**GIT - Core Version Control**

**1. What is Git and how does it work?**

**Answer:**  
Git is a distributed version control system used to track changes in source code during software development. Each developer has a full copy of the project history. Git uses snapshots (commits) rather than differences and supports branches and merges efficiently.

**2. Explain the Git workflow.**

**Answer:**  
Common Git workflow:

* git clone – Copy repo
* git checkout -b feature-branch – New branch
* git add . – Stage changes
* git commit -m "msg" – Commit
* git pull origin main – Sync with remote
* git push origin feature-branch – Push
* Create Pull Request for code review/merge

**3. What are the states in Git?**

**Answer:**

* **Working Directory** – Modified files
* **Staging Area (Index)** – Staged for commit
* **Local Repository** – Committed files
* **Remote Repository** – Shared on server

**4. Difference between git fetch and git pull?**

**Answer:**

* git fetch: Retrieves changes from remote but doesn't merge.
* git pull: Fetch + Merge.

**5. What is a rebase? How is it different from merge?**

**Answer:**

* **Merge**: Combines histories of two branches; may create merge commits.
* **Rebase**: Re-applies commits on top of another base; creates linear history.

Use rebase in feature branches, merge in main branches.

**6. What is the difference between origin, upstream, and head?**

**Answer:**

* origin: Default name for remote repository.
* upstream: The original repo forked from.
* HEAD: Current branch's latest commit reference.

**7. How do you resolve merge conflicts?**

**Answer:**

* git status to see conflicting files.
* Manually edit conflicts (look for <<<<<<<).
* git add <file> after resolving.
* git commit to finalize.

**💻 GitHub / GitLab / Bitbucket / Stash**

**8. How do GitHub and Git differ?**

**Answer:**  
Git is the version control system; GitHub is a cloud-based hosting platform for Git repositories with collaboration features like PRs, issues, and actions.

**9. What is a Pull Request (PR) in GitHub/GitLab/Bitbucket?**

**Answer:**  
A PR is a request to merge code from one branch to another (often feature → main). It allows for code review, CI checks, and discussions.

**10. How do GitLab pipelines differ from GitHub Actions?**

**Answer:**

| **Feature** | **GitLab CI/CD** | **GitHub Actions** |
| --- | --- | --- |
| YAML file | .gitlab-ci.yml | .github/workflows/\*.yml |
| Built-in Runner | Yes | Optional |
| Monorepo Support | Advanced | Less advanced |
| Cost | Free tiers + Paid | Free tiers + Paid |

**11. Explain GitLab Flow.**

**Answer:**  
GitLab Flow combines feature-driven development with CI/CD. It supports:

* Environment branches
* Code review and testing pipelines
* Merge requests as the deployment trigger

**12. How do you manage permissions in GitHub vs GitLab vs Bitbucket?**

**Answer:**

| **Tool** | **Permission Levels** |
| --- | --- |
| GitHub | Read, Triage, Write, Maintain, Admin |
| GitLab | Guest, Reporter, Developer, Maintainer, Owner |
| Bitbucket | Read, Write, Admin (at repo and project level) |

**13. Difference between GitHub and Bitbucket?**

**Answer:**

| **Feature** | **GitHub** | **Bitbucket** |
| --- | --- | --- |
| VCS | Git | Git & Mercurial |
| CI/CD | GitHub Actions | Bitbucket Pipelines |
| Integrations | GitHub Marketplace | Atlassian tools (Jira, Trello) |
| Access Control | Finer-grained | Team/project-oriented |

**14. What is Atlassian Stash?**

**Answer:**  
Atlassian Stash (now **Bitbucket Server**) is an on-premise Git repository management tool, useful for enterprise-grade control and integration with Jira, Bamboo, etc.

**15. Scenario: You accidentally committed secrets. How would you remove them?**

**Answer:**

1. Remove from file.
2. Run git rm --cached <file>
3. Commit and push.
4. Use tools like git filter-branch or BFG Repo-Cleaner to clean history.
5. Force push: git push --force

**16. How do you squash commits?**

**Answer:**

bash

CopyEdit

git rebase -i HEAD~n

# pick → squash (for older commits)

git push --force

**17. How do you enforce code quality in GitHub/GitLab/Bitbucket?**

**Answer:**

* PR templates
* Required reviews
* Status checks (CI/CD)
* Code coverage reports
* Protected branches
* Static analysis integrations (e.g., SonarQube)

**18. What is a Git tag? When do you use it?**

**Answer:**  
Git tags are used to mark release points (v1.0, v2.1).

bash

CopyEdit

git tag v1.0

git push origin v1.0

**19. How to revert a commit already pushed to remote?**

**Answer:**

bash

CopyEdit

git revert <commit\_hash> # Safe, creates a new commit

git push origin main

**20. How do you enforce branch policies in GitHub/GitLab/Bitbucket?**

**Answer:**

* Enable protected branches
* Require PRs with approvals
* Restrict direct push
* Enable CI/CD status checks
* Use CODEOWNERS for mandatory review

**✅ 1. Why use rebase if merge is safer?**

**Answer:**

* git merge preserves the complete history (safe, traceable), but can create **noisy commit history** with unnecessary merge commits.
* git rebase creates a **clean, linear history**, ideal for feature branch workflows.

**Use rebase when:**

* You want a **tidy commit history** (especially before merging feature branches).
* Working in **small teams** or solo, where history rewriting risk is minimal.
* You want to **replay local commits** on top of the updated remote base (main or develop).

**Why it can be risky:**

* Rewrites commit hashes.
* Not suitable for shared branches (can cause divergence/conflicts).

🔁 **Best practice:**

* Rebase locally before PR (git pull --rebase).
* Avoid rebasing public/shared branches.

**✅ 2. When would you use GitHub Actions over Jenkins?**

**Answer:**

| **Criteria** | **GitHub Actions** | **Jenkins** |
| --- | --- | --- |
| Setup | Native to GitHub, zero infra setup | Requires installation, plugins, server |
| Use case | Fast CI/CD for GitHub projects | Complex, enterprise-grade CI/CD |
| Cost | Free for public repos, free tier for private | Requires hosting (self-hosted/cloud) |
| Extensibility | GitHub Marketplace | Plugin ecosystem |
| Integration | Seamless GitHub integration | Needs webhook configuration |
| Secrets management | Built-in | Requires setup (Vault or Jenkins Credentials Plugin) |

✅ **Use GitHub Actions when:**

* You use GitHub as VCS.
* You want fast, integrated CI/CD.
* You need **simple pipelines** with minimal config.
* You want **event-driven workflows** (push, PR, issue comment).

**✅ 3. How do you set up CI/CD using GitLab pipeline with Docker runners?**

**Answer:**

**Steps to set up GitLab CI/CD:**

1. **Define .gitlab-ci.yml:**

yaml

CopyEdit

stages:

- build

- test

- deploy

build:

stage: build

script:

- docker build -t my-app .

test:

stage: test

script:

- docker run my-app npm test

deploy:

stage: deploy

script:

- ./deploy.sh

1. **Register Docker runner:**

bash

CopyEdit

sudo gitlab-runner register

# Choose 'docker' as executor

1. **Configure Docker in config.toml:**

toml

CopyEdit

[[runners]]

name = "docker-runner"

executor = "docker"

[runners.docker]

image = "node:16"

privileged = true

1. **Push .gitlab-ci.yml to repo** – pipeline triggers automatically.

💡**Advanced:** Use Docker-in-Docker (DinD) for full container workflows:

yaml

CopyEdit

services:

- docker:dind

**✅ 4. How do you use Git submodules and what are the risks?**

**Answer:**

**Submodules:** Allow embedding one Git repo as a sub-directory of another.

bash

CopyEdit

git submodule add <repo\_url> path/to/dir

git submodule update --init --recursive

**Use Cases:**

* Reuse shared libraries across multiple projects.
* Isolate dependencies that are actively version-controlled.

**Risks:**

* **Extra maintenance overhead** – each submodule has its own commit tracking.
* Can easily get **out of sync**.
* CI/CD complexity increases (requires recursive clones).
* New team members often forget to init/update them.

✅ **Alternatives:** Git subtree, or dependency management tools (Maven, npm, etc.)

**✅ 5. Can you recover a deleted remote branch?**

**Answer:**  
Yes — **only if you or someone else has the branch locally or knows the commit hash**.

**Option 1: Recover using reflog (locally):**

bash

CopyEdit

git reflog

# Find the last known commit of the branch

git checkout -b deleted-branch <commit\_hash>

git push origin deleted-branch

**Option 2: Another dev has local copy:**

bash

CopyEdit

git checkout <branch>

git push origin <branch>

**Option 3: From CI/CD logs or PRs:**

* Extract commit hash from PR or CI logs, then recreate branch:

bash

CopyEdit

git checkout -b recovered-branch <commit\_hash>

git push origin recovered-branch

💡**Tip:** Avoid accidental deletions by:

* Protecting main branches
* Setting branch deletion policies