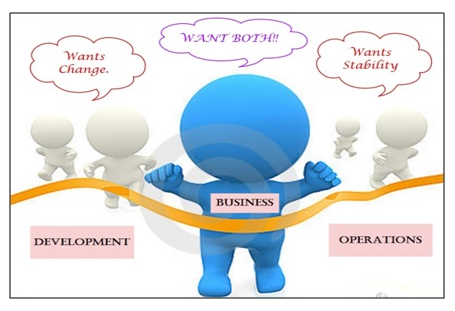
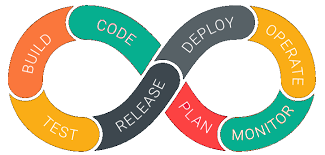
Why Devops?

* Generally development and IT support groups do not talk to each other and act like absolute silos.
* Dev focuses on producing new systems and applications and ensures that the users get to use it as fast as possible whereas Operations looks from a complete different aspect as their focus is to ensure the users get to use a fast and bug free stable system.
* Both Development and Operations strive towards the goal of making a customer happy and satisfied with their delivered systems and services. Yet their approaches in getting to this goal are contradictory to each other.



DevOps is basically a combination of two words, "development" and "operations." Usually, DevOps is a culture that follows the set of practices to combine the DevOps and  IT operation teams. Its major goal is to shorten the system delivery life cycles. This culture increases the organization's speed to deliver the applications and services.

DevOps Lifecycle:



**1. Continuous Development**

The first phase of the DevOps lifecycle is where the planning and software coding takes place. The planning involves understanding the vision of the project and envisioning a software based on those perceptions.

Planning doesn’t involve any major tools, but maintaining the code entails the use of a range of tools. Developing the source code for application begins by choosing from the different programming languages. JavaScript, C/C++, Ruby, and Python are prominently used for coding applications in DevOps.

The process of maintaining the code is called Source Code Management (SCM), where version control tools such as **GIT, TFS, GitLab, Subversion, and Mercurial**, among others, are used.

In the SCM process, **GIT** is a preferred tool that enables a distributed version control. It facilitates data assurance through circulated non-linear workflows. For large projects, where a vast number of collaborators are involved in the development activity, GIT establishes reliable communication between the teams through the Commit messages.

With the help of a version control tool, a stable version of the application code is built in the continuous development phase. Developers can also package the code into .exe (executable) files by using Garden, **Maven**, and similar tools.

**2. Continuous Integration**

The source code gets modified several times, and these frequent changes happen on a weekly or a daily basis. Code integration, the next phase, is the core of the entire DevOps lifecycle. In continuous integration, new codes that support add-on functionalities are built and integrated into the existing code.

In this phase, bugs in the source code are detected early on. To generate new code that brings more functionalities to the application, developers run tools for unit testing, code review, integration testing, compilation, and packaging.

The continuous integration of this new code into the existing source code helps reflect the changes that end-users would experience with the updated code.

**Jenkins** is popularly used as a reliable DevOps tool for procuring the updated source code and constructing the build into.exe format. These transitions occur seamlessly, and the updated code is packaged and proceeded to the next phase, which is either the production server or the testing server.

**3. Continuous Testing**

Some developers carry out the continuous testing phase prior to the continuous integration phase. Based on the updations in the application code, this phase can be repositioned around the continuous integration phase in the DevOps lifecycle.

Here, the developed software is continuously tested for bugs. A test environment is simulated with the use of Docker containers. Through automated testing, developers save effort and time, usually lost in manual testing. Reports generated by automated testing improve the test evaluation process. Analyzing the failed test-cases becomes easy. After going through a UAT (User Acceptance Testing) process, the resultant test-suite is simpler and bug-free. **TestNG, Selenium and JUnit** are some of the DevOps tools used for automated testing. These tools can also arrange test-case execution in a pre-set timeline.

Quality assessors (QAs) can use these tools for parallel testing of several other code-bases.

It ensures the flawless functionality and inter-networking of the application. In the end, the tested code is re-sent to the continuous integration phase for updating the source code.

**4. Continuous Feedback**

Continuous testing and continuous integration are the two crucial phases that ensure consistent improvements in the application code. Continuous feedback is a peculiar phase where these improvements are analyzed.

Developers can gauge the outcome of these modifications on the final product. Most importantly, customers who tested these applications can share their experiences in this phase. In a majority of cases, this phase of the DevOps lifecycle provides a turning point to the application development process. The feedback is assessed promptly and developers begin working on the new changes. Sooner, there is a positive response in customer feedback, which paves the way for releasing new versions of the software application.

**5. Continuous Monitoring**

Monitoring the performance of an application is of key importance for [application developers](https://www.cuelogic.com/blog/should-you-hire-a-fullstack-developer-or-a-devops). In this phase, developers record data on the use of application and continuously monitor each functionality. “Server not reachable” or “low memory” are some of the common system errors resolved in this phase.

Continuous monitoring helps in sustaining the availability of services in the application. It also determines the threats and root causes of recurring system errors. Security issues get resolved and problems are automatically detected and fixed.

Compared to the software development teams, the IT operations teams are more involved in this phase. Their role is pivotal in supervising user activity, checking the system for unusual behavior, and tracing the presence of bugs.

Sensu, ELK Stack, NewRelic, Splunk and Nagios are the key DevOps tools used in continuous monitoring. These tools enable complete control in overseeing the performance of the system, the production server, and the application. The operations team can actively engage in increasing the reliability and productivity of the applications with the help of these tools.

When major issues are detected in this phase, the application is swiftly rerun through all the earlier phases of the DevOps lifecycle. That is how finding a resolution to all sorts of issues becomes faster in this phase.

**6. Continuous Deployment**

Conventionally, the phase of continuous deployment takes place before continuous monitoring. But, developers make sure that this phase is always active in the DevOps lifecycle, especially after the application goes live and starts receiving high traffic.

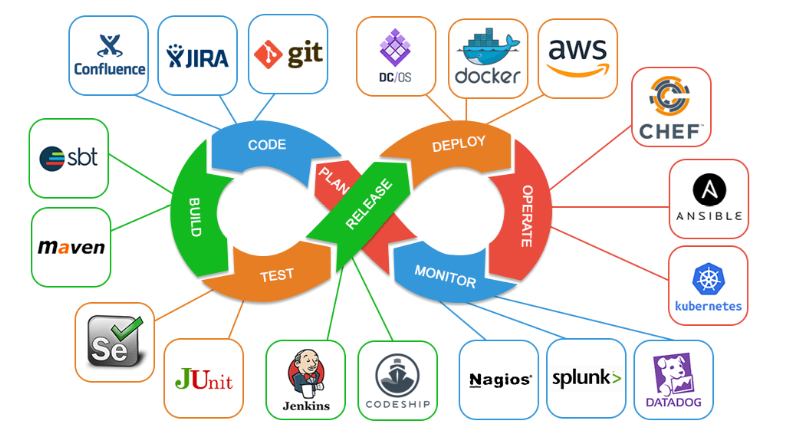
In this phase, the finalized application code is deployed to the production servers. Configuration Management is a key process in this phase, and it carries out the precise deployment of application code on all servers. Consistency in the application’s performance and functional conditions is established and curated. Code is released to the servers, updates are scheduled for all servers, and these configurations are kept consistent throughout the production process. Ansible, Puppet, and Chef are some of the effective [DevOps tools](https://www.cuelogic.com/blog/devops-with-puppet-chef-and-ansible) used for Configuration Management, where they frequently execute the quick and continuous deployment of new code.

Containerization tools are used to achieve continuous deployment through the Configuration Management process. Vagrant, a containerization tool, develops coherence in different environments - from development and testing to staging and production. Similarly, the scalability of continuous deployment is handled by tools like Docker. These tools nullify all sorts of production failures and system errors by replicating and packaging the software couplings from testing, staging, and development phases. Ultimately, the application runs smoothly on different computers.

**7. Continuous Operations**

The last phase of the DevOps lifecycle is the shortest phase and the least complicated one. The purpose of continuous operation is to automate the process of releasing the application and the subsequent updates. Development cycles in continuous operations are shorter, allowing developers to ongoingly accelerate the time-to-market for the application.

DevOps tools



## DevOps Practices

The following are DevOps best practices:

Continuous Integration

Continuous Delivery

Microservices

Infrastructure as Code

Monitoring and Logging

Communication and Collaboration

<https://aws.amazon.com/devops/what-is-devops/>

GIT

Version Control:

Version Control is the management of changes to documents, computer programs, large websites and other collection of information.

Version Control System: is a application that helps maintaining of the work done previously and current.

Advantages:

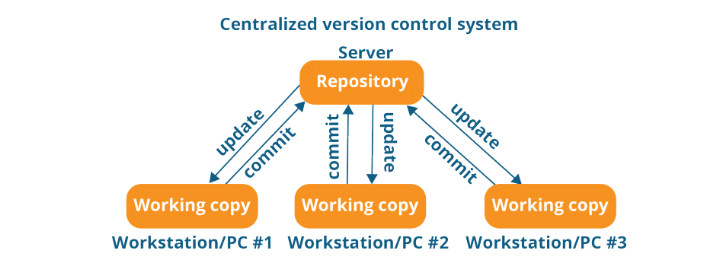
* Developers can work simultaneously.
* History of work is maintained.
* No repetitive work.

There are two types of VCS:

• Centralized Version Control System (CVCS)

• Distributed Version Control System (DVCS)

Centralized VCS:

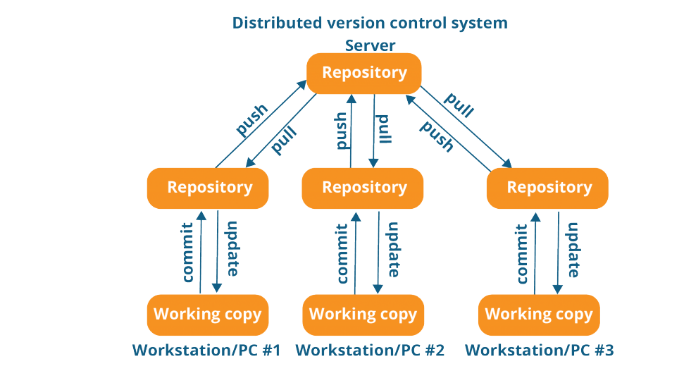


Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

• It is not locally available; meaning you always need to be connected to a network to perform any action.

• Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

Distributed VCS



The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

• All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.

• Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.

• If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor’s local repositories.

What Is Git?

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.

Features Of Git:

• Free and Open Source

• Speed

• Scalable

• Reliable

• Secure(SHA1-Secure hash function)

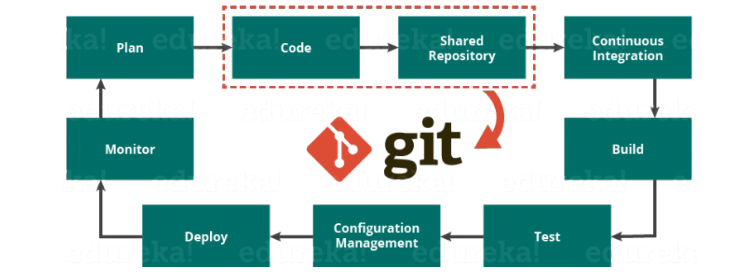
• Supports non-linear development:

• Easy Branching

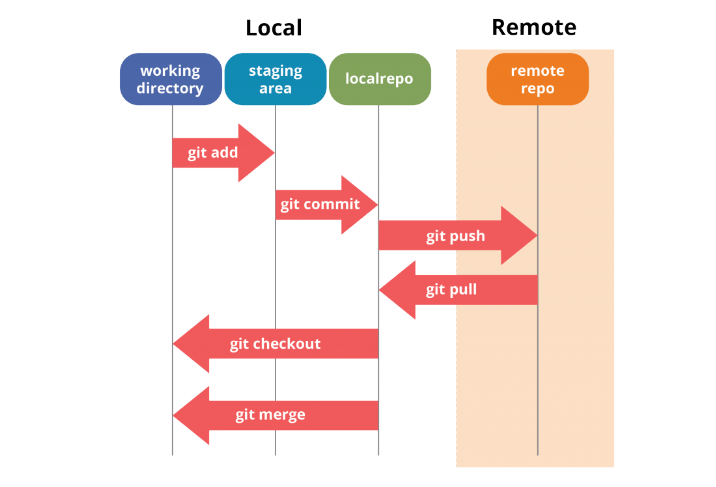
• Distributed Development

• Compatibility with existing systems or protocol

Role of Git In DevOps:



Git plays a vital role when it comes to managing the code that the collaborators contribute to the shared repository. This code is then extracted for performing continuous integration to create a build and test it on the test server and eventually deploy it on the production.



Git Commands:

git config

This sets the configuration values for your username, email, gpg key, preferred diff algorithm, file formats and more:

git config –global user.name “Your Username Here”

git config –global user.email “user@domain.com”

git init

This initializes a git repository, and creates the initial .git directory in a new or already existing project:

git init

Initialized empty Git repository in /home/username/GIT/.git/

git clone

This command creates a Git repository copy from a remote source. The command will also add the original location as a remote location so you are able to fetch from it again and push to it if you have permissions:

git clone git@github.com:user/test.git

git add

This will add file changes that are in your working directory to your index:

git add

git rm

This will remove files from your index and your working directory so they will not be tracked:

git rm filename

git commit

This Git command takes all of the changes written in the index, creates a new commit object pointing to it, and sets the branch to point to that new commit:

git commit -m ‘committing added changes’

git commit -a -m ‘committing all changes, equals to git add and git commit’



git status

This Git command shows the status of files in the index versus the working directory. It will list out files that are untracked (only in your working directory), modified (tracked but not yet updated in your index), and staged (added to your index and ready for committing):

git status

# On branch master #

# Initial commit #

# Untracked files: #

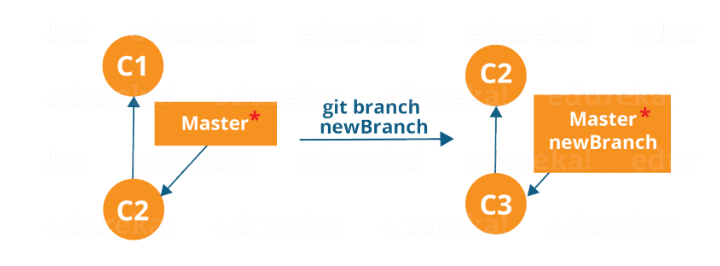
# (use “git add <file>…” to include in what will be committed) #

README

git branch

This lists existing branches, including remote branches if ‘-a’ is provided. It will create a new branch if a branch name is provided:

git branch -a \* master remotes/origin/master



git merge

This will merge one or more branches into your current branch. It also automatically creates a new commit if there are no conflicts:

git merge newbranchversion

git reset

This command will reset your index and working directory to the state of your last commit. Effectively taking you back:

git reset –hard HEAD

git tag

This Git command tags a specific commit with a simple, human readable handle that never moves:

git tag -a v2.0 -m ‘this is version 2.0 tag’

git pull

This will fetch all the files from the remote repository and merge them with your local one:

git pull origin

git push

This Git command will push all the modified local objects to the remote repository and advances its branches:

git push origin master

git remote

This shows all the remote versions of your repository:

git remote origin

## Branching

Branches in Git are nothing but pointers to a specific commit. Git generally prefers to keep its branches as lightweight as possible.

There are basically two types of branches viz. *local branches* and *remote tracking branches*.

A local branch is just another path of your working tree. On the other hand, remote tracking branches have special purposes. Some of them are:

* They link your work from the local repository to the work on central repository.
* They automatically detect which remote branches to get changes from, when you use git pull.

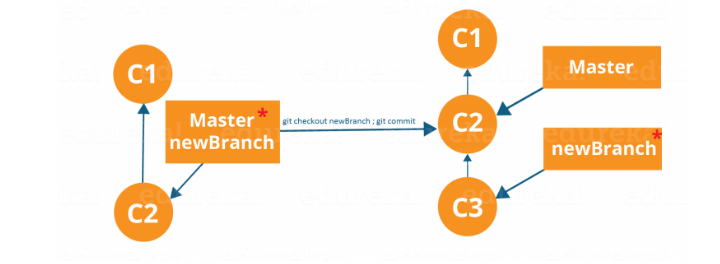
You can check what your current branch is by using the command:

git branch

The one mantra that you should always be chanting while branching is “branch early, and branch often”

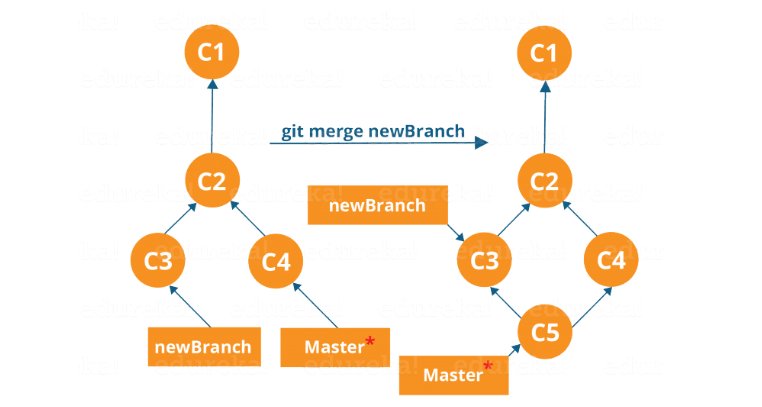
To create a new branch we use the following command:

git branch <branch-name>



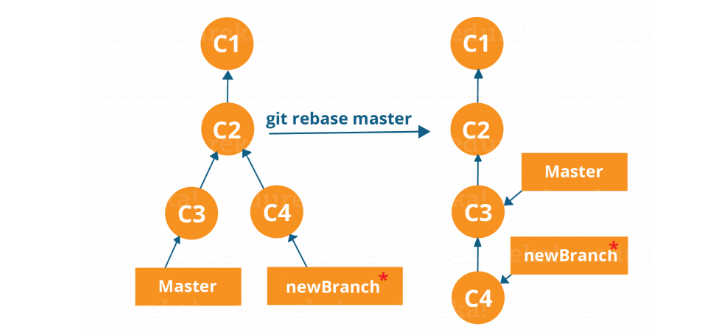
## **Merging**

Merging is the way to combine the work of different branches together. This will allow us to branch off, develop a new feature, and then combine it back in.



## **Merging**

Merging is the way to combine the work of different branches together. This will allow us to branch off, develop a new feature, and then combine it back in.



Maven

*Maven* is a powerful build tool for Java software projects.

**What is a Build Tool?**

A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:

* Generating source code (if auto-generated code is used in the project).
* Generating documentation from the source code.
* Compiling source code.
* Packaging compiled code into JAR files, WAR files or ZIP files.
* Installing the packaged code on a server, in a repository or somewhere else.

--Maven also provides the structure of the application based on your archetype selection.

-- It also helps us in downloading and managing the dependencies(jar files) of the application using pom.xml.

**Convention over Configuration**

Maven uses **Convention** over **Configuration**, which means developers are not required to create build process themselves.

Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.

From where it downloads the dependencies:

Maven repositories:

Remote repository: this repo is rarely used only when dependencies are not found in central repo. Like JBoss etc…

Central repository: Maven central : https://mvnrepository.com/

Local repository: All dependencies downloaded from central are stored in local repository. (c:\users\user-name\.m2\repository

POM: **Project Object Model**

Web.xml: is a web configuration file in which we add configuration(metadata) related to the web application. Like servlet description, database information, initial parameters etc….

The advantage of automating the build process is that you minimize the risk of humans making errors while building the software manually. Additionally, an automated build tool is typically faster than a human performing the same steps manually.

**Build Life Cycles, Phases and Goals**  
The build process in Maven is split up into build life cycles, phases and goals. A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals.

## Maven Build Life Cycles, Phases and Goals

When Maven builds a software project it follows a build life cycle. The build life cycle is divided into build phases, and the build phases are divided into build goals.

**Build Life Cycles**  
Maven has 3 built-in build life cycles. These are:

1. default
2. clean
3. site

The default life cycle handles everything related to compiling and packaging your project.

The clean life cycle handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc.

The site life cycle handles everything related to generating documentation for your project.

**Build Phases**  
Each build life cycle is divided into a sequence of build phases, and the build phases are again subdivided into goals. Thus, the total build process is a sequence of build life cycle(s), build phases and goals.

You can execute either a whole build life cycle like clean or site, a build phase like install which is part of the default build life cycle

The most commonly used build phases are:

Build Phase Description

**validate** Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded.

**Compile** Compiles the source code of the project.

**Test** Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

**Package** Packs the compiled code in its distributable format, such as a JAR.

**Install** Install the package into the local repository, for use as a dependency in other projects locally.

**Deploy** Copies the final package to the remote repository for sharing with other developers and projects.

**Maven Goal**

**Each phase is a sequence of goals, and each goal is responsible for a specific task.**

When we run a phase – all goals bound to this phase are executed in order.

Here are some of the phases and default goals bound to them:

* *compiler:compile* – the *compile* goal from the *compiler* plugin is bound to the *compile* phase
* *compiler:testCompile* is bound to the *test-compile* phase
* *surefire:test* is bound to *test* phase
* *install:install* is bound to *install* phase
* *jar:jar* and *war:war* is bound to *package* phase

## **Building a Maven Project**

To build a Maven project, we need to execute one of the life cycles by running one of their phases:

|  |  |
| --- | --- |
| 1 | mvn deploy |

This will execute the entire default lifecycle. Alternatively, we can stop at the install phase:

|  |  |
| --- | --- |
| 1 | mvn install |

But usually we'll use the command:

|  |  |
| --- | --- |
| 1 | mvn clean install |

To clean the project first – by running the clean lifecycle – before the new build.

We can also run only a specific goal of the plugin:

|  |  |
| --- | --- |
| 1 | mvn compiler:compile |

A **goal** represents a specific task which contributes to the building and managing of a project.

o install maven on windows, you need to perform following steps:

1. Download maven and extract it
2. Add JAVA\_HOME and MAVEN\_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

MAVEN\_HOME=

PATH= MAVEN\_HOME\bin

mvn --version