Git Interview Questions and Answers

Sanjay Kumar PhD Following 5, 2025

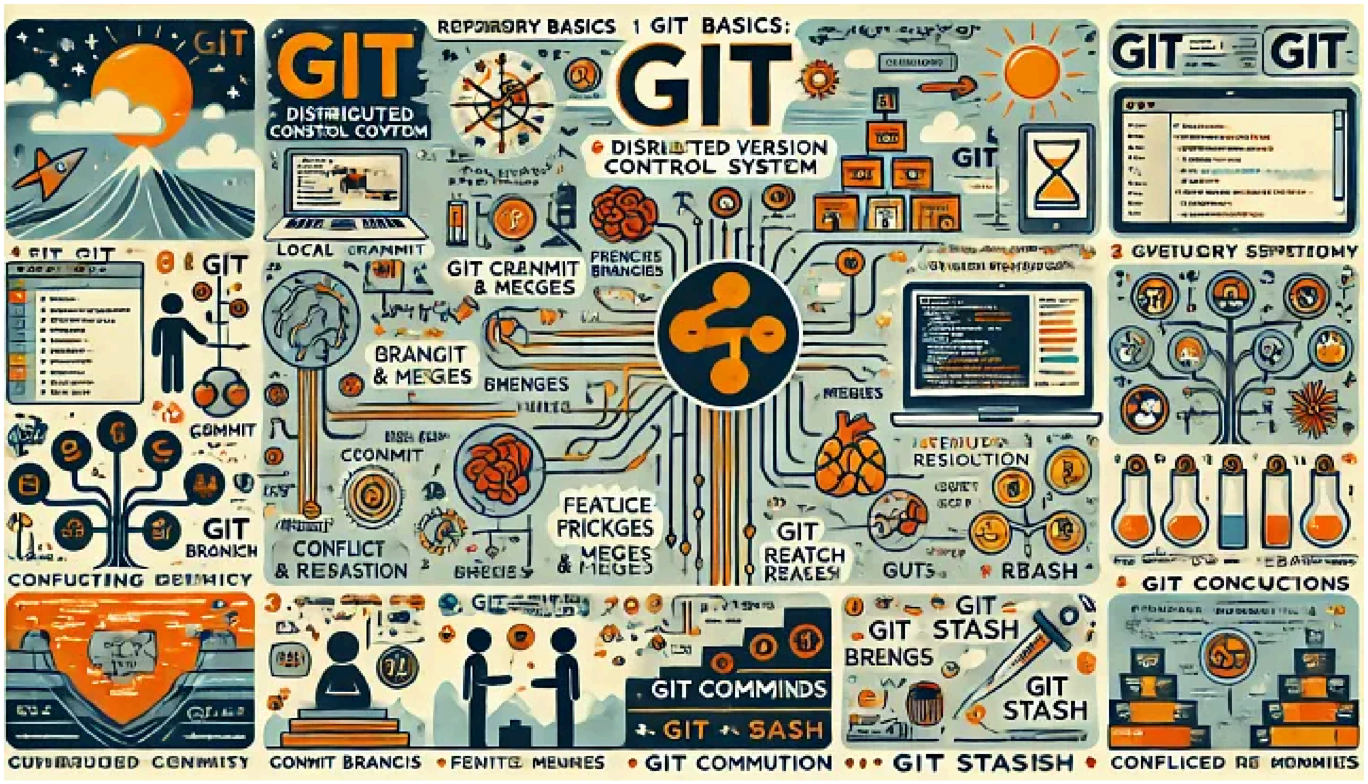


Image generated by Author using DALL E

1. What is Git?

Answer:
Git is a free and open-source **distributed version control system (VCS)** designed to handle everything from small to very large projects with speed and efficiency. It helps track changes in source code, enabling multiple developers to collaborate on the same project without conflicts. Git allows branching, merging, and maintaining different versions of code while ensuring data integrity.

2. Which programming language is Git written in?

Answer:
Git is written primarily in the **C programming language**, which ensures **high performance and efficiency** by minimizing the runtime overhead that higher-level languages introduce. This makes Git fast and lightweight, capable of handling large repositories with ease.

3. What is a repository in Git?

Answer:
A **repository (repo)** in Git is a **storage location** for a project's files and its entire version history. It includes:

- The **working directory** (where actual files reside)
- A hidden `.git` directory (which contains commit history, branches, tags, and configuration settings)

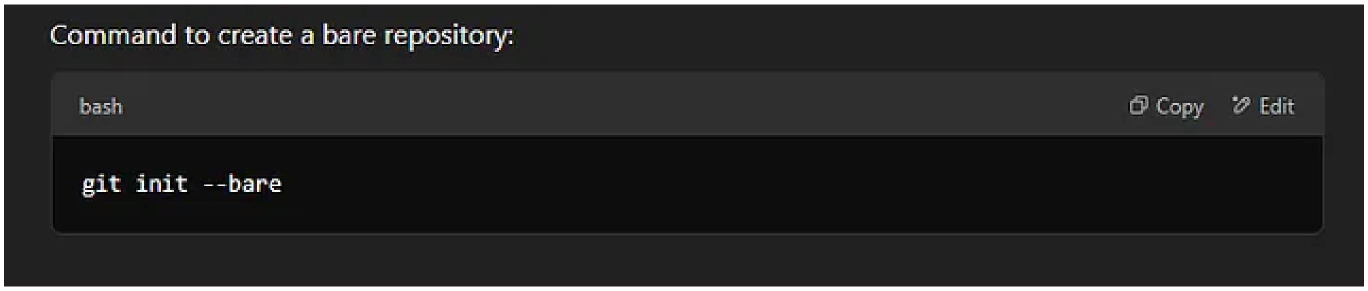
- A **staging area** (for tracking changes before committing them)

Repositories can be **local** (on a developer's machine) or **remote** (hosted on platforms like GitHub, GitLab, or Bitbucket).

4. What is a bare repository in Git?

Answer:
A **bare repository** is a Git repository that **does not have a working directory**. Instead, it contains only **Git metadata and version control information**. It is primarily used as a **central repository** for remote collaboration, where developers push and pull changes without directly modifying files.

Command to create a bare repository:

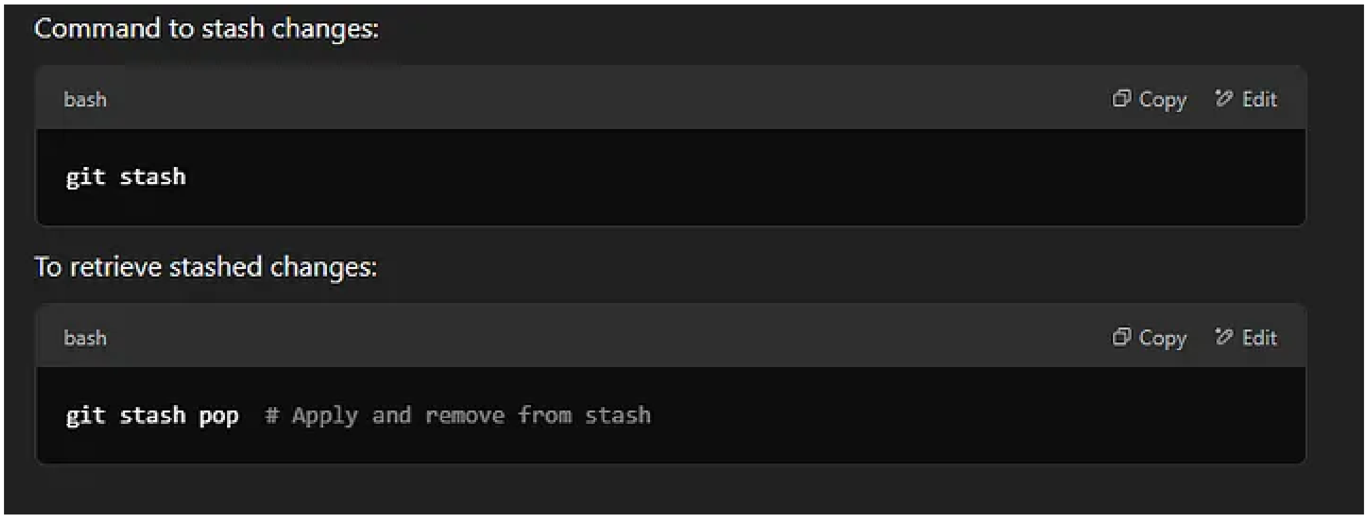


Command to create a bare repository:

```bash
git init --bare
```

5. What is the purpose of `git stash`?

Answer:
`git stash` temporarily saves **uncommitted** changes in a stack-like storage without committing them. This is useful when a developer needs to switch branches or work on another task but doesn't want to commit incomplete work.


Command to stash changes:

```bash
git stash
```

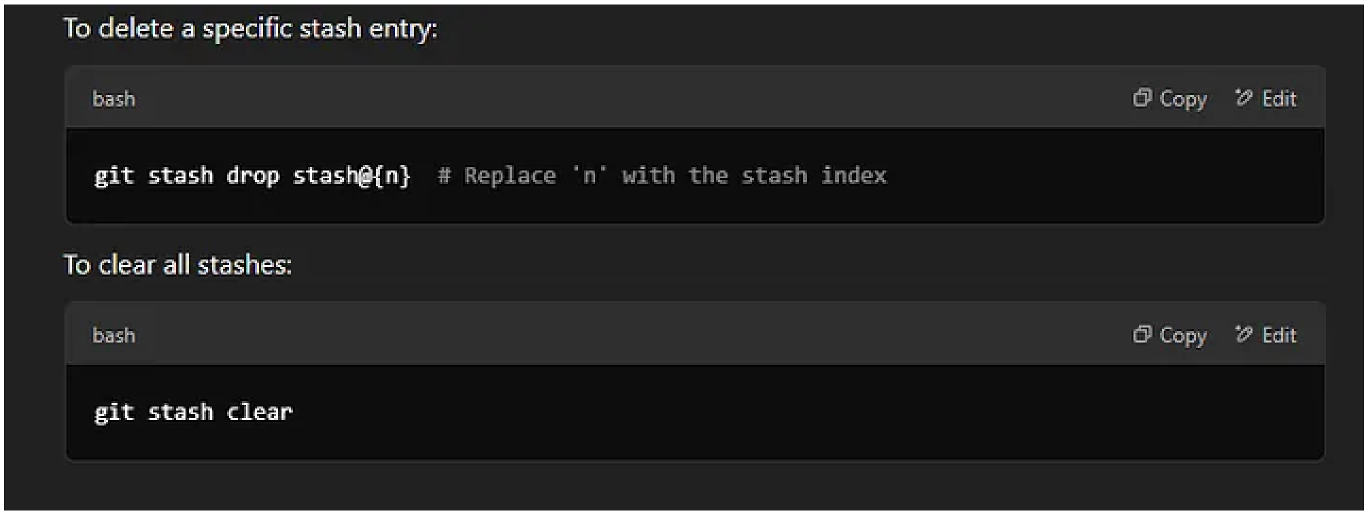
To retrieve stashed changes:

```bash
git stash pop  # Apply and remove from stash
```

6. What does `git stash drop` do?

Answer:
The command `git stash drop` removes the most recent stash entry from the list of stashed changes.

To delete a specific stash entry:


To delete a specific stash entry:

```bash
git stash drop stash@{n}  # Replace 'n' with the stash index
```

To clear all stashes:

```bash
git stash clear
```

7. What are the advantages of using Git?

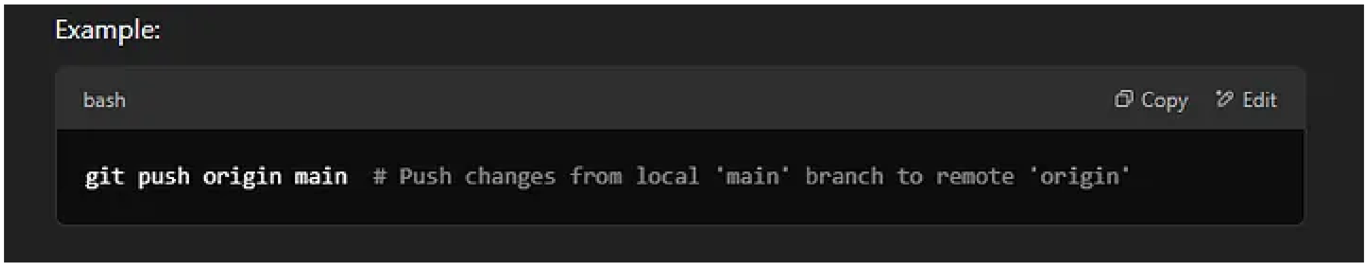
Answer:

- **Distributed version control** — Every developer has a full copy of the repository.
- **Fast and efficient** — Git is optimized for performance, handling large projects efficiently.
- **Supports local and remote repositories** — Developers can commit changes locally before pushing them remotely.
- **Powerful branching and merging** — Git allows seamless feature development and integration.
- **Optimized for collaboration** — Multiple developers can work simultaneously without overriding each other's work.
- **Strong security and data integrity** — Uses SHA-1 hashing to ensure data safety.

8. What does the `git push` command do?

Answer:
`git push` uploads committed changes from a **local repository** to a **remote repository**, making them available to other collaborators.

Example:


Example:

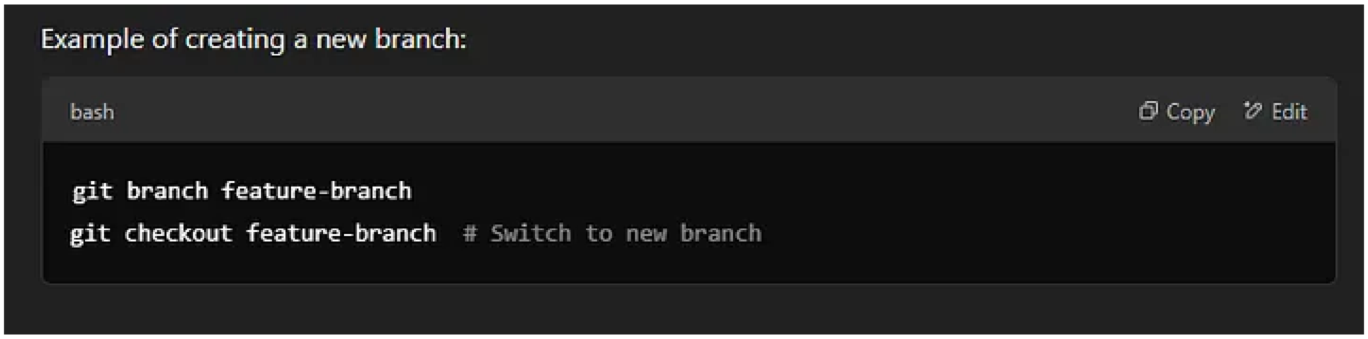
```bash
git push origin main  # Push changes from local 'main' branch to remote 'origin'
```

9. Why is branching important in Git?

Answer:
Branching allows developers to work on **different features or bug fixes independently**, without affecting the main codebase. It supports:

- **Parallel development** — Different teams can work on separate branches.
- **Feature isolation** — New features are developed in separate branches before merging into the main branch.
- **Rollback capabilities** — Developers can revert to previous versions if needed.

Example of creating a new branch:

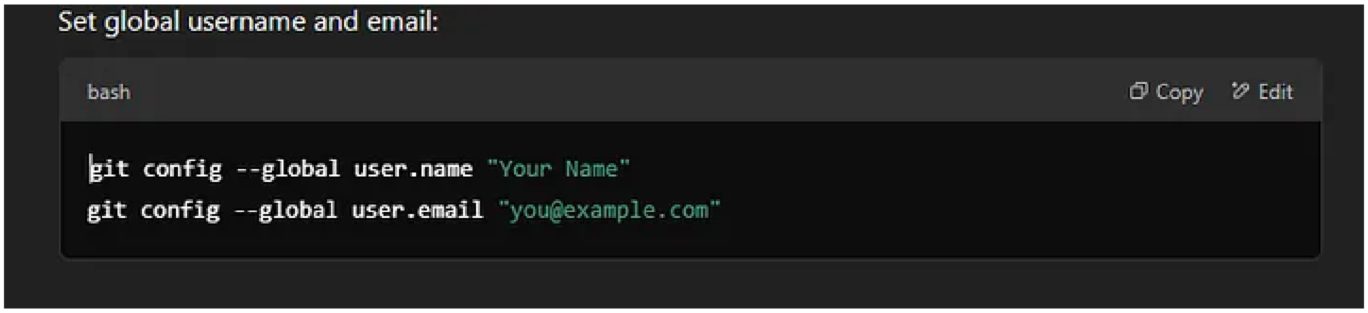

Example of creating a new branch:

```bash
git branch feature-branch
git checkout feature-branch  # Switch to new branch
```

10. What is the purpose of `git config`?

Answer:
`git config` is used to configure **user-specific Git settings** such as name, email, and default behaviors.

Set global username and email:


Set global username and email:

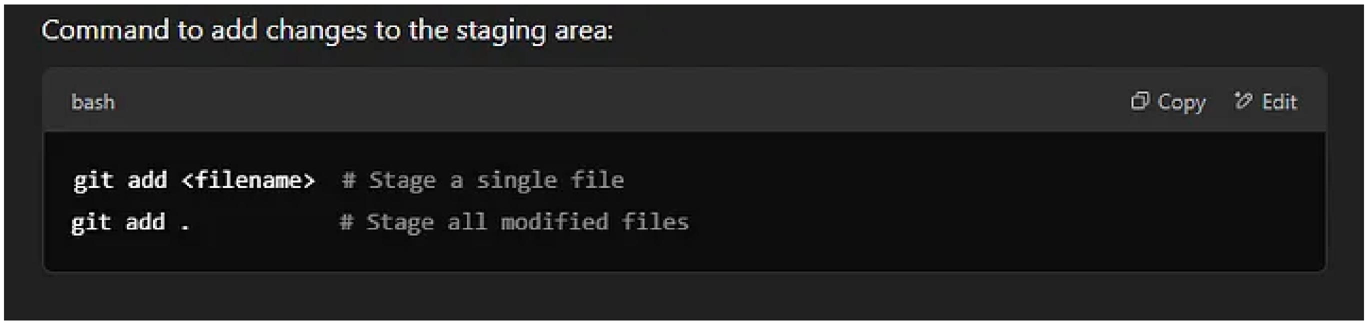
bash Copy Edit

```bash
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

11. What is the staging area (Index) in Git?

Answer:
The **staging area (Index)** is an **intermediate space** where changes are reviewed and formatted before committing them. It allows developers to selectively commit changes.

Command to add changes to the staging area:


Command to add changes to the staging area:

bash Copy Edit

```bash
git add <filename>  # Stage a single file
git add .           # Stage all modified files
```

12. What is a conflict in Git?

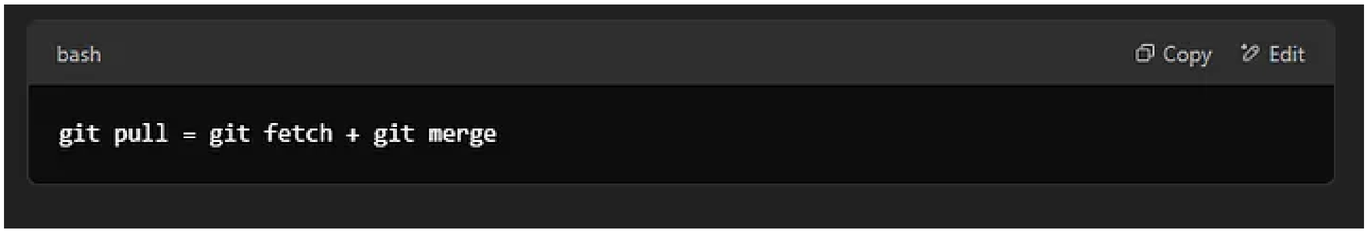
Answer:
A **Git conflict** occurs when two branches modify the same section of a file, and Git cannot automatically merge the changes. Developers must manually resolve conflicts before merging.

13. What is the difference between `git pull` and `git fetch`?

Answer:

- **`git fetch`** downloads changes from a remote repository but does **not** merge them.
- **`git pull`** downloads changes **and** automatically merges them into the current branch.

Formula:


bash Copy Edit

```bash
git pull = git fetch + git merge
```

14. How do you resolve a conflict in Git?

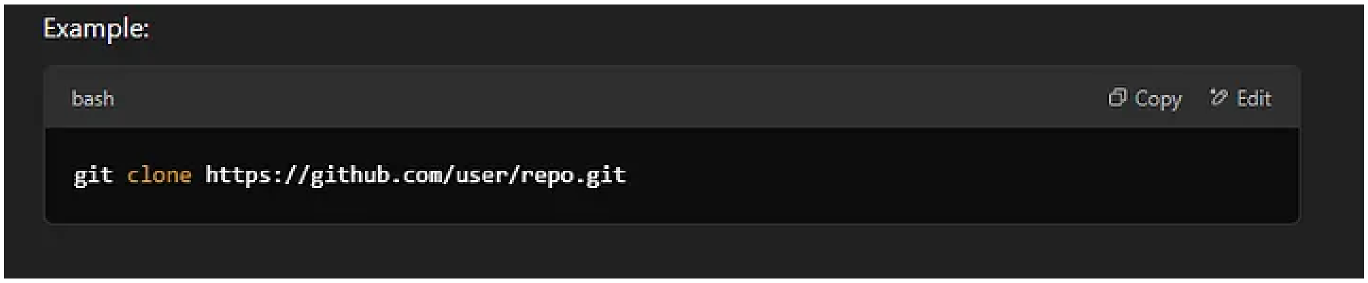
Answer:

1. Open the conflicting file and manually edit the conflicting sections.
2. Use `git add <file>` to mark the file as resolved.
3. Run `git commit` to finalize the merge.

15. What does `git clone` do?

Answer:
`git clone` creates a **copy** of a remote repository, including its full version history.

Example:


Example:

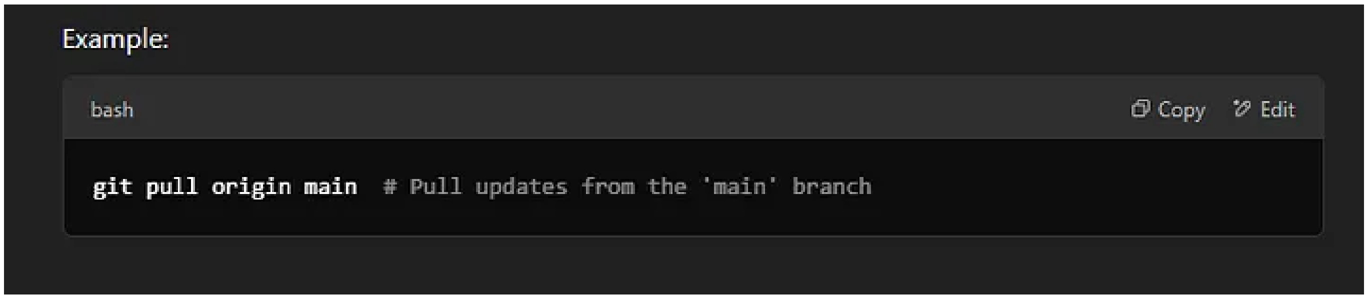
bash Copy Edit

```bash
git clone https://github.com/user/repo.git
```

16. What does `git pull origin <branch>` do?

Answer:
This command fetches and merges changes from the **remote repository** (**`origin`**) into the **current branch**.

Example:


Example:

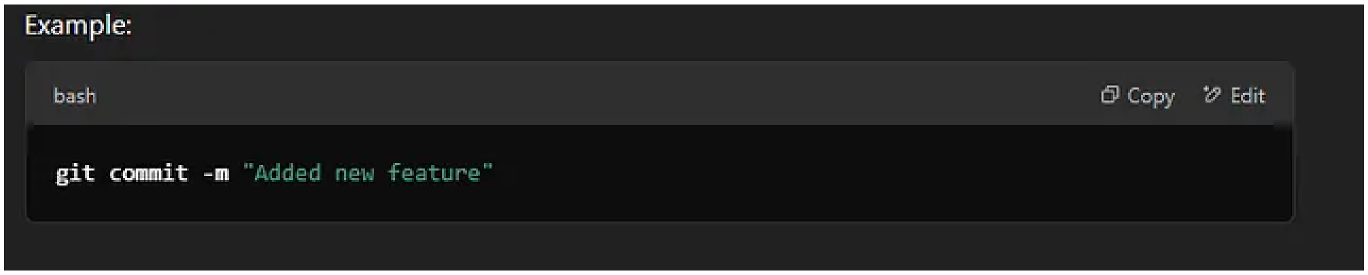
bash Copy Edit

```bash
git pull origin main  # Pull updates from the 'main' branch
```

17. What does `git commit` do?

Answer:
`git commit` saves changes from the **staging area** to the **local repository**.

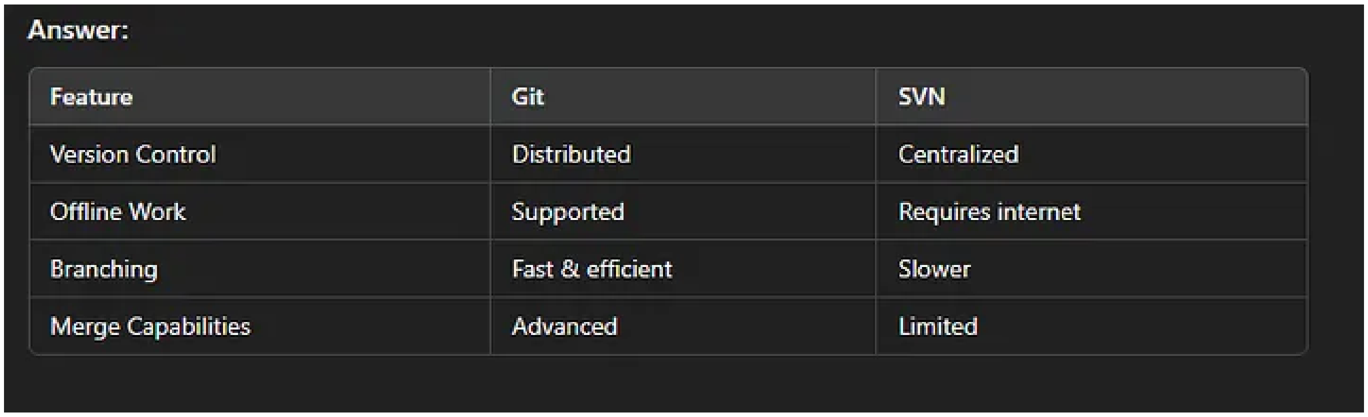
Example:



Example:

```bash
git commit -m "Added new feature"
```

18. How is Git better than Subversion (SVN)?



Answer:

Feature	Git	SVN
Version Control	Distributed	Centralized
Offline Work	Supported	Requires internet
Branching	Fast & efficient	Slower
Merge Capabilities	Advanced	Limited

19. What is a commit message?

Answer:

A **commit message** is a short description of the changes in a commit. It helps track modifications and understand project history.

20. Why should new commits be created instead of amending existing commits?

Answer:

- **Amending rewrites history**, making collaboration difficult.
- **New commits maintain clarity** in version control.

21. What are Git hooks?

Answer:
Git hooks are **scripts** that execute **before or after Git operations** (e.g., commits, merges). They help automate tasks like code formatting and security checks.

22. What does a commit object contain?

Answer:
A commit object consists of:

- A **snapshot** of project files at that moment.
- References to **parent commits**.
- A **unique SHA-1 hash identifier**.

23. What branching strategies have you used?

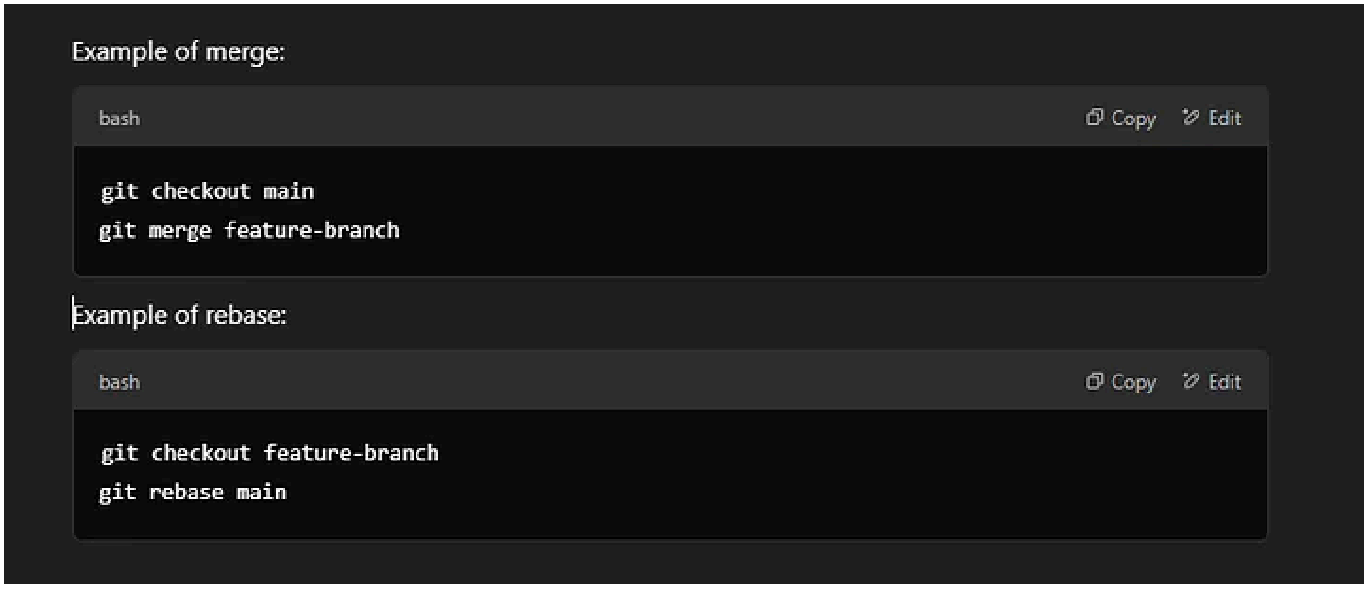
Answer:

- **Feature Branching** — Separate branches for new features.
- **Task Branching** — Each task has its own branch.
- **Release Branching** — A stable branch for releases.

24. What is the difference between `git merge` and `git rebase`?

Answer:
Both `git merge` and `git rebase` integrate changes from one branch into another, but they work differently:

- **`git merge`** creates a new merge commit, preserving the history of both branches.
- **`git rebase`** moves or **rebases** commits from one branch onto another, creating a linear history.



Example of merge:

```bash
git checkout main
git merge feature-branch
```

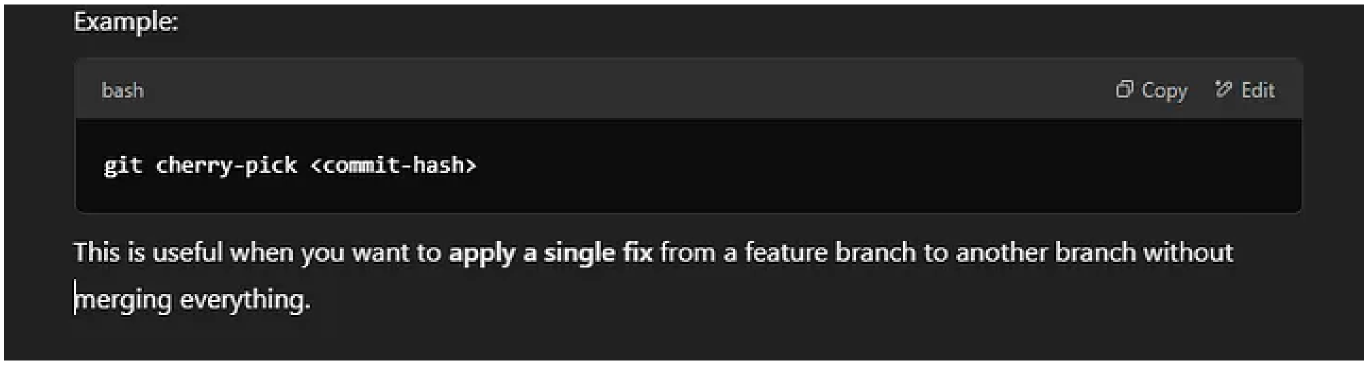
Example of rebase:

```bash
git checkout feature-branch
git rebase main
```

25. What is the `git cherry-pick` command?

Answer:
`git cherry-pick` allows selecting a specific commit from one branch and applying it to another.

Example:


Example:

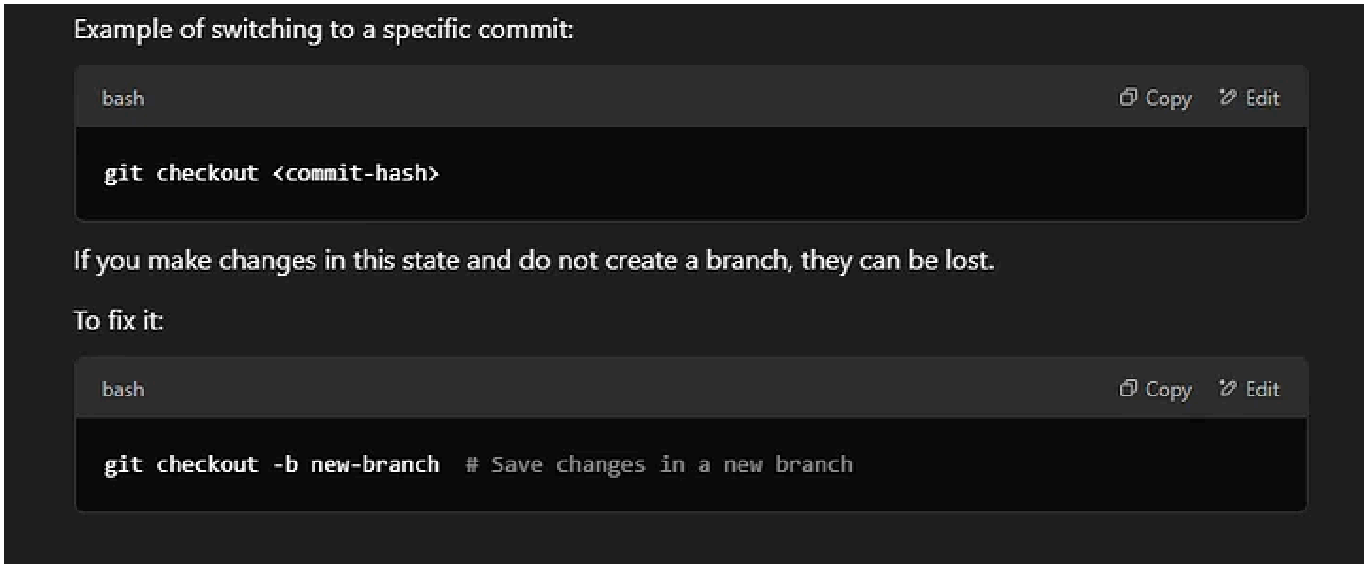
```bash
git cherry-pick <commit-hash>
```

This is useful when you want to **apply a single fix** from a feature branch to another branch without merging everything.

26. What is a detached HEAD in Git?

Answer:
A **detached HEAD** occurs when Git's HEAD is pointing to a specific commit instead of a branch.

Example of switching to a specific commit:


Example of switching to a specific commit:

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

If you make changes in this state and do not create a branch, they can be lost.

To fix it:

```bash
git checkout -b new-branch  # Save changes in a new branch
```

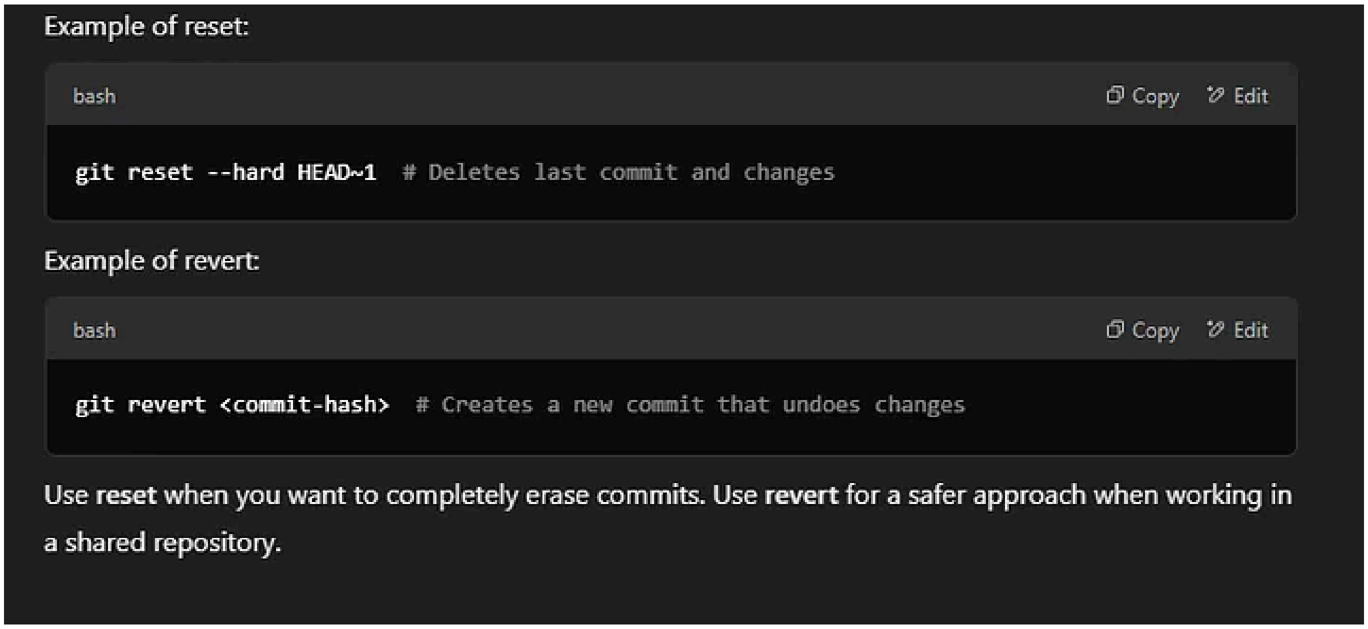
27. What is the difference between `git reset` and `git revert`?

Answer:

- **`git reset`** moves the branch pointer **backward** in history, effectively deleting commits.

- `git revert` creates a **new commit** that undoes a previous commit while keeping the history intact.

Example of reset:



Example of reset:

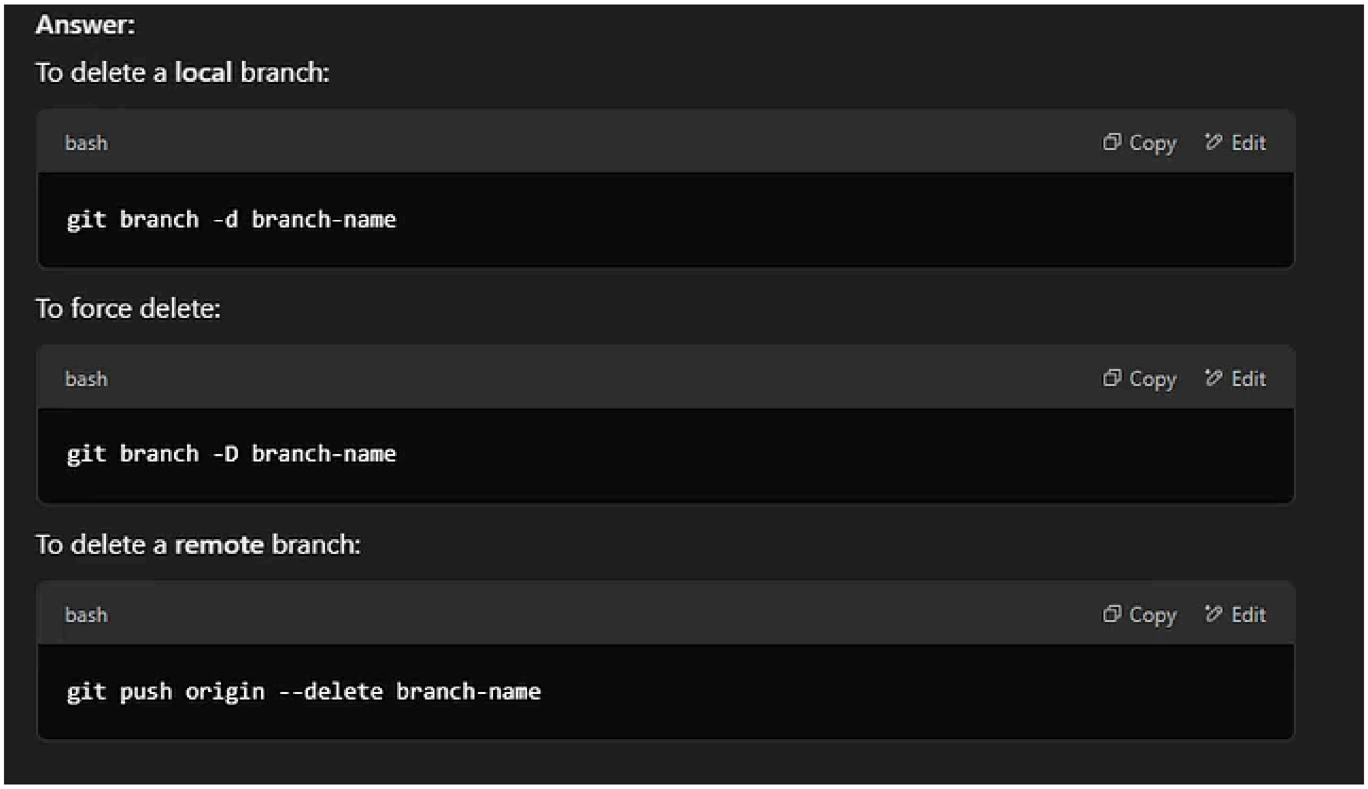
```bash
git reset --hard HEAD~1  # Deletes last commit and changes
```

Example of revert:

```bash
git revert <commit-hash>  # Creates a new commit that undoes changes
```

Use **reset** when you want to completely erase commits. Use **revert** for a safer approach when working in a shared repository.

28. How do you delete a Git branch?



Answer:

To delete a **local** branch:

bash Copy Edit

```bash
git branch -d branch-name
```

To force delete:

bash Copy Edit

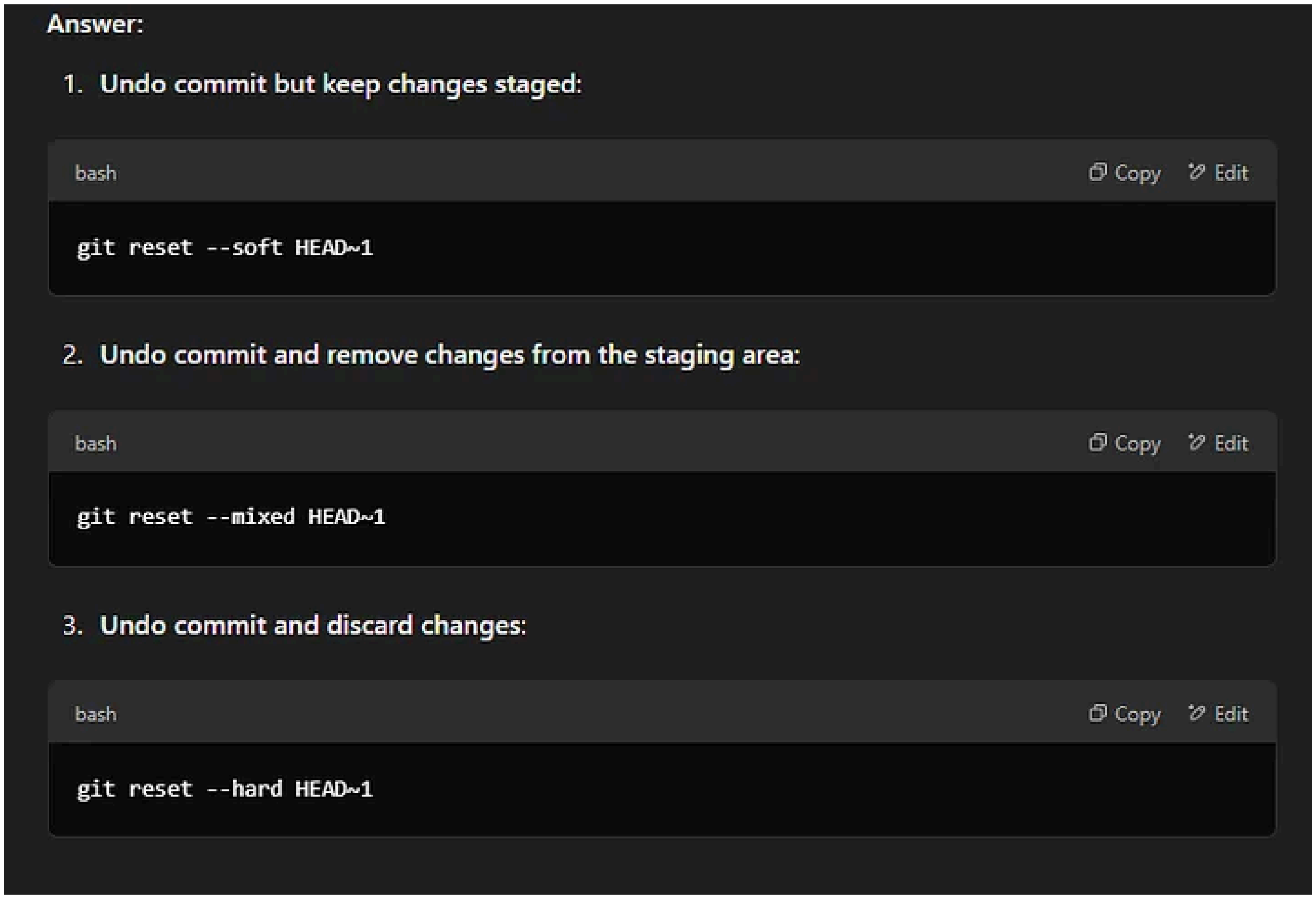
```bash
git branch -D branch-name
```

To delete a **remote** branch:

bash Copy Edit

```bash
git push origin --delete branch-name
```

29. How do you undo the last commit in Git?



Answer:

1. **Undo commit but keep changes staged:**

bash Copy Edit

```bash
git reset --soft HEAD~1
```

2. **Undo commit and remove changes from the staging area:**

bash Copy Edit

```bash
git reset --mixed HEAD~1
```

3. **Undo commit and discard changes:**

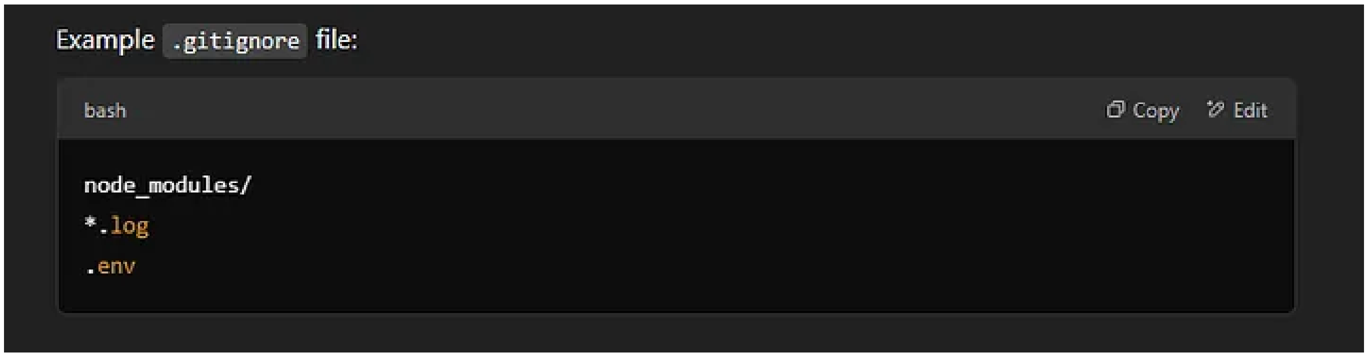
bash Copy Edit

```bash
git reset --hard HEAD~1
```

30. What does the `.gitignore` file do?

Answer:
The `.gitignore` file specifies **which files and directories should be ignored** by Git, preventing them from being tracked.

Example `.gitignore` file:


Example `.gitignore` file:

bash Copy Edit

```bash
node_modules/
*.log
.env
```

Machine Learning Data Science Git Github AI

Written by Sanjay Kumar PhD

965 followers · 443 following

Following

Data Science |Machine Learning | AI Product | GenAI | RAG | LLM | AI Agents | NLP | Analytics | Data Engineering | Deep Learning | Statistics

No responses yet

Hlgsagar

What are your thoughts?

More from Sanjay Kumar PhD

Sanjay Kumar PhD

Microsoft Azure Interview Questions and Answers

Core Azure Concepts & Cloud Computing Basics

Feb 27 70

In Artificial Intelligence in Plain... by Sanjay Kumar...

Top 100 AI Agent Interview Questions

What is an AI agent, and how does it work?

Jun 1 67 2

Sanjay Kumar PhD

Data Engineering Interview Questions and Answers

1. What is a Data Warehouse, and how is it different from a Data Lake?

Dec 26, 2024 61 2

Sanjay Kumar PhD

Advanced SQL Interview Questions and Answers

1. What is the difference between RANK(), DENSE_RANK(), and ROW_NUMBER()?

Mar 26 60

See all from Sanjay Kumar PhD

Recommended from Medium

Zudonu Osomudeya

50 Scenario-Based Interview Questions and Answers

Apr 4 19 2

SQL Mentor

Top 15 SQL Queries to Find Duplicates

A practical guide to identifying duplicates across tables, columns, and conditions

2d ago 4

Code With Hannan

The 7 SQL Patterns Every Analyst Uses (But Never Talks About)

SQL is the language of data analysts and engineers alike. While many focus on writing...

6d ago 41 1

In CodeX by Tripathi Aditya Prakash

SQL Joins Explained the Simplest Way

SQL joins are a critical part of working with relational databases. They allow you to...

Dec 23, 2024 4

Kavya's Programming Path

If I'm Taking Your React Interview, I'm Asking You These 30 Questio...

If you're preparing for a React job interview and just memorizing a few definitions, let me...

2d ago 3 1

Sanjay Kumar PhD

Top 100 Azure Data Engineering Interview Questions and Answers

1. How would you design a scalable ETL pipeline using Azure Data Factory for...

Jun 9

See more recommendations