***UNIT 1***

***Que DevOps?***

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



## How DevOps Works

Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team’s velocity.

## Benefits of DevOps

### Speed

Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, [microservices](https://aws.amazon.com/devops/what-is-devops/" \l "microservices) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) let teams take ownership of services and then release updates to them quicker.

### Rapid Delivery

Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers’ needs and build competitive advantage. [Continuous integration](https://aws.amazon.com/devops/continuous-integration/) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) are practices that automate the software release process, from build to deploy.

### Reliability

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like [continuous integration](https://aws.amazon.com/devops/continuous-integration/) and [continuous delivery](https://aws.amazon.com/devops/continuous-delivery/) to test that each change is functional and safe. [Monitoring and logging](https://aws.amazon.com/devops/what-is-devops/#monitoring) practices help you stay informed of performance in real-time.

### Scale

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, [infrastructure as code](https://aws.amazon.com/devops/what-is-devops/#iac) helps you manage your development, testing, and production environments in a repeatable and more efficient manner.

### Improved Collaboration

Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams [collaborate](https://aws.amazon.com/devops/what-is-devops/#communication) closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).

### Security

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and [policy as code](https://aws.amazon.com/devops/what-is-devops/#policyascode), you can define and then track compliance at scale.

**Que The basic of testing in devops?**

**Ans** DevOps is all about streamlining and automating your software delivery life cycle. When you enable DevOps in your organization, all the development and operation activity needs to be automated.

DevOps Testing is the continuous and automated process of software testing that enables continuous and faster delivery of software. The legacy approach of testing is performed manually. Manual testing involves more human activity, is more prone to error, and is more time-consuming. As DevOps focuses on automating processes, testing can fit into DevOps with the help of the right test automation tools.

**Features of testing in a DevOps environment**

Organizations are swiftly moving to DevOps for Agile teams. Both approaches focus on the automation of testing activity.

Let’s understand the features of DevOps testing

Testing is automated, and it is continuous.

Testing is carried out at different stages of SDLC.

Easy to roll back and detect errors in the code as results/reports will be instant.

Testing becomes a shared responsibility. Each team member is responsible for quality.

**Who is Involved in DevOps Testing?**

In DevOps Teams, everyone is equally responsible for the quality of the product. That means testing is done by the whole team. So no more blame game on the testing team. Designated testing team member will have expertise and skill set in the DevOps testing and he will drive the testing. Testing team member helps to choose the right tool, build the automation framework, automation code review, integrate testing activity with pipelines, etc. On the other hand, all team members will contribute to developing the automation scripts.

In DevOps testing, the tester needs to have a broad understanding of development, testing, and tools.

Some of the Key areas that DevOps QA members should focus on are given below.

Source Control (using Git, Bitbucket, AzureDevOps, etc)

Continuous Integration (using Jenkins, Bamboo, AzureDevOps)

Deployment Automation & Orchestration (using Jenkins, AzureDevOps, Octopus Deploy)

Container Concepts (Kubernetes, Docker)

Cloud (using AWS, Azure, GoogleCloud, etc)

Apart from above knowledge, tester may also need to write code in specific languages such as Java, Javascript, Python, C#, etc.

**Testing Types in DevOps**

DevOps Testing focuses on testing during every stage of the Software Development Lifecycle. It emphasizes automation testing. All different testing types are automated and hooked into the DevOps pipeline to achieve Continuous Integration(CI) and Continous Testing(CT)

Let’s take a look at different types of DevOps testing.

**Unit Testing**

Unit Testing focuses on small objects. It is also called white box testing. It covers all different conditions and methods available at the class level. Mostly it is written in the unit testing framework such as JUnit, NUnit, Jest, Mocha, etc.

**Component Test**

Larger and more complex applications are segregated into multiple components as it makes development easier and faster. Each component might contain different classes and objects to deliver small functionality. These functionalities are tested using an automation framework while building the application.

**Integration Test**

As the application is built with components, two or more components are integrated into the product. These Integrated components are tested using the DevOps testing tools. It may or may not involve database calls.

**API Testing**

Microservice architecture is the most popular and mostly followed in the organization. In API testing, individual or interrelated microservices are tested together. Mostly API contracts, static data, and data types are tested using API automation tools.

**Functional Testing**

Functional Testing focuses on user workflows or end-user functionalities. Often called as end to end Testing. As it touches all the different components, databases, APIs, etc. This is the most critical type of testing. DevOps testing emphasizes incorporating automation tools for end-to-end testing as manual testing is the most time-consuming. Fortunately, tools like Testsigma, Selenium, and Playwright do this job without any hassle.

Automate your tests for web, mobile, desktop applications and APIs, 5x faster, with Testsigma

**Performance/Stress Testing**

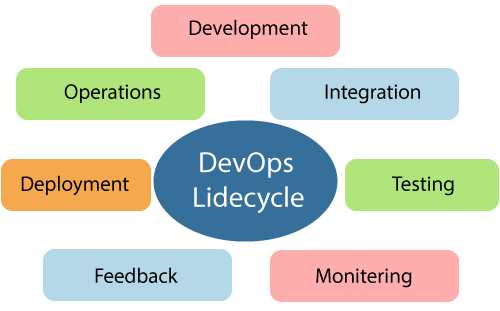
Performance testing is important as the user experience is closely associated with it. Performance testing measures the overall application behavior under different loads, different network conditions, and infrastructure configurations. Performance testing can be scripted and automated. Tools such as JMeter, Blazemeter, K6, etc can be used for stress testing.

DevOps Testing Tools

1. Mocha
2. Typemock
3. EMMA
4. Parasoft
5. SimpleTest
6. Apache JMeter
7. k6
8. Predator
9. Watir
10. TestComplete
11. TestProject
12. Selenium
13. Leapwork
14. Testsigma
15. Tosca
16. Appium
17. AppVerify
18. Bamboo
19. Docker
20. Jenkins
21. Opkey

# Ques- DevOps Lifecycle?

DevOps defines an agile relationship between operations and Development. It is a process that is practiced by the development team and operational engineers together from beginning to the final stage of the product.



Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:

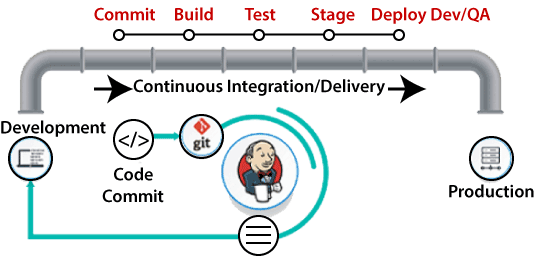
### 1) Continuous Development

This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

### 2) Continuous Integration

This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes **unit testing, integration testing, code review**, and **packaging**.

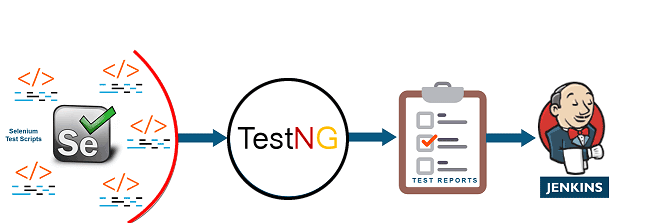
The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar. Then this build is forwarded to the test server or the production server.

### 3) Continuous Testing

This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as **TestNG, JUnit, Selenium**, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, **Docker** Containers can be used for simulating the test environment.



**Selenium** does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called **Jenkins**.

Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

### 4) Continuous Monitoring

Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application.

It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position. The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service.

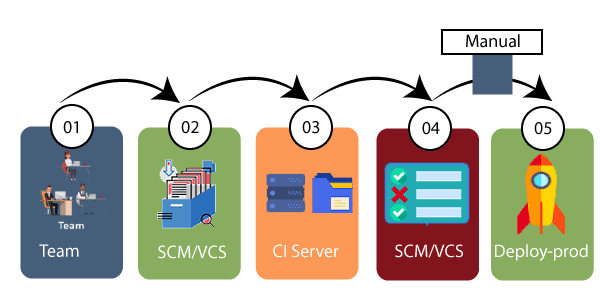
### 5) Continuous Feedback

The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.

### 6) Continuous Deployment

In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.



The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as **Chef, Puppet, Ansible**, and **SaltStack**.

Containerization tools are also playing an essential role in the deployment phase. **Vagrant** and **Docker** are popular tools that are used for this purpose. These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.

Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

### 7) Continuous Operations

All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.

It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

**Que- Integration of testing in DevOps?**

Software apps have to pass through five different stages if they are developed within a DevOps pipeline:

1. Continuous Development
2. Continuous Testing
3. Continuous Integration
4. Continuous Delivery
5. Continuous Monitoring

DevOps helps promote collaboration between development, operations, and QA. Since there are so many processes involved, you can’t define one tool as the [DevOps tool](https://intellipaat.com/blog/top-devops-tools/). There are various tools used throughout the various stages of the DevOps pipeline. The tools used for various stages will be different in different organizations.

Here we have listed out some common DevOps tools used in each stage:

### Continuous Development

Jira software is widely used for this stage in the DevOps pipeline. The software facilitates collaborations by visually highlighting issues throughout the workflow. The Jira platform is really easy to use for development planning. It also helps in tracking daily work and project progress reporting.

### Continuous Testing

A lot of organizations also use Jira to manage test cases. This is done to facilitate the working of both development and testing teams together. Zephyr for Jira is a tool that is used for creating, executing, tracking, and reporting test uses.

### Continuous Integration

A CI/CD server called Jenkins can automatically run tests every time a new code is pushed into the repository by the developer. CI can detect bugs early in the process. This helps make the bugs less complex and easy to resolve. Jenkins was originally made to be a build automation tool for Java. Nowadays, it has become a platform where you can build, automate and deploy any software project. It doesn’t matter which computer language, database, and/or control system was used to write the software.

### Continuous Delivery/Deployment

Tools like Puppet or Chef enable users to avoid ‘snowflake servers’ in their delivery/deployment environment. Snowflake servers are production servers that have been running for a long time and have been configured and modified repeatedly. This makes the server unique, like a snowflake and hence, the name. This server requires manual configuration that is beyond automated deployment scripts.

You can use Manifests in Puppet or Recipes in Chef, to explain the configuration of the elements of a server. This helps in configuring and specifying environments in a completely automatable format.

### ****Continuous Monitoring****

There are log monitoring tools like [Spunk](https://intellipaat.com/blog/what-is-splunk/) and Elf that help users analyze data on all the transactions that have taken place in their deployed IT applications. This helps ensure uniform performance, security, availability, and user experience. Both these tools are capable of providing a scalable way of collecting and indexing these log files. They also provide a search option for users so they can interact with data and create visualizations like dashboards, alerts, or reports.

**Que- DevOps Testing Strategy?**

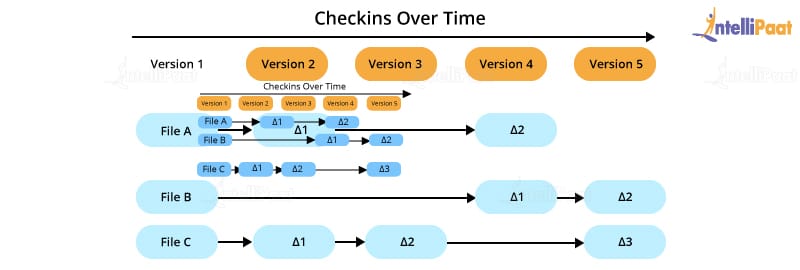
Speaking of the impact the DevOps culture is bringing about, you would by now know that the quality of the DevOps testing has a great impact on your organization’s success. Following are a few key tips to make the best use of the DevOps Testing Strategy for your organization’s success:

* You need to identify all the test cases that need to be executed for a particular build.
* The execution of all tests should be minimal yet powerful.
* The Development and QA teams need to identify all the areas that a particular build is affecting. They need to execute all the test cases related to the build.
* Coverage tools and specialized code analysis need to be configured to ensure that all of the code is covered.
* Don’t execute all regression test cases for a test pass.
* A new features testing strategy needs to be standardized while the QA creates test scripts and runs the automation tests on the builds in the interim. This process should continue until the code is stable enough and can be deployed in the Production environment.
* All the deployments need to be automated. The environments required for testing need to be formalized.
* Automation testing across different cross-platform environments should be run by QA with the help of automation techniques.
* Parallel execution of tests should be implemented to reduce the time-to-market.
* There should be an exit criterion set for each of the test cases. When the results get back to the chain, the Production can take a quicker decision.

**Que- Git?**

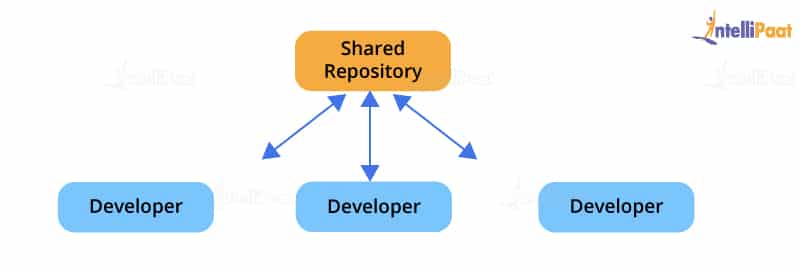
**Git Definition**

When you check for the definition of Git online, the best you can go something along these lines:Gitis a distributed version control system (DVCS) for tracking changes to files. But what does that mean? Git is an open-source VCS, which is not file-based, unlike other systems. Rather, it stores information as snapshots. Being a VCS, helps coders revert to their previous code when they hit a roadblock in the newer version, without affecting the original source code. On the other hand, what makes it different from other VCS is the way it sees data, which is more like a series of snapshots. It basically clicks a picture of how all your files look at the moment and saves the changes made to them over time.

******

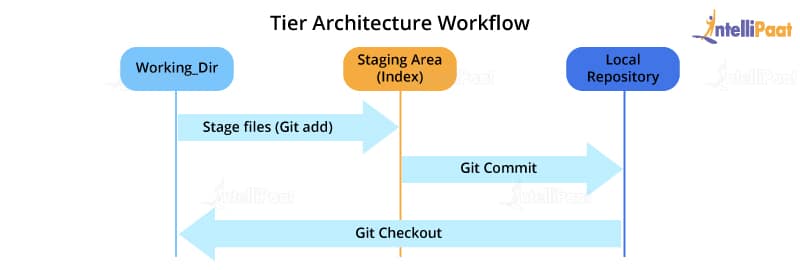
## ****Features of Git****

* **Works on a distributed system**: A distributed system is one that allows collaborators to access the central repository using a VCS, even from the remotest corner of the world. As Git maintains a snapshot every time a user pulls a file, the risk of data loss due to system failure or lack of Internet connection is mitigated. Users are allowed to work on the same bit of code simultaneously without getting interfered by others.



* **Compatible with all operating systems**: Git is compatible with almost all operating systems that are available today. Even the repositories created by other version control systems can be accessed by the Git repository.
* **Allows for non-linear development**: As users from remote parts of the world can access the Git repository, work on it, and update the project at any time they want, Git allows for development in a non-linear fashion. Git supports such a kind of development by providing its branching and merging features, and it uses specific tools for navigating through them. The projects are viewed in a tree form.
* **Branches like a tree**: While users are working on their projects, branches parallel to the main project file are created, so the original code is not affected. There is no restriction upon the number of branches created.
* **Light as a cotton ball**: One might think that making multiple copies of data from a central repository to a local one will eventually lead the system to crash due to overload. But, Git has got it covered. It compresses the data in such a way that it takes up minimal space, and whenever you need to retrieve data, the reverse technique is used. This helps save a lot of memory.
* **Fast as a flash**: Unlike other version control systems, Git is written in a language known to be the closest to the machine language, that is, C. Hence, it processes information much faster.
* **Reliable**: There will never be an issue of data loss as long as the copies of data in the central repositories are available in the local repositories of different collaborators.

## ****Git Architecture****



Most of the version control systems have a two-tier architecture. However, Git has a layer more, making it a three-tier architecture. But, why are there three layers of Git? Let’s find out.

**The three layers are:**

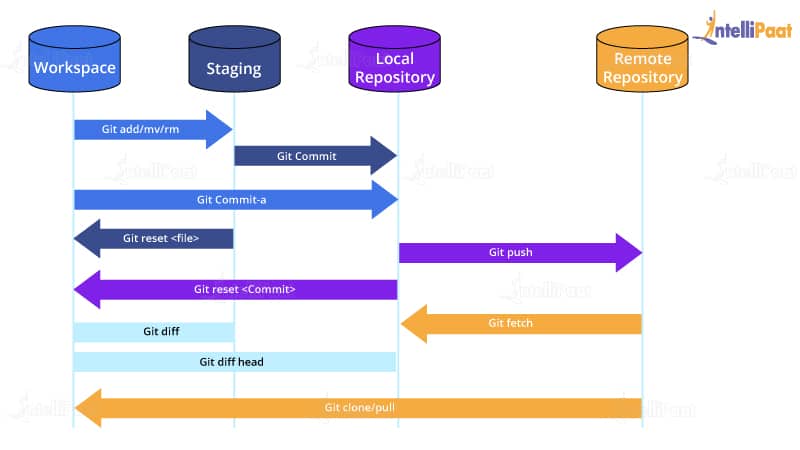
* **Working directory**: This is created when a Git project is initialized onto your local machine and allows you to edit the source code copied.
* **Staging area**: Post the edits, the code is staged in the staging area by applying the command, **git add**. This displays a preview for the next stage. In case further modifications are made in the working directory, the snapshots for these two layers will be different. However, these can be synced by using the same ‘git add’ command.
* **Local repository**: If no further edits are required to be done, then you can go ahead and apply the **git commit** command. This replicates the latest snapshots in all three stages, making them in sync with each other.

## ****How to use Git?****

Much of Git’s popularity is attributed to its user-friendly nature. You can check out these simple concepts to perform certain basic operations and get acquainted with [Git basics](https://intellipaat.com/blog/tutorial/devops-tutorial/git-tutorial/" \t "_blank).

* **Commit**: This is an object that takes the current state of the repository.
* **Pull**: This operation copies the changes made to a project to the local repository from the remote one.
* **Push**: This operation copies the changes made to a project to the remote repository from the local one.

Now, let’s take a look at some of the basic and commonly used commands in Git.



## ****Git Commands****

Based on what you work with, be it remote or local repositories, the Git commands change. Let’s take a look at the various [commands in Git](https://intellipaat.com/blog/tutorial/devops-tutorial/git-commands/).

### ****Git Commands When Working with Local Repositories****

* **git init:** This Git command converts a directory into an empty repository. This is the initial step you need to take to build a repository. Once you run git init, you will be able to add and commit files and directories.
* **git add:** This command allows you to add files in the Git staging area. The file must be added to the index of Git before being available to commit to any particular repository. You can use this command to add directories, files, etc.
* **git commit:** The commit command in Git allows you to track the changes in the files in a local repository. Each commit has its own unique ID for reference.
* **git status:** The git status command returns the present state of a repository, like if the file is in the staging area but has not been committed.
* **git config:** There are numerous configurations and settings possible in Git, and this command allows you to assign these settings. User.name and user.email are the two significant settings that set the name and email address of a user.
* **git branch:** This command determines the branch of the local repository and allows you to add or delete a branch.
* **git checkout:** You can use this command to switch to another branch.
* **git merge:** The merge command allows you to integrate two or more branches together. It combines the changes made in the branches.

### ****Git Commands When Working with Remote Repositories****

1. **git remote:** This Git command allows you to connect a remote repository to a local repository.
2. **git clone:** You can use the clone command to create a local copy of an already existing remote repository. This allows you to copy and download the required repository to the system. It is similar to the init command while working with remote repositories as it allows you to build a local directory, consisting of all the necessary files and history of the repository.
3. **git pull:** The pull command is used to run the latest version of any repository. This pulls all the changes made from the remote to the local repository.
4. **git push:** This command sends local commits to the respective remote repository. It needs two parameters, i.e., the remote repository and the specific branch where it needs to be pushed.

There are numerous other Git commands that are of more advanced level, such as git stash, git log, git rm, etc.

## ****Role of Git in DevOps****

Git plays a vital role in DevOps. [DevOps](https://intellipaat.com/blog/what-is-devops/" \t "_blank) is responsible for the development and operation processes of software. It allows non-linear workflows in a distributed manner by providing data assurance to create quality software. This tool offers a space for developers to work. All the users are notified when any of the developers make changes in the file.

Git allows developers to collaborate and makes it easy for them to visualize the work, saving a lot of time and effort. Further, Git helps in code management that can be pushed to the shared repository, making the code ready to be integrated, tested, and deployed. It eases the communications between the development department and the operations department, making it the best suitable tool in DevOps.

## ****Advantages of Git****

There are numerous advantages of Git, some of which are listed further in this ‘What is Git?’ blog.

### ****Performance****

Git is much stronger and reliable than other VCSs. Professionals can easily commit changes, compare version branches, and merge them. Besides, it allows for the better optimization of code for improved performance. The algorithms that are used to develop Git take complete advantage of the stored knowledge of attributes that help in developing source code files, types of file access patterns that help in recalling code files whenever required, and so on.

### ****Security****

Git is mainly built to secure the source code and maintain its integrity. All the data in the file, the relationship between the directories and files, commits, tags, versions, etc. are guarded through cryptographic algorithms like SHA-1. This algorithm secures the code and changes history from accidental and malicious attacks. Git ensures that the files, data, and the change history of the source code are secure.

### ****Flexibility****

Git is designed to be flexible, which allows it to support various types of non-linear workflows. It is also one of the most efficient tools in terms of dealing with small-scale and large-scale protocols and projects. Besides, its unique design makes it possible to support branching and tagging various business operations, storing all the user activities as an important part of the change history. Git is one of the very few VCSs that have this feature.

### ****Wide Acceptance****

Most developers need the security, performance, flexibility, and functionality that Git offers to build their projects. It is the most widely accepted VCS comparatively, and it provides universally accepted performance standards and usability.

## ****Disadvantages of Git****

Even though Git seems like the perfect solution for various problems related to the software development life cycle (SDLC), it has its shortcomings.

* Its steep learning curve itself is a disadvantage, which is due to the non-intuitive nature of its commands. You need a much deeper understanding of the internals of Git to avoid inconsistencies while executing.
* Binary files are not Git’s forte. Whenever files containing non-text content are updated or used frequently, Git gets dragged and slowed down.
* It does create a conducive environment for a large number of developers distributed over a large geographical area but, at the same time, creates a lot of conflict and confusion while merging codes. To solve this problem, developers can establish an open communication channel before merging their bits of changes with the project.

Git is being used by large companies to achieve their benchmark goals. As products are developed all around the world, it is increasingly becoming the platform of integration. The introduction toGithas contributed to the building of a large community by following the mantra of developing ecosystems for SDLC. With a long way ahead for more technological advancements, Git has definitely found its place at the top. Hopefully, this ‘What is Git?’ blog helped you understand the basics of Git and inspired you to start the coding journey with it.

## Que What is Version Control System?

Version control systems or VCSs are a collection of various software tools that allow professionals to record the changes that have been made to the respective documents and files by tracking the modifications made to the program code. One of the main reasons why VCS is necessary is, often, software programs are developed by a group of developers who may be working from various parts of the globe. Since all of them make some contributions to the software code and keep making changes, it is important that the changes be communicated to other team members. This improves the management and efficiency while developing the software.

### ****Types of Version Control Systems****

Version control systems can largely be classified into the following three types:

* **Local version control systems:** Compared to the other types of version control systems, the local VCS is the simplest one that has a database consisting of the record of all the changes done to files while using revision control. Revision control systems (RCS) are among the most popular VCS tools that store patch sets on the disk. They can easily re-create the exact look of a file at any point in time by adding the patches.
* **Centralized version control systems:** They comprise a single repository, and all users get their own copies. Your changes will be reflected in the repository, i.e., when you commit, the team members can see the changes by updating. In simple terms, the other members can only see the changes you make if you commit, and they update. This VCS makes it easy for developers and programmers to collaborate and gain insights into what the rest of the team members are doing on a given project. Moreover, it enables administrators to know who can do which parts of the project.
* **Distributed version control systems:** These systems consist of several repositories, and all the users have their own repositories. The distributed version control systems require professionals to complete four steps to make their changes visible to other members of the team, unlike the centralized version control systems that required only two. Here, the other members will not be able to access your changes just by you committing your changes. When you commit, the changes will be reflected in your local repository only. Further, you must push the changes so that they can be visible in the central repository. Now, the other members cannot view the changes just by updating. First, they need to pull those changes into their respective repositories and then update. Git is the most common and popular distributed version control systems.

## Que What is a distributed version control system?

A distributed [version control system](https://about.gitlab.com/topics/version-control/) (DVCS) brings a local copy of the complete repository to every team member’s computer, so they can commit, branch, and merge locally. The server doesn’t have to store a physical file for each branch — it just needs the differences between each commit.

Distributed source code management systems, such as Git, Mercurial, and Bazaar, mirror the repository and its entire history as a local copy on individual hard drives.

[Distributed version control systems](https://about.gitlab.com/blog/2020/10/02/distributed-version-control/) help software development teams create strong workflows and hierarchies, with each developer pushing code changes to their own repository and maintainers setting a [code review process](https://about.gitlab.com/topics/version-control/what-is-code-review/) to ensure only quality code merges into the main repository.

A DVCS can be puzzling, especially if a team member is accustomed to [centralized source code systems](https://about.gitlab.com/topics/version-control/what-is-centralized-version-control-system/), because a contributor can no longer rely on a server to resolve conflicts when merging and has to resolve them locally, which can result in confusing merge commits. However, despite the initial discomfort, a distributed source control system can ensure stable code development when multiple developers contribute to software development projects.

## What are the advantages of using a distributed version control system?

### Reliable backup copies

An interesting way to think about distribution version control is to visualize a collection of backups. When a team member clones a repository, she essentially creates an offsite backup, so if something catastrophic happens, like a server crash, every team member’s local copy becomes a backup. Unlike a centralized version control system, a distributed version control removes the reliance on a single backup, making development more reliable. A common misconception is that multiple copies could be a waste of space, but most development includes plain text files and many systems compress files, so the impact on hard drive storage is minimal.

### Fast merging and flexible branching

Because systems don’t require remote server communication, code can be quickly merged. A distributed version control also allows software development teams to use different branching strategies, a feature that isn’t possible with a centralized system. Distributed version control systems accelerate delivery and business value by helping team members focus on innovation rather than become bogged down with slow builds.

### Rapid feedback and fewer merge conflicts

A DVCS makes branching easy, because having an entire repository’s history on their local workstation ensures that they can quickly experiment and [request a code review](https://about.gitlab.com/blog/2020/06/08/better-code-reviews/). Developers benefit from fast feedback loops and can share changes with team members before merging the changeset. Merge conflicts are less likely, because contributors focus on their own piece of code. Furthermore, having easy access to the full local history helps developers identify bugs, track changes, and revert to previous versions.

### Flexibility to work offline

A distributed version control system doesn’t require an internet connection, so most development, except pushing and pulling, can be done while traveling or away from home or an office. Contributors can view the running history on their hard drive, so any changes will be made in their own repository. This increased flexibility enables team members to fix bugs as a single changeset. Increased developer productivity

With a local copy, developers can complete common development activities rapidly. A DVCS means that developers no longer have to wait on a server run through routine tasks, which can slow down delivery and cause frustration.

### Git: An example of a distributed version control system

Git is a distributed version control system known for its speed, workflow compatibility, and open source foundation. With Git, software teams can experiment without fearing that they’ll create lasting damage to the source code. Teams using a Git repository can tackle projects of any size with [efficiency and speed](https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control).

**Que Why is continuous testing important for DevOps?**

With continuous testing, code is automatically tested as soon as it is integrated. This directly supports DevOps and the goal of delivering high-quality software, faster. Additionally, CT helps save developer time and effort because they no longer have to wait for QA teams to finish testing before fixing their code.

**“Continuous Testing refers to the execution of automated tests that are carried out at regular intervals every time code changes are made. These tests are conducted as a part of the software delivery pipeline to drive faster feedback on recent changes pushed to the code repository.”**

Continuous Testing was introduced initially with the intention of reducing the time taken to provide feedback to developers. It attempts to address the following question –

***How fast can we identify that there’s a problem, inform the development team, and get it fixed?***

The primary goal here is to test more often, particularly at an individual level in the early stages of development, and then testing the unified codebase as a whole.

### What are the benefits of Continuous Testing in DevOps?

Listed below are some key benefits of incorporating Continuous Testing:

* Early discovery of critical bugs
* Seamless collaboration among developers, QA and Operations team
* Helps to assess the quality of software developed at each stage
* Can be seamlessly incorporated into DevOps
* Helps drive faster test results which leads to improved code quality
* Repeated testing ensures minimal failure rate for new releases
* Faster time to market with a viable product and continuous feedback mechanism

**Que - Difference between continuous testing and traditional testing.**

Continuous Testing differs from traditional testing in that it involves ongoing and automated testing practices that are seamlessly integrated into the DevOps pipeline.

Unlike traditional testing, which often occurs as a separate phase after development, Continuous Testing ensures that tests are executed frequently, often with every code change, providing rapid feedback to developers and reducing the likelihood of critical issues reaching production.

Here’s a comparison of Continuous Testing and traditional testing:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Continuous Testing** | **Traditional Testing** |
| **Timing and Frequency** | Ongoing, frequent, with each code change | Occurs as a separate phase after development |
| **Automation** | Relies heavily on test automation | May involve automation but is not as prevalent |
| **Feedback Loop** | Provides rapid feedback to developers | Feedback often occurs later in the lifecycle |
| **Integration with DevOps** | Integral part of the DevOps pipeline | May not align well with DevOps |
| **Speed and Efficiency** | Designed for speed and efficiency | Can be time-consuming, especially when manual |
| **Risk Mitigation** | Identifies and addresses issues early | May not catch issues until later in the cycle |
| **Culture and Collaboration** | Promotes collaboration and shared responsibility | May promote a more siloed approach |

Building the code , need for building the code??