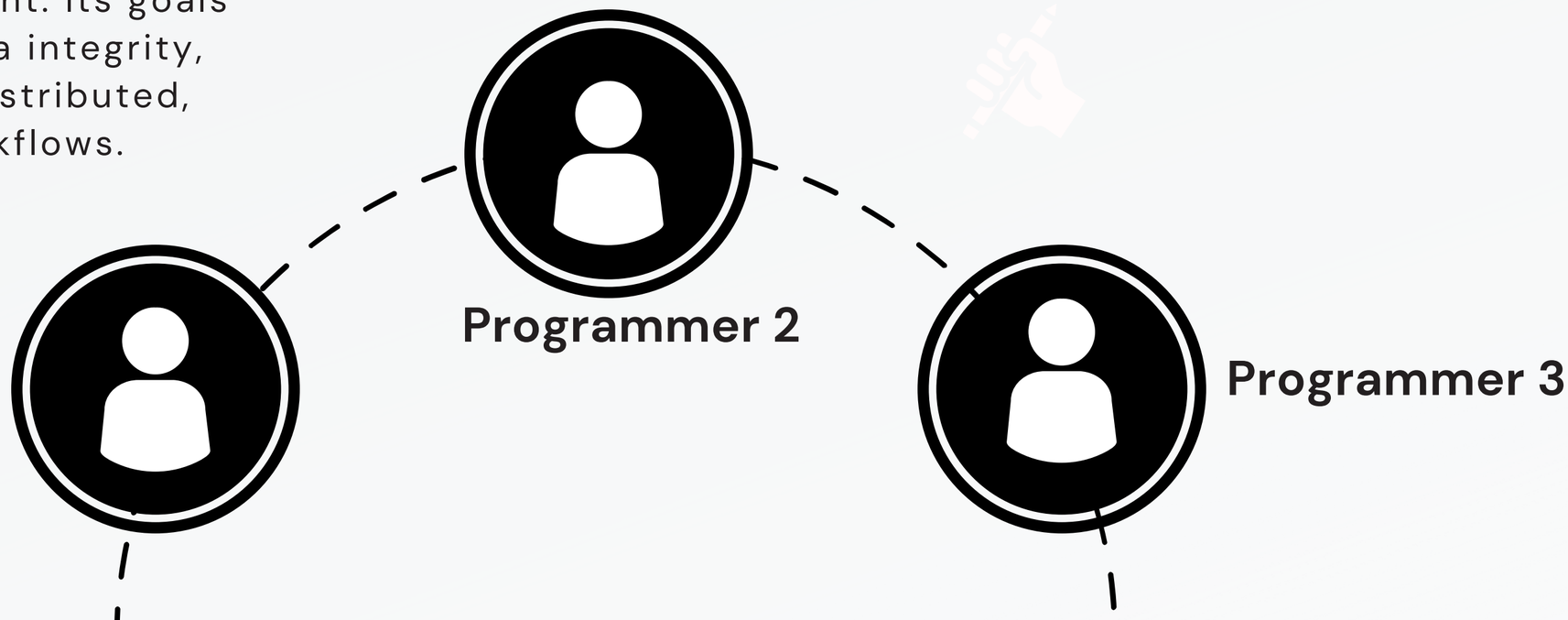
Git is a distributed version control system that tracks changes in any set of computer files, usually used for developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Working

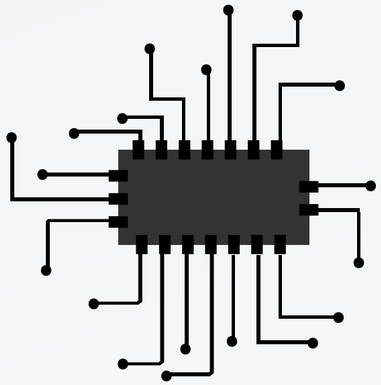
It works as a distributed version control so that the whole codebase including entire history is mirrored in every programmers's computer

Some more

With the use of git programmers are connected easily and they can make a project together

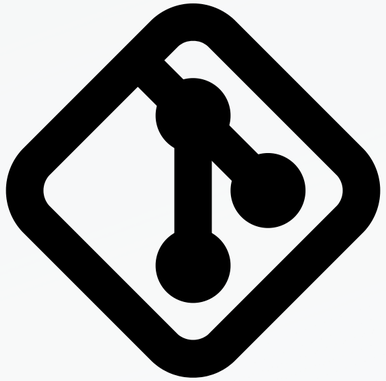


GIT STASH

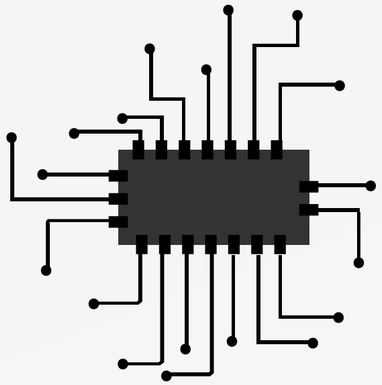


- Version control is essential in software development for tracking changes and collaborating effectively.
- Git Stash is a valuable feature that helps manage temporary changes during development.
- It is a command in Git that allows you to save changes that are not ready to be committed.
- It provides a way to store your modifications temporarily, so you can switch branches or pull updates without committing unfinished work.

- To use Git Stash, you simply run the command "git stash" in your terminal or Git client.
- This command will save your changes to a temporary area, effectively resetting your working directory to the last committed state.
- If you want to provide a name for the stash, you can use the command "git stash save "stash_name"".
- To view the list of stashes you have created, you can use the command "git stash list".

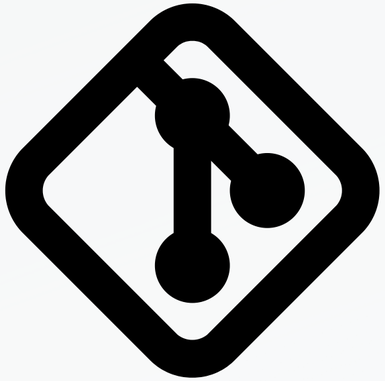


GIT BISECT

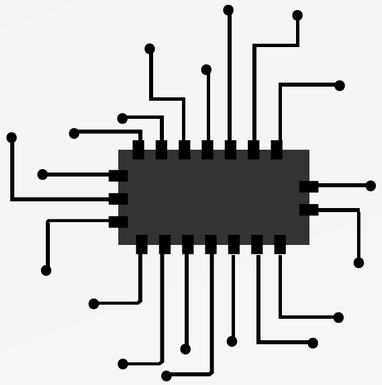


- Git Bisect is a powerful tool in Git used to find the specific commit that introduced a bug or issue in your codebase.
- It uses a binary search algorithm to efficiently narrow down the problematic commit.
- With each iteration, Git Bisect halves the search space until it identifies the culprit commit.
- Based on the test result, you mark the commit as "good" if the bug is not present or "bad" if the bug is present.

- To start run git bisect start
- Mark a commit as "good": = git bisect good <commit>
- Mark a commit as "bad": = git bisect bad <commit>
- Git Bisect will automatically checkout a commit for you to test.
- Based on the result of the test, mark the commit as "good" or "bad" using the respective commands.
- Once Git Bisect identifies the problematic commit, you can end the bisect process:
- git bisect reset

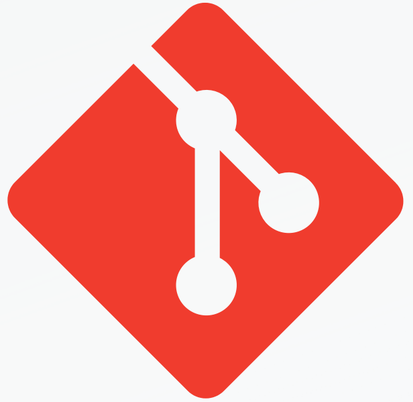


GIT REFLOG

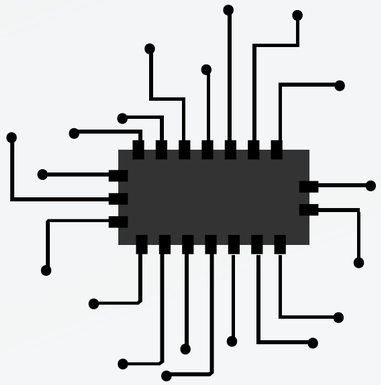


- Git Reflog, short for "reference log," records all the changes to your repository's branches.
- It is especially useful when you accidentally delete a branch or lose commits.
- By running the command "git reflog", you can view a detailed history of your branch changes, including commits, branch creations, deletions, and checkouts.
- You can use this information to recover lost commits or undo unintentional changes.

- To view the Git Reflog, you can run : git reflog
- If you accidentally delete a branch or lose commits, you can use the Git Reflog to recover them.
- Identify the commit hash from the Git Reflog corresponding to the lost work.
- Use the commit hash to create a new branch or cherry-pick the commit onto an existing branch.
- To clean up and remove unnecessary entries from the Git Reflog, you can run the = git reflog expire --expire-unreachable=now --all



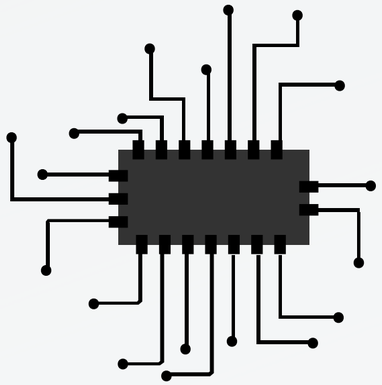
GIT DIFF



- Git Diff is a powerful tool that allows you to compare changes between different versions of files, commits, or branches in your Git repository.
- It provides a comprehensive and detailed view of the differences, highlighting additions, deletions, and modifications.
- Git Diff can be used in various scenarios, such as reviewing changes before committing, comparing branches, or examining modifications made in specific commits.

- To compare changes between two files: git diff <file1> <file2>
- To compare changes between two commits : git diff <commit1> <commit2>
- To compare changes between two branches : git diff <branch1> <branch2>
- Git Diff provides various options to customize the output and view specific types of changes.
- Some common options include "--stat" to show summary statistics, "--color-words" to highlight individual word changes, and "--cached" to compare changes in the staging area.

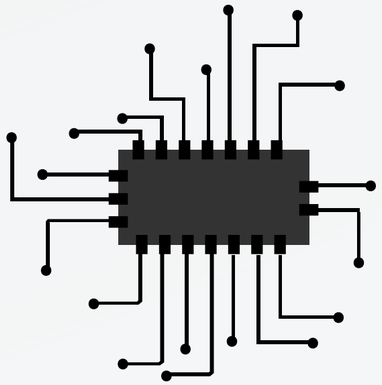
GIT SWITCH



- Git Switch is a relatively new command introduced in Git 2.23 as an alternative to the combination of **git checkout** and **git branch** commands.
- It simplifies the process of switching branches by providing a single command that handles both branch creation and switching.
- Git Switch offers a more intuitive and straightforward syntax for branch switching.
- It helps prevent accidental creation of new branches when intending to switch between existing branches.

- To switch to an existing branch, use the following command:
- git switch <branch_name>
- Git will check out the specified branch, making it the current branch.
- To create a new branch and switch to it in one step, use the following command:
- git switch -c <new_branch_name>
- You can use the **--discard-changes** flag to discard any modifications that have not been committed.

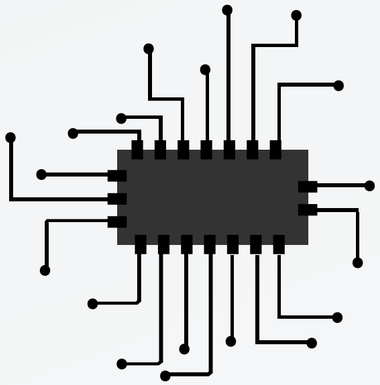
GIT REBASE



- Git Rebase is a command that allows you to incorporate changes from one branch onto another by moving, combining, or modifying commits.
- It provides a flexible way to rewrite the commit history and create a cleaner, more linear timeline.
- Git Rebase offers several benefits, such as:
- Simplifying the commit history by removing unnecessary or intermediate commits.
- Keeping the branch up-to-date with the latest changes from the main branch.

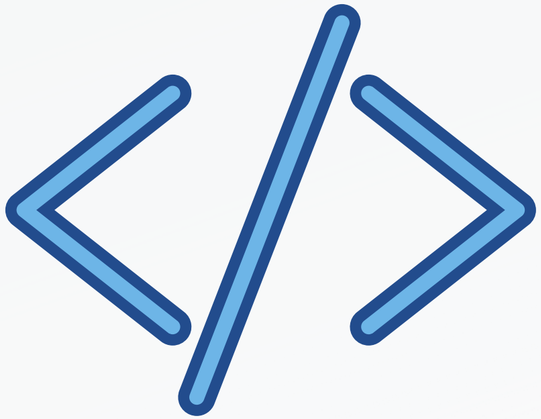
- To perform a basic rebase, use this command : git rebase <branch_name>
- Git Rebase also supports an interactive mode that allows more control over the commit history.
- Use the following command to enter interactive mode: git rebase -i <commit>
- Some common operations available during interactive rebase:
- Pick: Keeps the commit as is.
- Edit: Pauses the rebase process to allow amending or modifying the commit.
- Squash: Combines the commit with the previous commit.

GIT CHERRY-PICK



- Git Cherry-pick is a command that allows you to select individual commits from one branch and apply them onto another branch.
- It enables you to bring changes made in specific commits without merging or rebasing the entire branch.
- Git Cherry-pick offers several benefits, such as:
- Selectively incorporating changes from one branch into another.
- Applying bug fixes or specific features to other branches without merging the entire branch.

- To cherry-pick a single commit, use this : git cherry-pick <commit>
- Git Cherry-pick also allows you to apply multiple commits in one command.
- Specify the commit range using this : git cherry-pick <start_commit>..<end_commit>
- Mention that conflicts may arise when cherry-picking commits that modify the same lines of code as the target branch.
- After resolving the conflicts, use **git cherry-pick --continue** to continue the cherry-pick process.
- If you no longer need a cherry-picked commit and want to remove it from the branch,
- git cherry-pick --abort



THANK'S FOR LISTENING