Contents

[Basic Git Concepts 2](#_Toc195274054)

[1. What is Git, and how is it different from other VCS tools? 2](#_Toc195274055)

[2. What is the difference between Git and GitHub? 3](#_Toc195274056)

[3. What is the difference between git clone and git fork? 3](#_Toc195274057)

[4. Explain the Git lifecycle: Working Directory, Staging Area, and Repository. 3](#_Toc195274058)

[5. What is the purpose of .gitignore? 3](#_Toc195274059)

[**💥 Error Handling & Recovery** 3](#_Toc195274060)

[6. How do you resolve merge conflicts? 3](#_Toc195274061)

[7. What to do when you accidentally commit sensitive info? 4](#_Toc195274062)

[4](#_Toc195274063)

[8. How do you remove a file from Git history? 4](#_Toc195274064)

[9. Difference between revert, reset, and checkout? 4](#_Toc195274065)

[10. What is "detached HEAD" state, and how do you exit it? 5](#_Toc195274066)

[**🔐 GitHub-Specific (CI/CD, Collaboration)** 5](#_Toc195274067)

[11. How to Create a GitHub Repository 5](#_Toc195274068)

[12. How to Fork a Repository 6](#_Toc195274069)

[13. How to Enable Branch Protection Rules 6](#_Toc195274070)

[14. How GitHub Actions Work 7](#_Toc195274071)

[15. What are merge conflicts? How do you resolve them? 8](#_Toc195274072)

[16. What is origin? 8](#_Toc195274073)

[17. Difference Between git fetch and git pull? 8](#_Toc195274074)

[**🔁 Branching & Merging** 8](#_Toc195274075)

[18. How do you create a new branch? 8](#_Toc195274076)

[19. How do you switch branches? 8](#_Toc195274077)

[20. How do you create and switch to a new branch? 9](#_Toc195274078)

[21. How do you merge a branch into the main branch? 9](#_Toc195274079)

[**⬆️ Remote Repositories** 9](#_Toc195274080)

[22. How do you add a remote repository? 9](#_Toc195274081)

[23. How do you push changes to a remote repo? 9](#_Toc195274082)

[24. How do you pull changes from a remote repo? 9](#_Toc195274083)

[25. How do you rename a remote? 9](#_Toc195274084)

[**💻 Core Git Commands** 9](#_Toc195274085)

[26. How do you initialize a Git repository? 9](#_Toc195274086)

[27. How do you check the current status of your repo? 9](#_Toc195274087)

[28. How do you stage files for commit? 10](#_Toc195274088)

[29. How do you commit changes with a message? 10](#_Toc195274089)

[30. How do you view commit history? 10](#_Toc195274090)

[**🔍 Advanced Git Commands** 10](#_Toc195274091)

[31. How do you check the difference between staged and working directory? 10](#_Toc195274092)

[32. How do you check the difference between last commit and current state? 10](#_Toc195274093)

[33. How do you revert a file to a previous commit? 10](#_Toc195274094)

[34. How do you undo the last commit without losing changes? 10](#_Toc195274095)

[35. How do you discard all local changes? 10](#_Toc195274096)

[**📁 Tagging, Stashing & Cherry Picking** 11](#_Toc195274097)

[36. How do you create a tag in Git? 11](#_Toc195274098)

[37. How do you push a tag to remote? 11](#_Toc195274099)

[38. How do you stash changes temporarily? 11](#_Toc195274100)

[39. How do you apply stashed changes? 11](#_Toc195274101)

[40. How do you cherry-pick a specific commit from another branch? 11](#_Toc195274102)

[**🧠 DevOps & Real-world Git Scenarios** 11](#_Toc195274103)

[41. How do you manage secrets in GitHub? 11](#_Toc195274104)

[42. How do you roll back a deployment from GitHub history? 12](#_Toc195274105)

[43. How do you handle versioning in a GitOps pipeline? 12](#_Toc195274106)

[44. What branching strategy do you follow in DevOps projects? 13](#_Toc195274107)

[45. What is the difference between Git Flow, GitHub Flow, and trunk-based development? 13](#_Toc195274108)

# Basic Git Concepts

### What is Git, and how is it different from other VCS tools?

Git is a distributed Version Control System (VCS) that tracks changes in source code.  
Key differences from other VCS (e.g., SVN, CVS):  
✔ Distributed – Every user has a full copy of the repo (no single point of failure).  
✔ Fast & Efficient – Optimized for performance (branching, merging).  
✔ Offline Work – Commits can be made without a network connection.

### What is the difference between Git and GitHub?

| Git | GitHub |
| --- | --- |
| A local VCS tool for version control. | A cloud platform for hosting Git repositories. |
| Runs on your machine. | Provides collaboration features (PRs, Issues, CI/CD). |
| Open-source (free). | Offers free & paid plans (private repos). |

### What is the difference between git clone and git fork?

git clone → Creates a local copy of a remote repository.

fork (GitHub/GitLab) → Creates a copy of a repo under your account (used for contributing to others' projects).

### Explain the Git lifecycle: Working Directory, Staging Area, and Repository.

Working Directory – Where you modify files.

Staging Area (git add) – Prepares changes for commit.

Repository (git commit) – Permanently saves changes with a commit ID.

### What is the purpose of .gitignore?

✔ Specifies files/folders Git should ignore (e.g., logs, build files, env vars).  
✔ Prevents accidental commits of unnecessary/secret files.

**💥 Error Handling & Recovery**

### How do you resolve merge conflicts?

Merge conflicts occur when Git can't automatically merge changes. Steps to resolve:

Identify conflicts – Run git status to see conflicting files.

Open the file – Look for <<<<<<<, =======, >>>>>>> markers.

Edit & resolve – Manually choose which changes to keep.

Mark as resolved –

git add <file> # After fixing

git commit # Complete the merge

(Optional) Abort merge: git merge --abort.

### What to do when you accidentally commit sensitive info?

If pushed to remote:

Remove the file from history (see next question).

Rotate exposed secrets (passwords, API keys).

Force push to update remote:

git push --force

If not pushed:

Use git reset or git commit --amend to remove the sensitive data.

### 

### How do you remove a file from Git history?

Use git filter-branch or BFG Repo-Cleaner:

git filter-branch --force --index-filter \

"git rm --cached --ignore-unmatch <file>" \

--prune-empty --tag-name-filter cat -- --all

Then force push:

git push --force --all

Warning: Rewrites history—coordinate with your team!

### Difference between revert, reset, and checkout?

| Command | Effect | Use Case |
| --- | --- | --- |
| revert | Creates a new commit undoing a previous commit. | Safe for shared repos (doesn’t rewrite history). |
| reset | Moves HEAD (and branch pointer) to a past commit. Dangerous if pushed. | Discard local commits (--hard, --soft, --mixed). |
| checkout | Switches branches or discards unstaged changes. | git checkout <branch> or git checkout -- <file>. |

### What is "detached HEAD" state, and how do you exit it?

Detached HEAD means you’re not on a branch (e.g., after git checkout <commit-hash>).

To fix:

git switch -c <new-branch> # Create a new branch from the commit

# OR

git switch <existing-branch> # Return to a branch

Warning: Uncommitted changes in this state may be lost!

**🔐 GitHub-Specific (CI/CD, Collaboration)**

### How to Create a GitHub Repository

Steps:

Go to [GitHub](https://github.com/) → Click "+" → "New repository".

Enter:

Repository name (e.g., my-project).

Visibility (Public/Private).

Optional: Add a README, .gitignore, license.

Click "Create repository".

Post-creation:

# Link local repo to GitHub (if not initialized)

git remote add origin https://github.com/<username>/<repo>.git

git push -u origin main

### How to Fork a Repository

Steps:

Go to the GitHub repo you want to fork.

Click "Fork" (top-right).

Select your account as the destination.

After forking:

git clone https://github.com/<your-username>/<repo>.git # Clone your fork

git remote add upstream https://github.com/<original-owner>/<repo>.git # Track original

How to Raise a Pull Request (PR)

Steps:

Commit changes to a branch in your fork:

git checkout -b new-feature

git add .

git commit -m "Add new feature"

git push origin new-feature

On GitHub, go to your fork → "Pull Requests" → "New PR".

Select:

Base repo: Original repo’s main branch.

Head repo: Your fork’s new-feature branch.

Add a title/description → Click "Create PR".

### How to Enable Branch Protection Rules

Steps:

Go to Repo → Settings → Branches.

Under "Branch protection rules", click "Add rule".

Configure:

Branch name pattern (e.g., main).

Require PR approvals (e.g., 1+ reviewer).

Require status checks (e.g., CI passing).

Block force pushes.

Click "Save changes".

Note: Requires admin access to the repo.

### How GitHub Actions Work

GitHub Actions automates workflows (CI/CD, testing, deployments) via YAML files.

Key Components:

Workflow: Defined in .github/workflows/<file>.yml.

Events: Triggers like push, pull\_request, schedule.

Jobs/Steps: Tasks to run (e.g., build, test).

Example Workflow (Node.js CI):

name: Node.js CI

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-node@v4

with:

node-version: 20

- run: npm install

- run: npm test

How to Use:

Create .github/workflows/ci.yml in your repo.

GitHub automatically runs the workflow on the defined trigger.

### What are merge conflicts? How do you resolve them?

Merge conflicts occur when Git can't automatically combine changes from different branches (e.g., two people edit the same line).

To resolve:

Use git status to find conflicting files.

Edit files to keep the desired changes (remove <<<<<<<, =======, >>>>>>> markers).

Run git add and git commit to finalize the merge.

### What is origin?

origin is the default name for the remote repository (usually GitHub/GitLab).

It’s a shortcut for the URL you cloned from.

### Difference Between git fetch and git pull?

| Command | What It Does | When to Use |
| --- | --- | --- |
| git fetch | Downloads latest changes from the remote repo but does not merge them. | When you want to check updates before merging. |
| git pull | Fetches + merges changes into your current branch (git fetch + git merge). | When you want to directly update your branch with remote changes. |

**🔁 Branching & Merging**

### How do you create a new branch?

git branch branch-name

### How do you switch branches?

git checkout branch-name

### How do you create and switch to a new branch?

git checkout -b new-branch

### How do you merge a branch into the main branch?

git merge branch-name

**⬆️ Remote Repositories**

### How do you add a remote repository?

git remote add origin <url>

### How do you push changes to a remote repo?

git push origin branch-name

### How do you pull changes from a remote repo?

git pull

### How do you rename a remote?

git remote rename origin upstream

**💻 Core Git Commands**

### How do you initialize a Git repository?

git init

### How do you check the current status of your repo?

git status

### How do you stage files for commit?

git add filename or git add .

### How do you commit changes with a message?

git commit -m "message"

### How do you view commit history?

git log

**🔍 Advanced Git Commands**

### How do you check the difference between staged and working directory?

git diff

### How do you check the difference between last commit and current state?

git diff HEAD

### How do you revert a file to a previous commit?

git checkout commit-id filename

### How do you undo the last commit without losing changes?

git reset --soft HEAD~1

### How do you discard all local changes?

git reset –hard

**📁 Tagging, Stashing & Cherry Picking**

### How do you create a tag in Git?

git tag v1.0

### How do you push a tag to remote?

git push origin v1.0

### How do you stash changes temporarily?

git stash

### How do you apply stashed changes?

git stash apply

### How do you cherry-pick a specific commit from another branch?

git cherry-pick <commit-id>

**🧠 DevOps & Real-world Git Scenarios**

### How do you manage secrets in GitHub?

GitHub provides several ways to manage secrets securely:

GitHub Secrets (Encrypted Secrets):

Used in GitHub Actions workflows via secrets context (e.g., ${{ secrets.API\_KEY }}).

Stored encrypted and only accessible during workflow runs.

GitHub Environments:

Allows environment-specific secrets and deployment protection rules.

Third-party Integrations:

Tools like HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault can be integrated with GitHub Actions.

Avoid Hardcoding Secrets:

Never store secrets in code, config files, or Git history. Use .gitignore for files like .env.

### How do you roll back a deployment from GitHub history?

Rollback methods depend on the deployment strategy:

Using Git Revert:

git revert <commit-hash> # Reverts a specific commit

git push origin main

Using Git Reset (for local/force push):

git reset --hard <commit-hash> # Warning: Rewrites history

git push --force origin main # Use with caution in shared repos

CI/CD Rollback:

Trigger a redeployment of an older tagged release (e.g., v1.0.1).

Use GitHub Actions to re-run a workflow from a past commit.

Kubernetes (GitOps):

Tools like ArgoCD or Flux automatically sync to the previous Git commit.

### How do you handle versioning in a GitOps pipeline?

GitOps is a DevOps practice where Git is the single source of truth for your infrastructure and application deployments. Everything (code + infrastructure) is stored, version-controlled, and automated via Git.

Semantic Versioning (SemVer):

Tag releases in Git (v1.0.0, v1.1.0-alpha, etc.).

Automated Tagging:

Use tools like semantic-release to auto-generate versions based on commit messages (feat:, fix:, BREAKING CHANGE:).

GitOps Tools:

ArgoCD/Flux syncs Kubernetes manifests to a specific Git tag/branch.

Helm Charts:

Version Helm releases alongside Git tags.

### What branching strategy do you follow in DevOps projects?

Common strategies:

**Trunk-Based Development (Preferred for DevOps):**

Short-lived branches merged frequently into main.

Feature flags enable partial deployments.

**GitHub Flow:**

Branch off main → PR → Merge after review.

Simple and CI/CD-friendly.

**Git Flow (Legacy):**

Uses develop, feature/\*, release/\*, and main branches.

Overly complex for CI/CD pipelines.

### What is the difference between Git Flow, GitHub Flow, and trunk-based development?

| Aspect | Git Flow | GitHub Flow | Trunk-Based Development |
| --- | --- | --- | --- |
| Branches | main, develop, feature/\*, release/\*, hotfix/\* | main + short-lived feature branches | Only main (or very short-lived branches) |
| Release Cycle | Rigid (release branches) | Continuous (merge to main = deploy) | Continuous (direct to main) |
| Complexity | High | Low | Very low |
| CI/CD Suitability | Less ideal (slow merges) | Excellent | Best (frequent integrations) |
| Use Case | Traditional software (versioned releases) | Web apps, SaaS | DevOps, microservices, cloud-native |

Trend: Modern DevOps favors GitHub Flow or Trunk-Based Development for faster deployments.