Top 20 Git Interview Questions and Answers for All Levels

Learn how to ace your technical interview.

Contents May 28, 2024 · 13 min read

Kurtis Pykes
Data Science & AI Blogger | Top 1000 Medium Writers on AI and Data Science

TOPICS

Git

Career Services

Git is an essential tool in the modern developer's toolkit, renowned for its powerful version control capabilities. Created by Linus Torvalds in 2005 to support the development of the Linux kernel, Git has since become the backbone of countless software projects worldwide. Its efficiency and flexibility in managing project versions, coupled with robust support for collaboration, make it indispensable for teams of all sizes.

This article aims to prepare you for technical interviews by covering the top 20 Git interview questions that span from beginner to advanced levels. Whether you're new to Git or looking to deepen your understanding, these questions and answers will help you demonstrate your proficiency and ace your interview.

Become a Data Engineer
Build Python skills to become a professional data engineer.

Get Started for Free

Basic Git Interview Questions

If you're a relative newcomer to Git, it's likely that some of the basic interview questions will deal with beginner concepts and uses. If you need to brush up on these, be sure to check out DataCamp's Introduction to Git course.

1. What is a Git repository?

A Git repository stores a project's files and revision history and facilitates version control by tracking changes made over time. It can be located locally within a folder on your device or an online platform like GitHub. This enables users to collaborate, revert to previous versions, and efficiently manage project development using commands like commit, push, and pull.

2. How does Git work?

Git operates by recording changes made to files and directories in a project, capturing snapshots of its evolving condition. Users can oversee alterations, create branches for simultaneous development, merge branches, and revert to previous states if required. It also promotes collaboration and ensures effective version control in software development endeavors.

3. What is `git add`?

The `git add` command is used in Git to stage changes for inclusion in the next commit. It prepares modifications, additions, or deletions made to files in the working directory, marking them to be included in the upcoming commit snapshot. Note this command does not actually commit the changes but prepares them for staging.

4. What is `git push`?

The `git push` command is used in Git to upload local repository content to a remote repository. It transfers committed changes from the local repository to a remote one, typically on a server like GitHub or GitLab. This command enables collaboration by allowing users to share their changes with others on the same project.

You can learn more about **Git push and pull** in our separate tutorial.

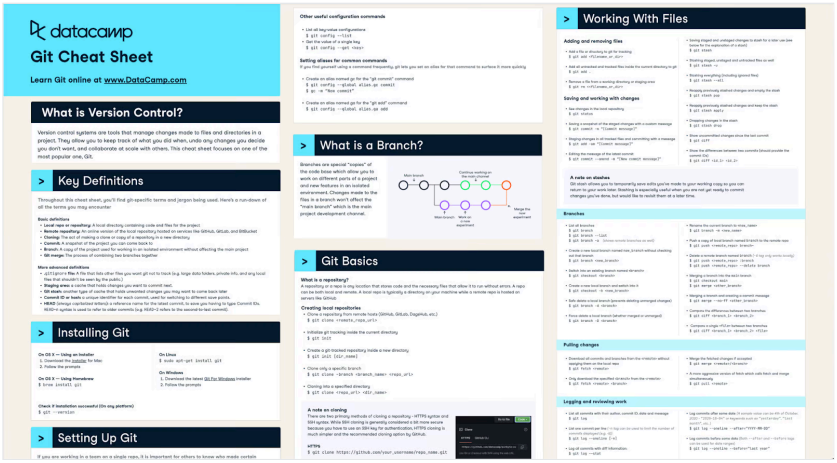
5. What is `git status`?

The `git status` command displays the current state of the repository in Git. It provides information about which files have been modified, which are staged for the next commit, and which are untracked. It helps users track the progress of their work and identify any changes that need to be committed or staged.

6. What is a commit in Git?

A commit represents a snapshot of the changes made to files in a repository at a specific point in time. When you commit changes in Git, you are effectively saving the current state of your files and can provide a descriptive message that explains the changes made (which is recommended).

Each commit creates a unique identifier, allowing you to track the history of changes in the repository. Commits play a crucial role in version control, as they provide a way to revert to previous states of the project, review the history of changes, and collaborate with others by sharing updates.



*Check out DataCamp's **Git Cheat Sheet** to help with your interview prep*

7. What is branching in Git?

Branching refers to the practice of diverging from the main line of development (typically called the "master" branch) to work on new features, fixes, or experiments without affecting the main codebase. It allows multiple parallel lines of development to coexist within the same repository.

Each branch represents a separate line of development with its own set of commits, enabling developers to work on different features or fixes simultaneously. Branching facilitates collaboration, experimentation, and organization within a project, as changes made in one branch can be merged back into the main codebase once they are completed and tested.

8. What is a conflict in Git?

Conflicts arise when conflicting changes are made to the same part of a file or files by different contributors, typically during a merge or rebase operation. Git cannot automatically resolve these conflicting changes, requiring manual intervention by the user to resolve the discrepancies.

Thus, to resolve conflicts, the conflicted files must be reviewed and edited based on the best suited reconciliation before the resolved version is committed.

9. What is merge in Git?

Merging is a fundamental operation in Git that facilitates collaboration and the integration of changes across different branches in a project. Namely, a merge is the process of combining the changes from different branches into a single branch, typically the main branch (e.g., master or main).

A merge integrates the changes made in one branch with another, resulting in a new commit that combines the histories of both branches. You can learn more about **how to resolve merge conflicts in Git** with our separate tutorial.

Get certified in your dream Data Engineer role

Our certification programs help you stand out and prove your skills are job-ready to potential employers.

Get your Certification

Intermediate Git Interview Questions

10. What is a remote in Git?

A remote is a repository hosted on a server or another computer for collaboration and sharing code with others. It serves as a centralized location where developers can push their local changes and pull changes made by others.

Remotes are typically set up on hosting platforms like GitHub, GitLab, or Bitbucket, and they enable distributed development and facilitate teamwork by providing a common location for storing and synchronizing project code among multiple contributors.

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

The main difference between git fetch and git pull lies in what they do and how they update the local repository.

The `git fetch` command retrieves changes from a remote repository to the local repository. It updates the remote-tracking branches (e.g., origin/master) in the local repository to reflect the state of the remote repository, but it does not update the working directory or merge any changes into the current branch. This means that after fetching, you can review the changes made in the remote repository without affecting your local work.

The `git pull` command also retrieves changes from a remote repository, but it goes a step further by fetching changes and merging them into the current branch in one step. It essentially performs a git fetch followed by a git merge to incorporate the changes from the remote repository into the current branch.

12. How do you revert a commit that has already been pushed and made public?

The `git revert <commit-hash>` command can be used to revert a commit that has already been pushed and made public.

The step-by-step process is as follows:

1. Identify the commit you want to revert to by finding its commit hash. This can be done using the `git log` command to view the commit history and find the commit hash you want to revert.

2. Once you have the commit hash, use the git revert command followed by the commit hash to create a new commit that undoes the changes introduced by the specified commit. For example:

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

POWERED BY datalab

3. Git will open a text editor to create a commit message for the revert. You can edit the message if needed, then save and close the editor.

4. After saving the commit message, Git will create a new commit that effectively undoes the changes introduced by the specified commit. This new commit will be added to the history, effectively reverting the changes made by the original commit.

5. Finally, push the new commit to the remote repository to make the revert public using the following command:

```
git push origin <branch-name>
```

POWERED BY datalab

Using `git revert` creates a new commit that undoes the changes introduced by the original commit, effectively reverting the changes without altering the commit history. This approach is safer than `git reset` or `git amend`, which can alter the commit history and cause issues for collaborators who have already pulled the changes.

13. What does `git reset` do?

The `git reset` command resets the current HEAD to a specified state. This means it can be used to undo changes, unstage files, or move the HEAD pointer to a different commit. Note there are three main modes of `git reset`:

- `--soft`: Resets the HEAD pointer to a specific commit, keeping changes staged. Files remain modified in the working directory, allowing you to re-commit them.
- `--mixed`: Resets the HEAD pointer to a specific commit, unstaging changes. Files remain modified in the working directory, but changes are not staged for commit.
- `--hard`: Resets the HEAD pointer to a specific commit, discarding all changes in the working directory and staging area. Use with caution, as it permanently deletes uncommitted changes.

14. What is `git stash`?

`git stash` is a Git command that temporarily stores changes in the working directory that are not ready to be committed. It allows developers to save their modifications without committing them to the repository.

Stashing is useful when switching branches, but you don't want to commit or lose your changes. Later, you can apply the stashed changes to your working directory or pop them

off the stash stack to continue working on them.

15. What is `git reflog`?

`git reflog` is a Git command used to view the reference logs, which record changes to the HEAD pointer and the history of commits that have been checked out in the repository. It provides a chronological list of recent actions performed in the repository, including commits, checkouts, merges, and resets.

The reflog is helpful for recovering lost commits or branches and understanding the sequence of actions taken in the repository.

16. How do you make an existing Git branch track a remote branch?

To make an existing Git branch track a remote branch, you can use the `git branch` command with the `--set-upstream-to` or `-u` option, followed by the name of the remote branch.

The syntax will look as follows:

```
git branch --set-upstream-to=<remote-name>/<branch-name>
```

POWERED BY datalab

Or...

```
git branch -u <remote-name>/<branch-name>
```

POWERED BY datalab

Advanced Git Interview Questions

17. How do you manage multiple configurations for different projects in Git?

To handle various configurations, utilize the `git config` command alongside the `--global`, `--system`, or `--local` flags to adjust configuration settings at distinct levels. Alternatively, employ `includeIf` in the Git configuration to incorporate specific setups based on the repository's path.

18. How do you handle large files with Git?

Handling large files in Git can be challenging due to their impact on repository size and performance. Use Git LFS to store large files outside the Git repository while keeping lightweight pointers to them in the repository. This reduces the size of the repository and improves performance. Git LFS supports various storage providers and integrates seamlessly with Git workflows.

19. What is the use of `git submodule` and how do you update one?

The `git submodule` command manages external dependencies within a Git repository. It allows you to include external repositories as submodules within your main repository. This is useful when you want to incorporate code from external sources while keeping it separate from your main project's codebase.

To update a submodule in Git, you can use the following steps:

1. Navigate to the directory of the submodule within your main repository.
2. Use `git fetch` to fetch the latest changes from the submodule's remote repository.
3. If you want to update to the latest commit on the branch tracked by the submodule, you can use `git pull`.
4. Alternatively, if you want to update to a specific commit or branch, you can use `git checkout` followed by the desired commit hash or branch name.
5. Once you have updated the submodule to the desired state, you need to commit the changes to the main repository to reflect the updated submodule state.

20. What is the significance of `git push --force-with-lease` over `git push --force`?

The git push --force-with-lease is a more cautious approach to force-pushing changes to a remote repository than git push --force because it prevents accidentally overwriting changes made by others on the remote repository.

When you use git push --force , you force-push your changes to the remote repository regardless of whether others have updated it since your last fetch. This can lead to the unintentional loss of other developers' work.

In contrast, git push --force-with-lease is a safer alternative. It checks whether the remote branch you are pushing to has been updated by others since your last fetch. If the remote branch has been updated, the push is rejected, preventing you from overwriting other developers' changes unintentionally.

Preparing for a Technical Interview

Presenting your Git knowledge and experience during interviews is crucial for showcasing your proficiency in version control and collaboration within software development teams.

Let's look at some tips you should follow when preparing for your technical interview to communicate your Git skills effectively:

Understand Git fundamentals

Ensure you have a solid understanding of Git fundamentals, including repositories, branching, merging, commits, and basic commands like pull, push, clone, and commit. This foundational knowledge will form the basis of your discussion during the interview. It also helps if you thoroughly understand essential principles like version control, discerning the disparities between Git and alternative version control systems (VCS), and comprehending their significance in software development.

Lastly, Familiarize yourself with diverse Git methodologies, such as Git Flow, GitHub Flow, and GitLab Flow. Assess the advantages and drawbacks of each approach and discern the situations in which they are most beneficial.

DataCamp's **complete guide to Git** is a good starting point for familiarizing yourself with the fundamentals.

Get hands-on experience

The more you use Git, the more you reinforce your knowledge. Regular practice enhances your familiarity with various commands and procedures. Seek to incorporate Git into your daily workflow to gain more exposure. Be sure to experiment with creating branches, merging them, and resolving conflicts.

If you're unsure of what projects to work on to gain hands-on experience with Git, participating in open-source projects via platforms like GitHub is a great way to get firsthand exposure to industry-standard collaboration tools and workflows.

Learn common issues and how to troubleshoot them

You are bound to encounter problems when using Git. Some common issues include merge conflicts, detached HEAD states, reverting changes, and recovering lost commits. Diagnosing Git issues enhances troubleshooting skills and fosters a deeper understanding of Git's underlying mechanisms.

By actively troubleshooting and analyzing error messages, you will gain insights into Git's internal workings and develop proficiency in identifying and resolving issues efficiently. This proactive approach mitigates potential risks and effectively builds confidence and expertise in managing version control workflows.

Practice mock interviews

By engaging in mock interviews, candidates can identify areas of weakness in their Git knowledge and communication skills, allowing them to focus their preparation efforts effectively.

Additionally, mock interviews offer valuable opportunities for candidates to refine their problem-solving abilities by tackling realistic Git-related scenarios and coding exercises. This hands-on practice helps candidates develop confidence in their Git skills and enhances their ability to articulate their thoughts clearly during the interview.

Conclusion

Git is a powerful version control system widely used in software development to manage code changes, collaborate with others, and maintain project history. Familiarity with Git is essential for technical interviews as it demonstrates proficiency in essential developer tools and workflows, showcases collaboration skills, and highlights the ability to manage code effectively in team environments.

Additionally, understanding Git concepts and commands enables efficient version control practices, ensuring code integrity, project continuity, and streamlined development processes. Thus, knowledge of Git is invaluable for aspiring software engineers and developers navigating technical interviews and pursuing successful careers

For further learning, check out the following resources:

- What is Git? – The Complete Guide to Git
- GitHub and Git Tutorial for Beginners
- GitHub Concepts
- Complete Git Cheat Sheet

AUTHOR
Kurtis Pykes

TOPICS

Git Career Services

Training more people?

Get your team access to the full DataCamp for business platform.

For Business

For a bespoke solution book a demo.

Continue Your Git Journey Today!

TRACK

Data Engineer in Python

40hrs hr

Gain in-demand skills to efficiently ingest, clean, manage data, and schedule and monitor pipelines, setting you apart in the data engineering field.

See Details →

Start Course

See More →

Related

BLOG

Top 20 GCP Interview Questions: A Guide for All Skill...

BLOG

56 Java Interview Questions And Answers For All Levels

BLOG

40 R Programming Interview Questions & Answers For All...

See More →

Grow your data skills with DataCamp for Mobile

Make progress on the go with our mobile courses and daily 5-minute coding challenges.

Download on the App Store

GET IT ON Google Play

Career Tracks

Skill Tracks

Courses

Data Science Roadmap

DATA COURSES

Python Courses

R Courses

SQL Courses

Power BI Courses

Tableau Courses

Alteryx Courses

Azure Courses

AWS Courses

Google Sheets Courses

Excel Courses

AI Courses

Data Analysis Courses

Data Visualization Courses

Machine Learning Courses

Data Engineering Courses

Probability & Statistics Courses

DATALAB

Get Started

Pricing

Security

Documentation

CERTIFICATION

Certifications

Data Scientist

Data Analyst

Data Engineer

SQL Associate

Power BI Data Analyst

Tableau Certified Data Analyst

Azure Fundamentals

AI Fundamentals

RESOURCES

Resource Center

Upcoming Events

Blog

Code-Alongs

Tutorials

Docs

Open Source

RDocumentation

Book a Demo with DataCamp for Business

Data Portfolio

PLANS

Pricing

For Students

For Business

For Universities

Discounts, Promos & Sales

Expense DataCamp

DataCamp Donates

FOR BUSINESS

Business Pricing

Teams Plan

Data & AI Unlimited Plan

Customer Stories

Partner Program

ABOUT

About Us

Learner Stories

Careers

Become an Instructor

Press

Leadership

Contact Us

DataCamp Español

DataCamp Português

DataCamp Deutsch

DataCamp Français

SUPPORT

Help Center

Become an Affiliate

Privacy Policy Cookie Notice Do Not Sell My Personal Information Accessibility Security Terms of Use

© 2025 DataCamp, Inc. All Rights Reserved.