- Introduction to DevOps

- Software Development Life Cycle

- Models (Waterfall, Agile)

- What is DevOps?

- Tools

- Version Control System

- Types of VCS

- Architecture of VCS

- Installation of Git Bash and Git

- How Git Works

- Branching

- Merging

- Merge Conflict

- Connecting Git with GitHub

- Concepts of Git

- Git Reset

- Git Restore

- Containers

- Types of Containers

- Docker Installation

- Launching a Container

- Docker Architecture

- Docker Hub

- Docker Commands

- Dockerfile

- Container Orchestration Tool

- Architecture of Kubernetes

- Components of Control Plane and Data Plane

- What is a POD?

- Launching a Kubernetes Cluster

- Resources/Objects in Kubernetes

- How Declarative Works

- Mini Kubernetes Cluster

- Controller Manager in Kubernetes

- Launching a Replica Set

- Deployment

- AWS CLI Installation

- Kubernetes EKS Launch Cluster

- Kubernetes Networking

- DevOps Interview Questions

**Introduction to DevOps**

**Introduction to DevOps:**

DevOps is a set of practices, cultural philosophies, and tools that integrate software development (Dev) and IT operations (Ops) to enhance the software development lifecycle. It aims to improve collaboration, automation, and continuous delivery to build, test, and release software faster and more reliably.

**Key Aspects of DevOps:**

1. Collaboration & Communication – Bridges the gap between developers and operations teams.
2. Automation – Automates repetitive tasks like testing, deployment, and monitoring.
3. CI/CD (Continuous Integration & Continuous Deployment) – Ensures faster and reliable code releases.
4. Infrastructure as Code (IaC) – Manages infrastructure using code (e.g., Terraform, Ansible).
5. Monitoring & Logging – Helps detect and resolve issues early using tools like Prometheus and ELK Stack.
6. Security (DevSecOps) – Integrates security within the DevOps pipeline.

**Benefits of DevOps:**

* Faster software delivery with reduced time-to-market.
* Higher reliability through automated testing and monitoring.
* Scalability & flexibility for cloud-based applications.
* Enhanced security by integrating security measures within CI/CD pipelines.

**Popular DevOps Tools:**

* CI/CD: Jenkins, GitHub Actions, GitLab CI
* Configuration Management: Ansible, Puppet, Chef
* Containerization & Orchestration: Docker, Kubernetes
* Monitoring & Logging: Prometheus, Grafana, ELK Stack
* Cloud Services: AWS, Azure, GCP

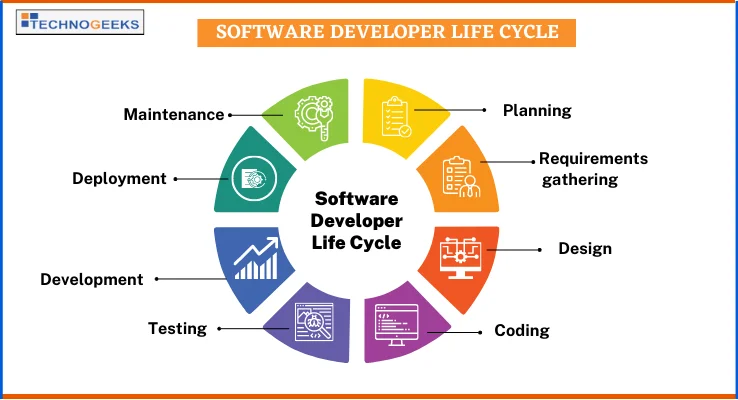
**Why DevOps is Important?**

With modern applications requiring fast and frequent releases, DevOps helps organizations stay competitive, agile, and scalable, ensuring that software is delivered efficiently and with high quality.

*Final Answer (Short Version for Interviews)*

*"DevOps is a culture and set of practices that integrate development and operations teams to automate and streamline software delivery. It focuses on CI/CD, automation, monitoring, and infrastructure as code, leading to faster releases, higher reliability, and improved security. Key tools include Jenkins, Docker, Kubernetes, and cloud platforms like AWS and Azure."*

**Software Development Life Cycle [ SDLC ]**

****

**Software Development Life Cycle (SDLC) – Explanation for DevOps Engineer Interview**

**What is SDLC?**

The Software Development Life Cycle (SDLC) is a structured process for developing, testing, deploying, and maintaining software efficiently. It ensures that the software meets business requirements, quality standards, and security while being delivered within time and budget constraints.

**Phases of SDLC**

1. Requirement Gathering & Analysis – Understanding business needs, functional & non-functional requirements.
2. Planning – Defining project scope, timelines, cost, and risks.
3. Design – Creating architectural and technical design, including database models and system structure.
4. Development (Implementation) – Writing and integrating code as per design specifications.
5. Testing – Ensuring functionality, security, and performance using automation/manual testing.
6. Deployment – Releasing software in production using CI/CD pipelines for automation.
7. Maintenance & Monitoring – Ensuring performance, handling bug fixes, updates, and monitoring system health.

**SDLC & DevOps Integration**

In DevOps, SDLC is enhanced by automation, continuous integration, and continuous deployment (CI/CD), ensuring fast and reliable software delivery.

Traditional SDLC vs. DevOps-Driven SDLC:

| Aspect | Traditional SDLC | DevOps SDLC |
| --- | --- | --- |
| Deployment | Manual & periodic | Continuous & automated |
| Testing | Late-stage testing | Continuous testing & automation |
| Collaboration | Dev & Ops work separately | Dev & Ops work together |
| Infrastructure | Managed manually | Infrastructure as Code (IaC) |

**DevOps Engineer Interview Questions on SDLC**

**Basic SDLC Questions:**

1. **What is SDLC? Explain its importance.**
   * SDLC is a structured approach to software development that includes planning, designing, developing, testing, deploying, and maintaining software. It ensures quality, security, and efficiency in software projects.
2. **What are the different phases of SDLC?**
   * The main phases are Requirement Gathering, Planning, Design, Development, Testing, Deployment, and Maintenance.
3. **Which SDLC model is best for DevOps? Why?**
   * The Agile Model is best for DevOps as it promotes continuous integration, delivery, and collaboration between teams, enabling faster releases and feedback loops.

**DevOps-Specific SDLC Questions:**

1. **How does DevOps fit into SDLC?**
   * DevOps enhances SDLC by automating development, testing, deployment, and monitoring through CI/CD pipelines, Infrastructure as Code (IaC), and real-time feedback mechanisms.
2. **What are the challenges of traditional SDLC, and how does DevOps solve them?**
   * Traditional SDLC faces slow deployments, lack of collaboration, and late-stage testing. DevOps solves these by implementing CI/CD, automation, early testing, and real-time monitoring.
3. **How do CI/CD pipelines enhance the SDLC process?**
   * CI/CD automates code integration, testing, and deployment, reducing manual errors, ensuring faster releases, and maintaining software quality across environments.

**Advanced DevOps SDLC Questions:**

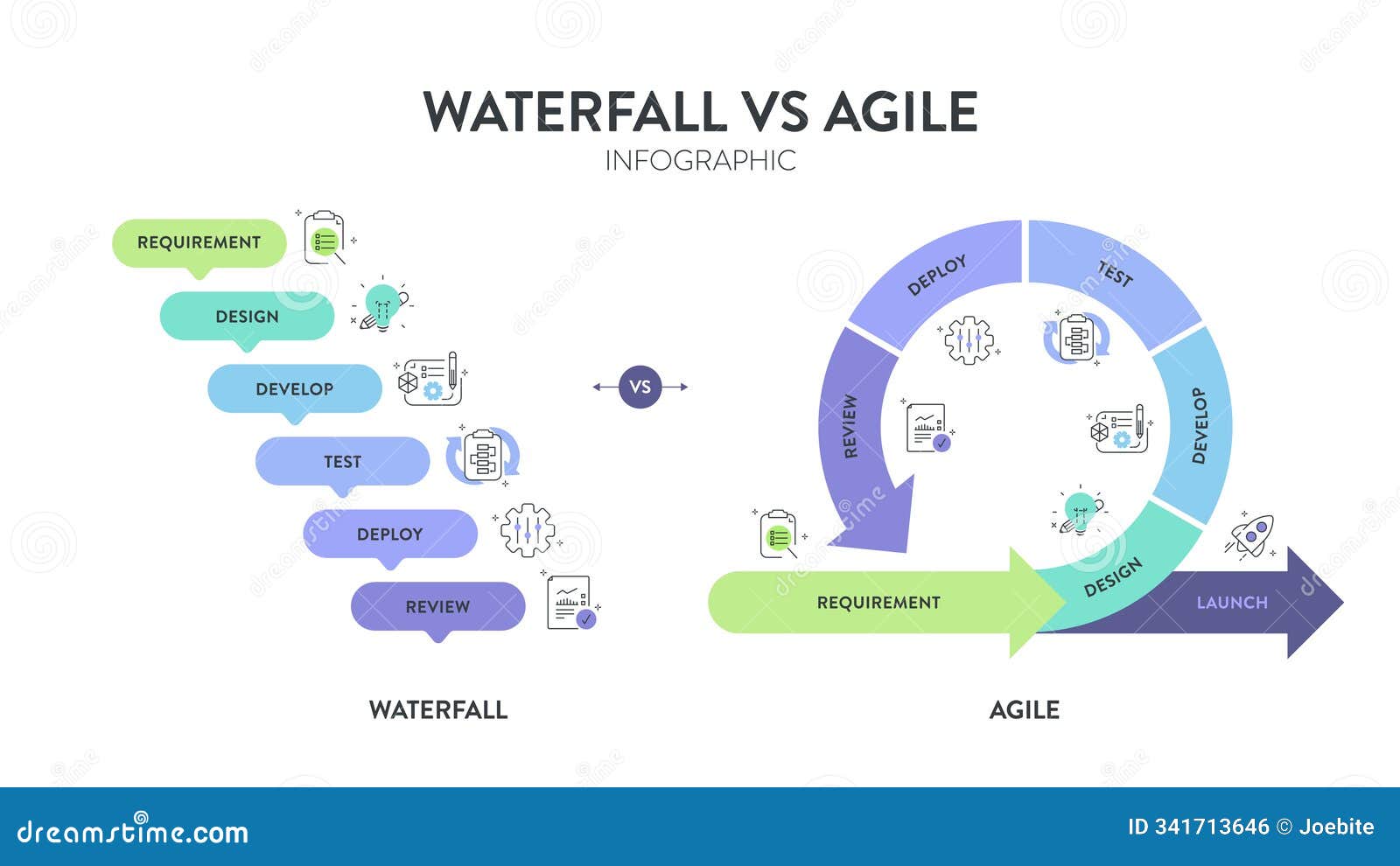
1. **How does Infrastructure as Code (IaC) impact the SDLC?**
   * IaC automates infrastructure provisioning, making deployments consistent, scalable, and efficient, reducing human errors, and improving SDLC speed.
2. **How do you ensure security in SDLC with DevOps (DevSecOps)?**
   * By integrating security tools into CI/CD pipelines, performing automated security scans, and following secure coding practices.
3. **How does containerization (Docker, Kubernetes) impact SDLC?**
   * It ensures consistency across environments, improves scalability, and simplifies deployment and rollback processes.
4. **Which DevOps tools do you use for different SDLC phases?**

* Requirement Gathering: Jira, Confluence
* Development: Git, GitHub, GitLab
* Testing: Selenium, JUnit, SonarQube
* Deployment: Jenkins, GitHub Actions, Terraform
* Monitoring & Maintenance: Prometheus, Grafana, ELK Stack

**Final Answer (Short Version for Interviews)**

*"SDLC is the structured process for developing and maintaining software. It includes phases like requirement gathering, design, development, testing, deployment, and maintenance. In DevOps, SDLC is enhanced through CI/CD automation, Infrastructure as Code (IaC), and continuous monitoring. This accelerates software delivery, improves quality, and ensures faster issue resolution."*

**Models (Waterfall, Agile)**



**SDLC Models – Waterfall vs. Agile (For DevOps Engineer Interview)**

**1. Waterfall Model**

The **Waterfall Model** is a traditional, sequential SDLC approach where each phase is completed before moving to the next. It follows a **linear structure** with strict documentation and well-defined stages.

**Phases of the Waterfall Model:**

1. **Requirement Gathering** – Detailed documentation of project requirements.
2. **System Design** – Architectural and technical design.
3. **Implementation (Development)** – Writing the actual code.
4. **Testing** – Validating functionality and fixing defects.
5. **Deployment** – Releasing the final product to users.
6. **Maintenance** – Bug fixes and updates after deployment.

**Advantages:**

✅ Simple and easy to manage.  
✅ Well-documented with clear phases.  
✅ Works well for small or well-defined projects.

**Disadvantages:**

❌ Rigid, does not allow changes in requirements.  
❌ Late testing, leading to delayed defect detection.  
❌ Not suitable for complex or evolving projects.

**2. Agile Model**

The **Agile Model** is an iterative and flexible approach where software is developed in small, incremental releases called **sprints** (typically 2-4 weeks). Agile promotes **continuous collaboration**, **feedback**, and **adaptability** to changing requirements.

**Phases of the Agile Model (Scrum Framework):**

1. **Concept & Requirement Analysis** – Gather high-level requirements.
2. **Sprint Planning** – Define features for the next sprint.
3. **Development & Testing (Iterative)** – Code, test, and review within sprints.
4. **Demo & Feedback** – Showcase working software to stakeholders.
5. **Deployment & Maintenance** – Continuous releases with updates.

**Advantages:**

✅ Highly flexible and adaptable to changes.  
✅ Faster time to market with continuous delivery.  
✅ Improved collaboration between teams (Dev, Ops, QA, Business).

**Disadvantages:**

❌ Requires strong team coordination and communication.  
❌ Difficult to estimate time and cost accurately.  
❌ May lead to scope creep if not managed properly.

**Waterfall vs. Agile – Key Differences**

| **Aspect** | **Waterfall Model** | **Agile Model** |
| --- | --- | --- |
| **Approach** | Sequential (Step-by-Step) | Iterative & Incremental |
| **Flexibility** | Rigid, changes are difficult | Highly flexible to changes |
| **Delivery** | Final product at the end | Continuous small releases |
| **Testing** | Performed after development | Continuous testing in every sprint |
| **Risk Handling** | High risk due to late testing | Low risk due to early feedback |
| **Collaboration** | Limited between teams | Strong DevOps collaboration |
| **Use Case** | Small, well-defined projects | Large, evolving projects |

**DevOps Engineer Interview Questions on Waterfall & Agile**

**Basic SDLC Model Questions:**

1. **What is the Waterfall Model? When is it used?**
   * *The Waterfall Model is a sequential software development approach where each phase is completed before moving to the next. It is used for simple projects with well-defined requirements.*
2. **What is the Agile Model? Why is it preferred in DevOps?**
   * *The Agile Model is an iterative approach that delivers software in small, continuous increments. DevOps prefers Agile because it promotes faster releases, automation, and continuous feedback.*

**DevOps-Specific Questions on Agile vs. Waterfall:**

1. **Why is the Waterfall Model not suitable for DevOps?**
   * *Waterfall lacks flexibility, has late testing phases, and does not support continuous integration or deployment, making it incompatible with DevOps principles.*
2. **How does Agile align with DevOps?**
   * *Agile and DevOps both emphasize collaboration, automation, continuous testing, and rapid releases, making them complementary for modern software development.*
3. **What are the key Agile methodologies used in DevOps?**
   * *Scrum, Kanban, and SAFe (Scaled Agile Framework) are commonly used Agile methodologies in DevOps.*

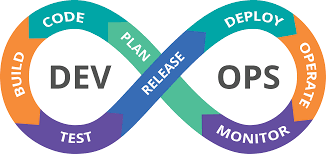
**Advanced DevOps Agile Model Questions:**

1. **What is a sprint in Agile, and how does it help in DevOps?**
   * *A sprint is a short development cycle (2-4 weeks) in Agile that focuses on delivering a working feature. In DevOps, sprints align with CI/CD to automate and release software frequently.*
2. **How do CI/CD pipelines improve Agile development?**
   * *CI/CD automates code integration, testing, and deployment, ensuring continuous delivery and minimizing manual effort in Agile workflows.*
3. **What tools are used to implement Agile in DevOps?**
   * *Jira (for Agile project management), Jenkins/GitHub Actions (for CI/CD), Selenium (for automated testing), and Docker/Kubernetes (for deployment).*

**Final Answer (Short Version for Interviews)**

*"Waterfall is a linear, step-by-step SDLC model suited for small, well-defined projects, but it lacks flexibility and adaptability. Agile, on the other hand, is an iterative and incremental model that enables rapid software releases, collaboration, and continuous integration. DevOps prefers Agile as it aligns with automation, continuous delivery, and iterative improvements."*

**What is DevOps**



**Introduction to DevOps**

**DevOps** is a **culture, methodology, and set of practices** that integrate **Development (Dev)** and **Operations (Ops)** teams to automate and streamline the software development lifecycle (SDLC). The goal of DevOps is to enable **faster, more reliable software delivery** through **continuous integration, continuous deployment (CI/CD), automation, and collaboration**.

**Key Principles of DevOps**

1. **Collaboration & Communication** – Bridging the gap between development and operations.
2. **Automation** – Automating repetitive tasks like testing, deployment, and monitoring.
3. **Continuous Integration & Continuous Deployment (CI/CD)** – Ensuring fast and reliable releases.
4. **Infrastructure as Code (IaC)** – Managing infrastructure through code (Terraform, Ansible).
5. **Monitoring & Feedback** – Using real-time monitoring to improve system performance.
6. **Security (DevSecOps)** – Integrating security within the DevOps lifecycle.

**DevOps Lifecycle & Stages**

1. **Plan** – Define project scope and roadmap (Jira, Confluence).
2. **Develop** – Write and integrate code (Git, GitHub, GitLab).
3. **Build** – Compile and package code (Maven, Gradle).
4. **Test** – Automate testing for bugs and performance (Selenium, JUnit).
5. **Release** – Manage version control and approvals (GitHub Actions, GitLab CI).
6. **Deploy** – Deploy applications automatically (Docker, Kubernetes, Terraform).
7. **Monitor & Operate** – Track system performance and errors (Prometheus, Grafana).

**Benefits of DevOps**

✅ **Faster Deployment:** Reduces time-to-market with automation.  
✅ **Higher Reliability:** Automated testing and monitoring ensure fewer failures.  
✅ **Scalability:** Supports cloud-native and microservices architectures.  
✅ **Cost Efficiency:** Reduces infrastructure costs through optimization.  
✅ **Security & Compliance:** Implements security checks in CI/CD pipelines.

**DevOps Tools & Technologies**

| **Category** | **Popular Tools** |
| --- | --- |
| **Version Control** | Git, GitHub, GitLab, Bitbucket |
| **CI/CD** | Jenkins, GitHub Actions, GitLab CI/CD |
| **Configuration Management** | Ansible, Puppet, Chef |
| **Containerization** | Docker, Kubernetes |
| **Infrastructure as Code (IaC)** | Terraform, AWS CloudFormation |
| **Monitoring & Logging** | Prometheus, Grafana, ELK Stack |
| **Security (DevSecOps)** | SonarQube, Snyk, Trivy |

**DevOps Engineer Interview Questions & Answers**

**Basic DevOps Questions:**

1. **What is DevOps? Why is it important?**
   * *DevOps is a culture and set of practices that combine development and operations to improve software delivery. It enables faster releases, automation, and collaboration, ensuring high-quality software deployment.*
2. **How is DevOps different from traditional IT operations?**
   * *Traditional IT operations follow a siloed approach with long release cycles, while DevOps integrates teams, automates processes, and enables continuous software delivery.*
3. **What are the key benefits of DevOps?**
   * *Faster releases, improved collaboration, automation, cost efficiency, and enhanced security.*

**DevOps Lifecycle & CI/CD Questions:**

1. **What is CI/CD in DevOps?**
   * *Continuous Integration (CI) is the practice of frequently merging code changes, while Continuous Deployment (CD) automates software release to production.*
2. **What are the key stages of a CI/CD pipeline?**
   * *Code, Build, Test, Release, Deploy, Monitor.*
3. **Which CI/CD tools have you worked with?**
   * *Jenkins, GitHub Actions, GitLab CI/CD, CircleCI.*

**DevOps Tools & Automation Questions:**

1. **What is Infrastructure as Code (IaC)? Why is it important?**
   * *IaC allows managing infrastructure using code, making it automated, scalable, and consistent. Examples: Terraform, Ansible.*
2. **How do you manage configuration in DevOps?**
   * *Using tools like Ansible, Puppet, and Chef to automate configuration management.*
3. **What is containerization, and how does it help DevOps?**
   * *Containerization packages applications and dependencies into isolated environments, ensuring consistency across development, testing, and production. Tools: Docker, Kubernetes.*

**Advanced DevOps & Security (DevSecOps) Questions:**

1. **What is DevSecOps, and why is it important?**

* *DevSecOps integrates security into DevOps workflows by automating security checks and vulnerability assessments throughout CI/CD.*

1. **How do you monitor applications in DevOps?**

* *Using tools like Prometheus, Grafana, ELK Stack for performance monitoring, logging, and alerting.*

1. **How do you handle failures in a DevOps pipeline?**

* *Implement rollback strategies, automate error detection, and use observability tools to detect and fix issues quickly.*

**Final Answer (Short Version for Interviews)**

*"DevOps is a culture that integrates software development and operations to enhance collaboration, automate workflows, and improve software delivery. It emphasizes CI/CD, Infrastructure as Code (IaC), containerization, and real-time monitoring to ensure faster, more reliable, and scalable software releases."*

**Tool**

***DevOps Tools – Categories & Popular Choices***

*DevOps relies on a variety of tools to* ***automate, monitor, and streamline*** *the software development lifecycle (SDLC). Below is a list of key DevOps tools categorized by their functionalities.*

***1. Version Control Tools (Track changes in code and collaborate effectively)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Git*** | *Distributed version control system for tracking source code.* |
| ***GitHub*** | *Cloud-based Git repository with collaboration features.* |
| ***GitLab*** | *Provides CI/CD pipelines along with version control.* |
| ***Bitbucket*** | *Integrates with Jira and provides Git repository management.* |

***2. Continuous Integration / Continuous Deployment (CI/CD) Tools (Automate code integration, testing, and deployment)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Jenkins*** | *Open-source CI/CD automation server.* |
| ***GitHub Actions*** | *CI/CD automation directly within GitHub.* |
| ***GitLab CI/CD*** | *Built-in CI/CD functionality in GitLab.* |
| ***CircleCI*** | *Cloud-based CI/CD platform.* |
| ***Travis CI*** | *Simple CI/CD tool for GitHub projects.* |

***3. Configuration Management Tools (Manage infrastructure and system configurations as code)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Ansible*** | *Agentless configuration management and automation.* |
| ***Puppet*** | *Automates configuration and state management.* |
| ***Chef*** | *Infrastructure automation using Ruby-based DSL.* |
| ***SaltStack*** | *Manages large-scale infrastructure automation.* |

***4. Infrastructure as Code (IaC) Tools (Provision and manage infrastructure using code)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Terraform*** | *Declarative IaC tool for provisioning cloud infrastructure.* |
| ***AWS CloudFormation*** | *Automates AWS infrastructure setup.* |
| ***Pulumi*** | *IaC using standard programming languages like Python, TypeScript.* |

***5. Containerization & Orchestration Tools (Package and manage applications in lightweight containers)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Docker*** | *Creates, manages, and runs containers.* |
| ***Kubernetes*** | *Orchestrates containerized applications.* |
| ***Podman*** | *Alternative to Docker for managing containers.* |
| ***OpenShift*** | *Enterprise Kubernetes platform.* |

***6. Monitoring & Logging Tools (Track system performance, logs, and alerts)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Prometheus*** | *Monitoring and alerting for applications.* |
| ***Grafana*** | *Visualizes data from Prometheus and other sources.* |
| ***ELK Stack*** | *(Elasticsearch, Logstash, Kibana) for centralized logging.* |
| ***Datadog*** | *Cloud monitoring and analytics.* |
| ***New Relic*** | *Application performance monitoring.* |

***7. Security & DevSecOps Tools (Integrate security within the DevOps pipeline)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***SonarQube*** | *Code quality and security analysis.* |
| ***Snyk*** | *Detects vulnerabilities in dependencies.* |
| ***Trivy*** | *Container security scanning.* |
| ***OWASP ZAP*** | *Security testing for web applications.* |

***8. Cloud & DevOps Platforms (Host and manage DevOps infrastructure on the cloud)***

| ***Cloud Provider*** | ***DevOps Services*** |
| --- | --- |
| ***AWS*** | *AWS CodePipeline, AWS Lambda, AWS ECS, AWS EKS* |
| ***Microsoft Azure*** | *Azure DevOps, Azure Kubernetes Service* |
| ***Google Cloud*** | *Google Kubernetes Engine (GKE), Cloud Build* |

***9. Communication & Collaboration Tools (Enhance teamwork and issue tracking)***

| ***Tool*** | ***Description*** |
| --- | --- |
| ***Slack*** | *Real-time communication tool for DevOps teams.* |
| ***Jira*** | *Project management and issue tracking.* |
| ***Trello*** | *Kanban board for tracking DevOps workflows.* |
| ***Confluence*** | *Documentation and knowledge-sharing platform.* |

***Summary of DevOps Toolchain***

| ***Category*** | ***Key Tools*** |
| --- | --- |
| ***Version Control*** | *Git, GitHub, GitLab* |
| ***CI/CD*** | *Jenkins, GitHub Actions, GitLab CI/CD* |
| ***Configuration Management*** | *Ansible, Puppet, Chef* |
| ***IaC*** | *Terraform, AWS CloudFormation* |
| ***Containers & Orchestration*** | *Docker, Kubernetes, OpenShift* |
| ***Monitoring & Logging*** | *Prometheus, Grafana, ELK Stack* |
| ***Security & DevSecOps*** | *SonarQube, Snyk, Trivy* |
| ***Cloud Providers*** | *AWS, Azure, Google Cloud* |
| ***Collaboration*** | *Slack, Jira, Trello* |

***PRACTICAL***

1. **mkdir devops** – Creates a new directory named "devops".
2. **cd** **mkdir** – Incorrect command; likely meant cd devops to navigate into the "devops" directory.
3. **nano 1.txt** → content – Opens or creates "1.txt" in the nano editor to add content.
4. **nano 2.txt** → content – Opens or creates "2.txt" in the nano editor to add content.
5. **git init** – Initializes an empty Git repository in the current directory.
6. **git status** – Shows the status of changes (staged, unstaged, or untracked files).
7. **git add 1.txt 2.txt / git add .** – Stages specific files or all files for commit.
8. **git commit -m "msg what you needed"** – Commits staged files with a message.
9. **git status** – Again checks the repository status after committing changes.
10. **Syncing repositories** – Ensures the local repository is in sync with the remote one.
11. **git remote add origin "<URL to Repository>"** – Links the local repository to a remote GitHub/GitLab repository.
12. **git push origin master** – Pushes committed changes to the "master" branch on the remote repository.
13. **git config** --global credential.helper store – Saves credentials to avoid re-entering them for each push/pull.
14. **git clone <URL>** – Clones a remote repository to the local system.
15. **git pull <URL>** – Fetches and merges changes from the remote repository.
16. **git branch <name-of-the-branch>** – Creates a new branch with the specified name.
17. **git branch -D <branch-name>** – Deletes the specified branch forcefully.
18. **git checkout feature1** – Switches to the "feature1" branch.
19. **git log** – Shows the commit history.
20. **git stash** – Temporarily saves uncommitted changes for later use.
21. **git stash -u** – Stashes both tracked and untracked files.
22. **git stash pop** – Restores stashed changes and removes them from the stash list.
23. **git checkout -- 2.txt** – Incorrect syntax; likely meant git checkout -- 2.txt to undo changes in "2.txt".
24. **git diff <commit-id of version x> <commit-id of version y>** – Compares differences between two commits.

**Version Control System (VCS) - Explanation & Interview Questions**

A **Version Control System (VCS)** is a tool that helps developers manage changes to source code over time. It allows multiple developers to collaborate on a project while keeping track of every modification made to files.

**Key Features of VCS:**

* Tracks changes in files and directories.
* Supports collaboration among multiple developers.
* Provides rollback functionality in case of mistakes.
* Maintains different versions (branches) of the code.

**Types of Version Control Systems (VCS)**

There are **three main types** of VCS:

1. **Local Version Control System (LVCS)**
   * Stores versions locally on a single machine.
   * Example: RCS (Revision Control System).
2. **Centralized Version Control System (CVCS)**
   * A single server stores all versions of the code, and developers access it remotely.
   * Example: SVN (Apache Subversion), Perforce.
3. **Distributed Version Control System (DVCS)**
   * Each developer has a local copy of the entire repository.
   * Example: **Git, Mercurial**.

**Architecture of VCS**

The **architecture of a VCS** defines how files and versions are stored and managed.

**Git Architecture (DVCS)**

1. **Working Directory** – The local copy of files.
2. **Staging Area (Index)** – Stores changes before committing them.
3. **Local Repository** – The version history stored on the developer's machine.
4. **Remote Repository** – A shared repository (e.g., GitHub, GitLab, Bitbucket) where all developers can push or pull changes.

**Installation of Git and Git Bash**

**Steps to Install Git on Windows**

1. **Download Git** from [Git official website](https://git-scm.com/downloads).
2. Run the installer and follow the installation steps.
3. Choose **Git Bash** as the command-line interface.
4. Complete the installation and verify using:
5. git --version

**Steps to Install Git on Linux (Ubuntu/Debian)**

1. Open the terminal.
2. Run the following command:
3. sudo apt update
4. sudo apt install git
5. Verify the installation:
6. git --version

**Git Restore**

git restore is used to discard changes in the working directory.

**Common Use Cases of git restore:**

1. **Discard changes in a file (before staging):**
2. git restore filename
3. **Discard changes in multiple files:**
4. git restore .
5. **Restore a file from a specific commit:**
6. git restore --source=<commit-hash> filename

**Important Interview Questions and Answers**

**1. What is a Version Control System (VCS)?**

**Answer:**  
A Version Control System is a tool that tracks changes in files and enables multiple developers to collaborate on projects efficiently. It helps in managing different versions of code and provides rollback functionality.

**2. What are the types of Version Control Systems?**

**Answer:**

* **Local VCS** (RCS)
* **Centralized VCS** (CVCS - SVN, Perforce)
* **Distributed VCS** (DVCS - Git, Mercurial)

**3. What are the key differences between Centralized and Distributed VCS?**

| **Feature** | **Centralized VCS** | **Distributed VCS** |
| --- | --- | --- |
| Storage | Central server | Each user has a copy |
| Internet Dependency | Required for most operations | Not required for local commits |
| Examples | SVN, Perforce | Git, Mercurial |

**4. What are the key components of Git Architecture?**

**Answer:**

* **Working Directory** – Local files.
* **Staging Area** – Holds changes before commit.
* **Local Repository** – Stores commit history.
* **Remote Repository** – Shared repository on platforms like GitHub.

**5. How do you install Git on Windows/Linux?**

**Answer:**

* **Windows:** Download from Git website, install, and use git --version to verify.
* **Linux:** Run sudo apt install git and verify using git --version.

**6. What is git restore and how is it different from git checkout?**

**Answer:**

* git restore is used to discard changes before staging.
* git checkout is used to switch branches or restore files.

Example:

git restore file.txt # Discards changes

git checkout branch-name # Switches branch

**How Git Works?**

Git is a **Distributed Version Control System (DVCS)** that allows multiple developers to collaborate on projects efficiently. It keeps track of changes to files, enables versioning, and allows users to revert to previous versions if necessary.

**Key Components of Git Workflow:**

1. **Working Directory** – Contains the local project files.
2. **Staging Area (Index)** – Stores changes before committing.
3. **Local Repository** – Contains all committed changes.
4. **Remote Repository** – A shared repository (e.g., GitHub, GitLab).

**Basic Git Workflow:**

1. **Modify files** → Make changes in the working directory.
2. **Stage changes** → Use git add filename to move changes to the staging area.
3. **Commit changes** → Use git commit -m "message" to save changes in the local repository.
4. **Push changes** → Use git push origin branch\_name to update the remote repository.
5. **Pull updates** → Use git pull origin branch\_name to sync changes from remote.

**Branching in Git**

Branching allows developers to create separate development paths without affecting the main codebase.

**Types of Branches:**

1. **Main (Master) Branch** – The production-ready code.
2. **Feature Branch** – Used to develop new features.
3. **Bugfix Branch** – Created to fix issues in the code.
4. **Hotfix Branch** – Immediate fixes for production issues.

**Creating and Switching Branches:**

* **Create a new branch:**
* git branch new-feature
* **Switch to another branch:**
* git checkout new-feature

Or (in newer Git versions):

git switch new-feature

* **List all branches:**
* git branch
* **Delete a branch:**
* git branch -d new-feature

**Merging in Git**

Merging is used to integrate changes from one branch into another.

**Types of Merging:**

1. **Fast-forward Merge**
   * Happens when there are no new commits in the main branch since the feature branch was created.
   * Command:
   * git checkout main
   * git merge new-feature
2. **Recursive (Three-Way) Merge**
   * Happens when both branches have new commits.
   * Command:
   * git checkout main
   * git merge new-feature

**Merge Conflict**

A **merge conflict** occurs when Git cannot automatically resolve differences between merged branches.

**When Do Merge Conflicts Happen?**

* When the same file is modified in both branches.
* When a file is deleted in one branch and modified in another.

**Resolving Merge Conflicts:**

1. **Identify the conflicting files:**
2. git status
3. **Open the conflicted file:**
   * The file will contain conflict markers (<<<<<<<, =======, >>>>>>>).
4. **Manually edit the file** and choose the correct changes.
5. **Mark the conflict as resolved:**
6. git add filename
7. **Commit the resolved merge:**
8. git commit -m "Resolved merge conflict"

**Important Interview Questions and Answers**

**1. How does Git work internally?**

**Answer:**  
Git stores changes as a series of snapshots (commits). Each commit points to a previous commit, forming a history. The local and remote repositories enable distributed version control.

**2. What is the purpose of branching in Git?**

**Answer:**  
Branching allows developers to work on separate features or fixes without affecting the main codebase. It enables parallel development.

**3. How do you create and switch branches in Git?**

**Answer:**

git branch new-feature # Create branch

git switch new-feature # Switch to branch

**4. What are the different types of merges in Git?**

**Answer:**

* **Fast-forward Merge** – Moves the branch pointer forward when no divergence exists.
* **Three-way Merge** – Combines changes from both branches when they have diverged.

**5. What is a merge conflict and how do you resolve it?**

**Answer:**  
A merge conflict occurs when Git cannot automatically merge changes. It is resolved by manually editing the conflicted file, staging it, and committing the changes.

**6. How do you delete a branch in Git?**

**Answer:**

git branch -d branch\_name # Delete a local branch

git push origin --delete branch\_name # Delete a remote branch

**Connecting Git with GitHub**

GitHub is a cloud-based hosting service for Git repositories. Connecting Git with GitHub allows users to store and manage their code online while collaborating with others.

**Steps to Connect Git with GitHub**

**1. Create a GitHub Account**

* Go to [GitHub](https://github.com/) and create an account if you don’t have one.

**2. Install Git (if not installed)**

* Check if Git is installed:
* git --version
* If not installed, download it from [Git SCM](https://git-scm.com/) and install it.

**3. Configure Git with GitHub**

* Set up your name and email (same as your GitHub email):
* git config --global user.name "Your Name"
* git config --global user.email "your-email@example.com"

**4. Generate SSH Key and Add it to GitHub**

* Generate SSH key:
* ssh-keygen -t rsa -b 4096 -C "your-email@example.com"
* Copy the key:
* cat ~/.ssh/id\_rsa.pub
* Add the key to **GitHub**:
  + Go to **GitHub > Settings > SSH and GPG keys > New SSH Key**
  + Paste the key and save it.

**5. Clone a Repository from GitHub**

git clone git@github.com:username/repository.git

**6. Push a Local Repository to GitHub**

git init # Initialize Git in the local project

git add . # Stage all files

git commit -m "Initial commit"

git branch -M main # Rename branch to main

git remote add origin git@github.com:username/repository.git

git push -u origin main # Push to GitHub

**Concepts of Git**

Git has several fundamental concepts that help in version control.

**1. Repository (Repo)**

A **repository** is a collection of files managed by Git. It contains the complete history of changes.

* **Local Repository** – Stored on the user’s machine.
* **Remote Repository** – Hosted on platforms like GitHub, GitLab.

**2. Working Directory**

The folder where files are being edited.

**3. Staging Area (Index)**

A place where changes are added before committing.

**4. Commit**

A snapshot of changes in the repository.

git commit -m "Commit message"

**5. Branch**

A copy of the code where new changes are made.

**6. Merge**

Combining changes from one branch into another.

**7. Remote**

A reference to a repository hosted on platforms like GitHub.

git remote add origin git@github.com:username/repository.git

**8. Pull**

Fetch and merge changes from a remote repository.

git pull origin main

**9. Push**

Upload local commits to a remote repository.

git push origin main

**Git Reset**

git reset is used to **undo changes** in Git by modifying the commit history.

**Types of Git Reset**

1. **Soft Reset** – Moves HEAD to the previous commit but keeps changes staged.
2. git reset --soft HEAD~1
   * The last commit is removed, but the changes remain staged.
3. **Mixed Reset** *(Default)* – Moves HEAD to the previous commit and unstages changes.
4. git reset --mixed HEAD~1
   * The last commit is removed, and changes move to the working directory.
5. **Hard Reset** – Removes the commit and all changes permanently.
6. git reset --hard HEAD~1
   * This **cannot be undone** if changes are not backed up.

**Important Interview Questions and Answers**

**1. How do you connect Git with GitHub?**

**Answer:**

* Generate an SSH key and add it to GitHub.
* Use git remote add origin <repository-url> to connect a local repository with GitHub.
* Use git push -u origin main to push code.

**2. What is the difference between git reset --soft, git reset --mixed, and git reset --hard?**

| **Reset Type** | **Effect on HEAD** | **Effect on Staging Area** | **Effect on Working Directory** |
| --- | --- | --- | --- |
| --soft | Moves HEAD back | Keeps changes staged | Keeps changes in working directory |
| --mixed | Moves HEAD back | Unstages changes | Keeps changes in working directory |
| --hard | Moves HEAD back | Removes changes | Deletes changes permanently |

**3. What are the main concepts of Git?**

**Answer:**

* **Repository** – Storage for all versions of the project.
* **Branch** – Independent lines of development.
* **Commit** – Saves changes in Git history.
* **Merge** – Combines branches.
* **Push/Pull** – Send and receive changes with a remote repository.

**4. How do you undo the last commit but keep the changes?**

**Answer:**

git reset --soft HEAD~1

**5. How do you remove the last commit and all changes permanently?**

**Answer:**

git reset --hard HEAD~1

**6. How do you check remote repository URLs in Git?**

**Answer:**

git remote -v

***Top 10 important Git & GitHub interview questions***

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

Git is a **distributed version control system** that tracks changes in files and allows multiple developers to collaborate on projects. It enables branching, merging, and rollback of changes efficiently.

💡 **Application:**

* Used in software development to maintain different versions of code.
* Helps teams work independently on features and merge them later.

**2. What is the difference between Git and GitHub?**

* **Git** is a version control tool used to manage source code locally.
* **GitHub** is a cloud-based platform for hosting Git repositories.

💡 **Application:**

* Git is used for version control in local systems.
* GitHub is used for team collaboration, pull requests, and project tracking.

**3. Explain the purpose of git init, git clone, and git pull.**

* git init – Initializes a new Git repository in a local directory.
* git clone <repo-url> – Creates a copy of a remote repository locally.
* git pull – Fetches and merges changes from a remote repository into the local branch.

💡 **Application:**

* Used when starting a new project (git init).
* To get a copy of an existing repository (git clone).
* To sync the latest changes from the team (git pull).

**4. How does branching work in Git?**

Branching allows developers to work on different features or bug fixes without affecting the main codebase.

* Create a branch: git branch feature-branch
* Switch to the branch: git checkout feature-branch
* Merge it back: git merge feature-branch

💡 **Application:**

* Developers create separate branches for features, test them, and merge them into the main branch after validation.

**5. What is the difference between git merge and git rebase?**

* **git merge** combines branches and creates a merge commit.
* **git rebase** applies changes from one branch onto another in a linear history.

💡 **Application:**

* Use merge when combining long-term branches.
* Use rebase for a cleaner commit history before pushing to production.

**6. What are git stash and git stash pop?**

* git stash temporarily saves changes without committing.
* git stash pop restores the stashed changes and removes them from the stash list.

💡 **Application:**

* Useful when switching branches without committing incomplete work.

**7. How do you undo a commit in Git?**

* Undo last commit but keep changes: git reset --soft HEAD~1
* Undo last commit and remove changes: git reset --hard HEAD~1
* Create a new commit to undo: git revert <commit-id>

💡 **Application:**

* Used when a commit contains incorrect changes and needs to be fixed.

**8. How do you resolve merge conflicts in Git?**

1. Run git status to see conflicting files.
2. Open the file and manually resolve the conflicts.
3. Stage the resolved file: git add <file>
4. Commit the changes: git commit -m "Resolved merge conflict"

💡 **Application:**

* Merge conflicts occur when two people edit the same file simultaneously.
* Resolving conflicts ensures all changes are integrated properly.

**9. What is a Pull Request in GitHub?**

A **Pull Request (PR)** is a request to merge code from one branch into another.

💡 **Application:**

* Used in open-source contributions and team collaborations to review code before merging.

**10. How do you check the commit history in Git?**

* git log – Shows the full commit history.
* git log --oneline – Shows a compact version of commit history.

💡 **Application:**

* Helps track who made changes and when, aiding in debugging and rollback.