

MODULE 4

Understanding a Release Cycle

Facilitator Notes:

Introduce the module to the participants and tell them that you will be talking about the project release lifecycle, different stages of a release lifecycle, source code repositories, how to install and configure source code repositories and maven release goals – prepare, perform, clean and rollback.

You will be discussing about the project release lifecycle, different stages of a release lifecycle, source code repositories, how to install and configure source code repositories and maven release goals – prepare, perform, clean and rollback.

Module Objectives

At the end of this module, you will be able to:

- Define project release lifecycle
- Identify different stages of a release cycle
- State different source code repositories
- Know how to install and configure git
- Learn how to check in source code to git repository
- Explain maven prepare goal
- Describe maven perform goal
- Discuss maven clean goal
- Define maven rollback goal



Facilitator Notes:

Explain the module objectives to the participants.

Module Topics

Let us take a quick look at the topics that we will cover in this module:

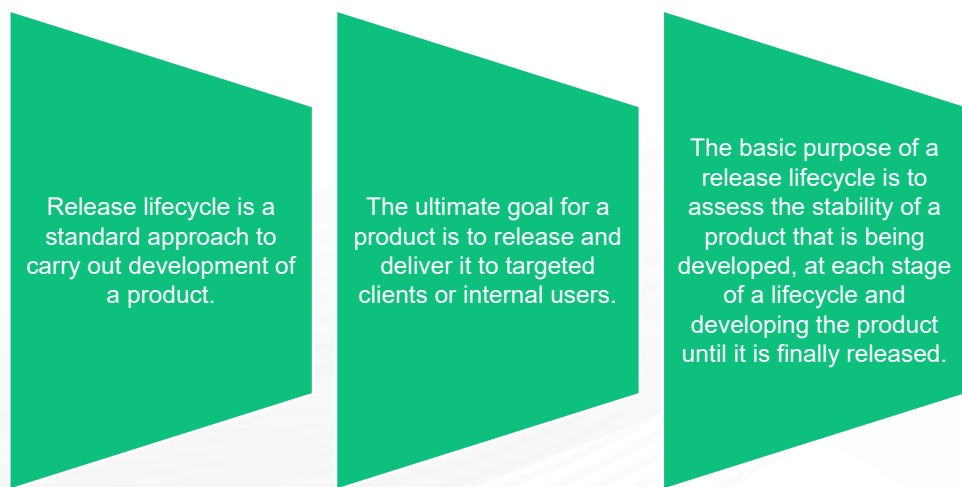
1. Project release lifecycle
2. Different stages of release cycle
3. Different types of source code repositories
4. Install and configure source code repository (Git)
5. Create organization, repository and branch in SCM
6. Check-in code to source repository
7. Maven prepare, perform, clean and release goals



Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

1. Project Release Lifecycle



Review copy only not to be circulated without prior permissions from Xebia

Facilitator Notes:

Discuss about the project release lifecycle.

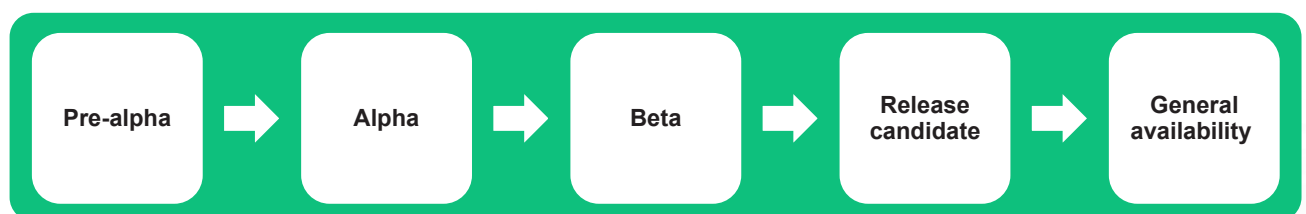
A **project release life cycle** includes different stages of development and maturity of the product: ranging from its initial development, testing and its final release, including updated versions of the **released** version to improve the product or fix the bugs.

Typical project release activities are as follows:

- Designing
- Planning
- Configuration Management
- Release and Deployment Management
- Testing Communication
- Rollout Planning

1.1 Different Stages of a Release Cycle

Generally, a product release life cycle consists of five stages, namely:



Facilitator Notes:

Discuss about different stages of a product release life cycle.

Release management makes an organization to think about the entire life cycle of the product release. Most of the companies do not have release policies or ineffective policies which is not an efficient way of releasing a product. Following the release management process has many benefits.

A business team or project management team visualize and define the software release life cycle, depending on their own approach. Generally, release lifecycle solves highest level process issues. Some of the process issues are as follows:

- **Release Unit Identification** : A 'release unit' describes the portion of a service or IT infrastructure that is normally released together according to the organization's release policy. The unit may vary, depending on the type(s) or item(s) of service asset or service component such as software and

hardware. The general aim is to decide the most appropriate release unit level for each service asset or component. An organization may, for example, decide that the release unit for business-critical applications is the complete application in order to ensure that testing is comprehensive. The same organization may decide that a more appropriate release unit for a website is at the page level.

- **Release Policies:** For designing release and release package, the policies given here needs to be considered:
 - The release and deployment teams need to understand the relevant architecture to be able to plan, package, build and test a release to support the new or changed service. This helps to prioritize the release and deployment activities and manage dependencies.
 - Dependent services will need to be built and tested in Service Transition, for example, an IT financial service may be dependent on several internal support services and an external service. There are normally dependencies between particular versions of service components required for the service to operate.
 - A release package may be a single release unit or a structured set of release units. Where possible, release packages should be designed so that some release units can be removed if they cause issues in testing
- **Releases or Bundled Changes:** Whenever possible, changes to an existing service should be bundled together and released on a regular (e.g., monthly) basis using the Release Management process. All implementation work on the release should be completed by the Planned End Date/Time.
- **Support and the End-of-Life Cycle:** A key part of the release management process definition should be a policy surrounding the end of life for your releases. Normally, a release reaches end of life because a newer release replaces it.

Pre-alpha

- Issued before the release of an alpha or beta release
- It is referred as a feature not complete
- Nightly builds are examples of Pre-Alpha builds

Alpha

- Released to QA team for testing
- Alpha build is internal to the organization

Beta

- First version released outside the organization
- Released for the purpose of evaluation or grey box testing

Facilitator Notes:

Discuss about different stages of a release cycle – Pre-Alpha, Alpha and Beta.

Pre-Alpha

There are many types of pre-alpha versions in the open source world. **Milestone** versions—It is released as soon as specific sets of functionality are completed. **Nightly builds** are typically checked out from control systems and build automatically overnight. It allows the QA team to test the newly implemented functionality immediately, and find the new bugs.

Alpha

The alpha build of the product delivered to the QA team, but generally internal to the community or organization that develops the product. In a rush to release the product quickly many companies are releasing alpha builds to external customers or value-chain partners. This enables more extensive usability testing in the alpha phase.

Beta

A **beta version** is the first version that is released to the community that developed software or outside the organization, to evaluate the product or perform real-world/grey box testing. Generally, Beta release will include all features and includes known issues and less priority bugs.

Beta testers are usually external customers or prospective customers of the organization that helps in developing the software. The company provides the product to them for free of cost or for a lesser price, they act as free testers.

Release candidate

- It refers to a potential final version of the product
- It is ready for release unless critical bugs are discovered
- The product is code complete in this stage

**General availability/
Final Release**

- It is the final version of a product
- It is a very stable release and bug-free
- This is often referred as golden release

Facilitator Notes:

Discuss about release cycle – Release candidate and General availability.

Release candidate

- It is ready for release unless critical bugs are found. In this phase, all the product features are completed and no showstopper bugs are identified. This phase of the product is referred as code complete.
- Microsoft Corporation generally uses the 'release candidate' term and in 1990s, Apple Inc. used 'golden master' name for its release candidates.

- A release candidate is called code complete when the development team agrees that no new source code will be added for this release, there may still be some changes to documentation and source code changes.

General availability/Final Release

- It is the final version of the product and typically identical to the final release, only last-minute bugs might be fixed. It is considered to be very stable and bug-free with quality a that is suitable to release to wider distribution.
- The term gold refers to the use of 'gold master disc' was commonly used to send the final version of the product to manufacturers, to create the mass-produced retail copies.

2. Source Code Repository

Source Code Repository

- A **source code repository** is a web hosting and file archive facility where a large amount of **source code**, is maintained, either publicly or privately.
- The source code repository maintains files and directories with revision history.
- Git is the widely used source code repository.
- The source code repository supports multi developer projects and open source projects, etc.

Facilitator Notes:

Discuss about the source code repository.

The source code repository has an inbuilt version control system. Version control systems help software teams to manage their changes to the source code over time. It keeps track of each and every modification to the code. If a developer makes any mistake, they can always roll back the code very easily.

Benefits of Source code repository:

1. Maintains long-term change history of a file. It stores the details like the creation or deletion of a file, along with its author and date details. This helps in performing root cause analysis of the bugs.
2. It efficiently manages branching and merging. Team members can work concurrently to work on independent features. There are multiple workflows available for branching and merging source code repository.
3. Traceability – It is possible to trace every change that is made to the source code and connect it to bug management software like jira to annotate every change with a message that describes the purpose of that check-in.

2.1 Different Types of Source Code Repositories

More popular source code repositories are as follows:

GitHub

It is established in 2008 and supports more than 24 million users and 69 million repositories which includes clones and forks.

BitBucket

It is established in 2008 by Atlassian which developed Jira and confluence. Bitbucket has 5 million users.

GitLab

Gitlab was developed in 2011. It is a web based repository manager with an issue tracker and wiki functionality.

Facilitator Notes:

Discuss about the different source code repositories.

GitHub: It provides a more user friendly and developer-focused environment. Git provides paid repositories to organizations which include private repositories. Git offers silver plans to educators and researchers.

BitBucket: BitBucket is known for its development products, JIRA and Confluence. It offers unlimited private repositories up to 5 users.

GitLab: GitLab is a web-based Git repository manager with the issue tracker wiki functionality. Gitlab is often installed within a group or institution that provides a local repository which is separate from gitlab.com.

SourceForge

It is available from 1999 with more than 3.7 million users. SourceForge lost its importance once Git came into existence in 2011.

BitBucket

Launchpad is hosted in 2004 by Canonical and lists some significant projects as users, such as Ubuntu and MySQL.

GitLab

Assembla has a strong following amongst smaller companies and has extensive project-management facilities in addition to the software-development services.

Facilitator Notes:

Discuss about different source code repositories.

SourceForge: SourceForge was one of the best known project hosting site, but lost its importance to Git in May 2011. Some users have found sourceforge net server can be a little slow at times of high demand. In some countries like Iran and Syria SourceForge is not accessible.

Launchpad: Launchpad is hosted by Canonical and lists some significant projects as users, such as Ubuntu and MySQL. It provides a system (Blueprints) for the feature and specifications tracking and the Soyuz release-management system.

Assembla: Assembla has a strong following amongst smaller companies and has extensive project-management facilities in addition to the software-development services.

2.1.1 Version Control System

Following are the key details of version control system:

- Version control systems are a category of software tools that help a software team manage changes to source code over time.
- Version control software keeps track of every modification to the code in a special kind of database.
- If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.
- Version control software is an essential part of the everyday of the modern software team's professional practices.

Facilitator Notes:

Discuss about version control system

The source code repository has in-built version control system. In this slide we will discuss about CVS and SVN.

CVS:

- CVS is a very old revision control system. It was first released in 1986, and Google Code still hosts the original Usenet post announcing CVS. CVS is the de facto standard and is installed virtually everywhere. However, the code base isn't as fully featured as SVN or other solutions.
- The learning curve isn't too steep for CVS, and it's a very simple system for making sure files and revisions are kept up to date. While CVS may be an older technology, it's still quite useful for any designer or developer for backing up and sharing files.

- Tortoise CVS is a great client for CVS on Windows, and there are many different IDEs, such as Xcode (Mac), Eclipse, NetBeans and Emacs, that use CVS.

SVN:

- Subversion is probably the version control system with the widest adoption. Most open-source projects use Subversion as a repository because other larger projects, such as SourceForge, Apache, Python, Ruby and many others, use it as well. Google Code uses Subversion exclusively to distribute code.
- Because of Subversion's popularity, many different Subversion clients are available. If you're a Windows user, Tortoise SVN is a great file browser for viewing, editing and modifying your Subversion code base. If you're on a Mac, Versions is an elegant client that provides a 'pleasant way to work with Subversion.' Xcode is Apple's developer environment and Subversion client that ships with Leopard on a Mac.

Centralized vs Distributed version control system:

- Distributed version control systems (DVCSs) solve different problems than Centralized VCSs.
- Centralized VCS systems are designed with the intent that there is One True Source that is trusted. All developers work (checkout) from that source, and then add (commit) their changes. The only real difference between CVS, Subversion, ClearCase, Perforce, VisualSourceSafe and all the other DVCSes is in the workflow, performance, and integration that each product offers.
- Distributed VCS systems are designed with the intent that one repository is as good as any other, and that merges from one repository to another are just another form of communication. Any semantic value as to which repository should be trusted is imposed from the outside by process, not by the software itself.

The real choice between using one type or the other is organizational—if your project or organization wants centralized control, then a DVCS is a non-starter. If your developers are expected to work all over the country/world, without secure broadband connections to a central repository, then DVCS is probably your salvation.

2.2 How to Install GitHub

There are multiple ways to use Git, using GUI tools and command line tools.

Command line: All the git commands can be executed.

GUI: Only partial subset of git functionality can be executed through the GUI

- The most official build is available for download on the Git website. <http://git-scm.com/download/win>
- Git can be installed on Windows, Linux and MacOS.

Facilitator Notes:

Explain how to install the Github.

Github is the widely used open source repository. In this section we will see how to install Git hub and access it.

Installing on Windows

- The most official build of git can be downloaded on the Git website - <http://git-scm.com/download/win>.

Installing from Source

- On Linux, generally people prefer to install Git from source, because they can install the most recent version compare to the binary version.
- The following libraries should be installed on Linux before installing Git: openssl, autotools, zlib, curl, libiconv and expat.
- **Debian / Ubuntu (apt-get)**
 - \$ sudo apt-get update
 - \$ sudo apt-get install git
- **CentOS/Linux (yum)**
 - \$ sudo yum install git
- Verify the git version after installing it:
 - \$ git --version
 - git version 2.9.3

2.3 First Time Git Setup

After installing git on the system, you will need to configure a few things to customize your git environment. Below steps need to be performed initially and only once:

- **/etc/gitconfig file:** Contains values applied to every user on the system and all their repositories
- **~/.gitconfig or ~/.config/git/config file:** Values are specific to the user. You can configure Git read and write to the above file specifically by passing global option, and this affects all the repositories that are configured on the system.

Facilitator Notes:

Explain how to setup the Github.

Review copy only not to be circulated without prior permissions from Xebia

The first after installing Git is to configure user name and email address. Every git coming uses this information, so this step is very important.

\$ git config --global user.name "John Doe"

\$ git config --global user.email johndoe@example.com

Checking Your Settings

To check your configuration settings, you can use the git config --list command to list all the settings Git can find at that point:

- \$ git config --list
- user.name=John Doe
- user.email=johndoe@example.com
- color.status=auto
- color.branch=auto
- color.interactive=auto
- color.diff=auto

2.4 How to Create an Organization in GitHub

The screenshot below shows the user interface to create an organization account in GitHub.

Sign up your team

Step 1:
Set up the organization

Step 2:
Invite members

Step 3:
Organization details

Create an organization account

Organization name *

This will be your organization name on <https://github.com/>.

Billing email *

We'll send receipts to this inbox.

Choose your plan

<p><input checked="" type="radio"/> Free</p> <p>Unlimited users and public repositories</p>	\$0
<p><input type="radio"/> Team</p> <p>Starts at \$25 / month which includes your first 5 users.</p> <p>Unlimited public repositories</p> <p>Unlimited private repositories</p>	\$9 per user / month
<p><input type="radio"/> Business</p> <p>Includes everything in the Team plan, plus:</p> <p>SAML based single sign-on (SSO)</p> <p>Access provisioning</p> <p>99.95% uptime SLA</p> <p>24/5 email support with < 8-hour response time</p> <p>Learn more about our Business Plan or contact our team.</p>	\$21 per user / month

Organization accounts allow your team to plan, build, review, and ship software — all while tracking bugs and discussing ideas.

The credit card and plan you choose will be billed to the organization — not gnanu1228 (your user account).

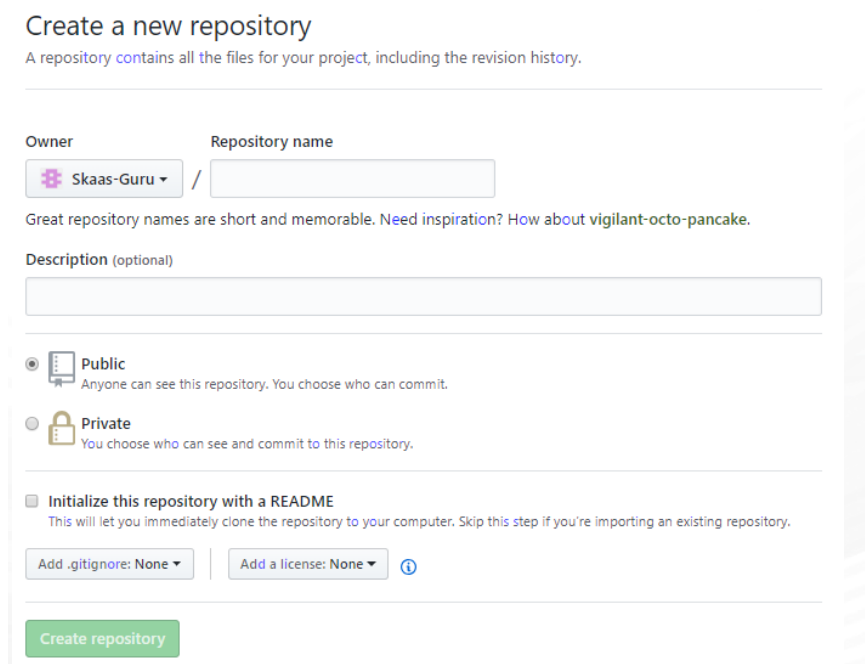
Facilitator Notes:

Explain how to create an organization in github.

- First step is to create an Organization by selecting the plan. For public repositories there is no charge. If you want to add a team to github there is a charge.
- Second step is to invite or add your team members to your organization.
- Final step is to provide organization details like number of team members, the nature of this repository, etc.

2.5 How to Create a Repository in GitHub

The screenshot below shows the UI for creating a new repository on GitHub.



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and 'A repository contains all the files for your project, including the revision history.' Below this, there are two input fields: 'Owner' (with a dropdown menu showing 'Skaas-Guru') and 'Repository name' (an empty text box). A note below these fields says 'Great repository names are short and memorable. Need inspiration? How about vigilant-octo-pancake.' Underneath is a 'Description (optional)' text area. Further down, there are two radio button options: 'Public' (selected) and 'Private'. Below these are checkboxes for 'Initialize this repository with a README' and 'Add .gitignore: None' and 'Add a license: None'. At the bottom is a green 'Create repository' button.

Facilitator Notes:

Explain how to create a repository in github.

On GitHub website, you can create repositories under your organization:

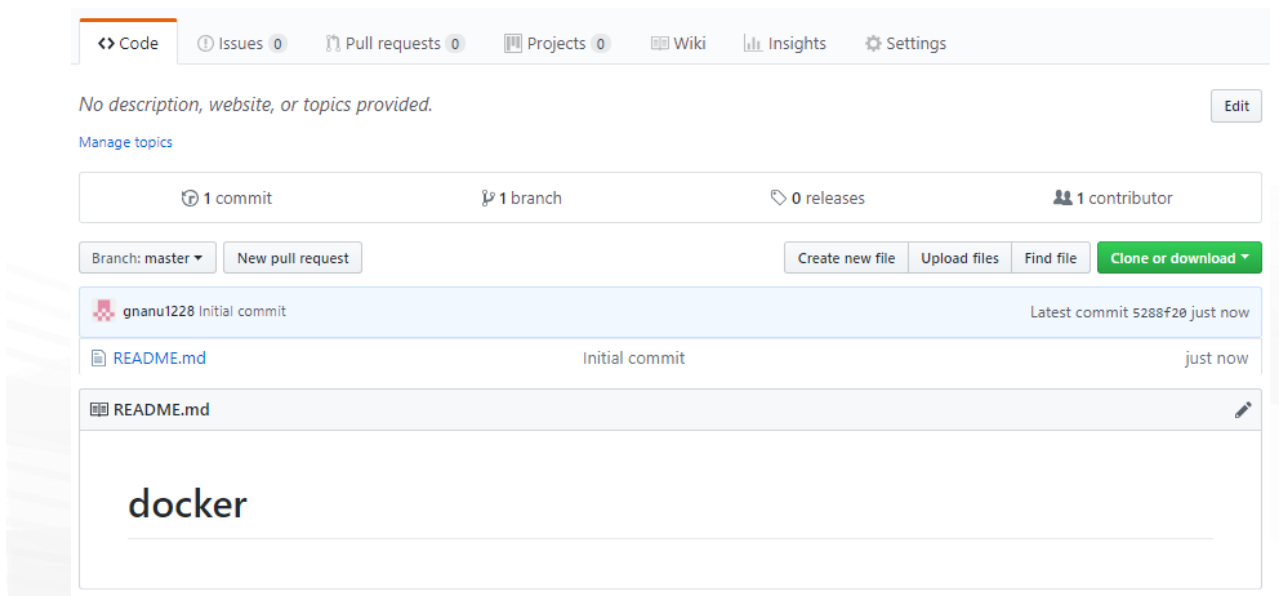
1. On GitHub, navigate to the home
2. Click the New Repository selector menu
3. Type a unique name for your new repo

Review copy only not to be circulated without prior permissions from Xebia

4. Select the type of repository (Private/Public)
5. Select to initialize the read me if you do not have an existing repository configured
6. Press **Create repository**

2.6 How to Create a Branch in Github

The screenshot below shows the GitHub UI for creating a new branch.



Facilitator Notes:

Explain how to create a branch and checkin the code to github.

Steps for creating a new branch on GitHub.

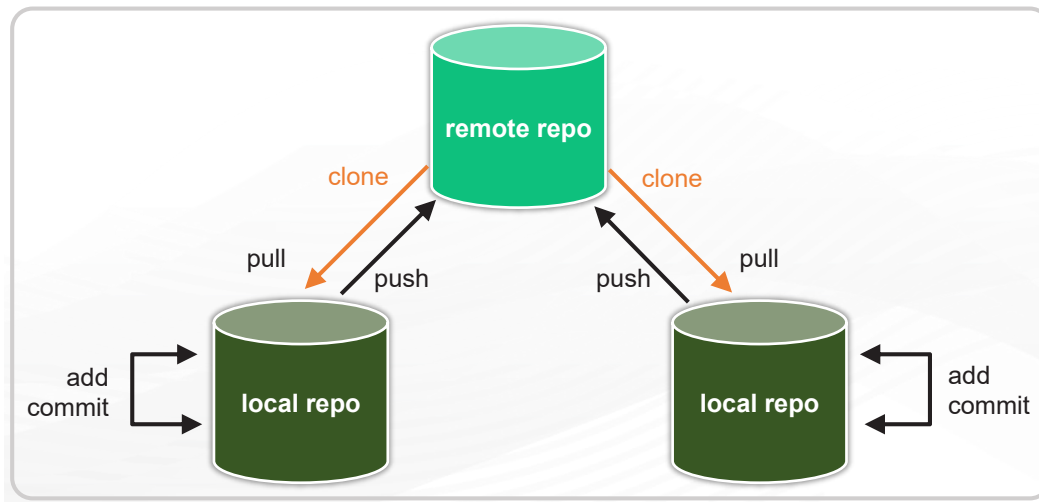
- Step 1: Create a local git repository
- Step 2: Create a commit
- Step 3: Create a new branch or Push the local code to existing repository
- Step 4: Push a branch to GitHub

On GitHub website, you can directly create or delete branches:

1. On GitHub, navigate to the main page of the repository
2. Click the branch selector menu
3. Type a unique name for your new branch
4. Press **Enter**

2.7 How to Check-in Code to Github

The diagram illustrates the process of source code check-in from local repo to the remote repo.



Facilitator Notes:

Explain how to check in source code to github repository.

The steps to check-in the source code from the local repository to the remote repository is given below.

1. **Open git command line and navigate to your project**
2. **Set your origin before checking in the code**
`git remote add origin https://github.com/test/test-repo.git`
3. **Type git add . to add all your local changes**
4. **Type git commit -m "First commit" to commit local changes**
5. **Push your branch to git hub**

`git push origin master`

Group Discussion



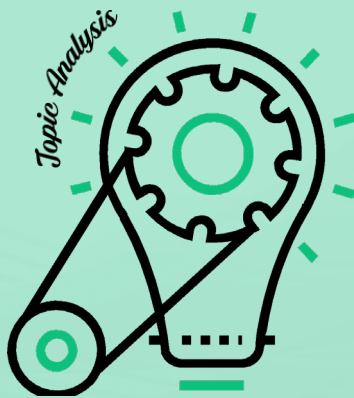
Review copy only not to be circulated without prior permissions from Xebia

Facilitator Notes:

Divide the students into groups and discuss about the different types of source code repositories.

We've so far seen about the important concepts associated with release lifecycle, source code repositories and different type of SCM. Form different groups and each group should talk about one concept in detail along with analogies or examples to show your understanding.

What did You Grasp?



1. Nightly builds are examples of which builds?
 - A) Alpha
 - B) Pre-Alpha
 - C) Beta
 - D) Release candidate
 - E) General availability
2. State True or False
In release candidate high priority bugs can be fixed?
 - A) True
 - B) False



3. In which stage, the first version of the product is released to external people?
 - A) Alpha
 - B) Pre-Alpha
 - C) Beta
 - D) Release candidate
 - E) General availability
4. Which order is followed in git to commit the code to repository?
 - A) Init, add, push, commit
 - B) Init, add, commit, push
 - C) Init, commit, add, push
 - D) Init, push, commit, add

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. b. Pre-Alpha
2. b. False: Generally high and low priority bugs are not addressed in release candidate build. Only critical bugs will be fixed at this stage.
3. c. Beta
4. b. Init, add, commit, push

3. Deploying Build to Production

The following factors have to be considered while doing production deployments:

Automate as much as possible

Build and pack your application only once

Deploy the same way all the time

Deploy using feature flags in your application

Deploy in small batches, and do it often

Facilitator Notes:

Discuss about production deployment tips.

The following factors have to be considered while doing production deployments:

1. **Automate as much as possible:** Deployments can be as easy as clicking a button. Even reverting the deployment is a button-click away. You can get to the point where anyone, even non-technical people, can do deployments. Tools like Ansible, Jenkins, Terraform can be used to automate the deployments. Automation makes deployments smoother by letting you move forward and backward with ease. The process becomes repeatable and reliable.
2. **Build and pack your application only once:** The build process usually takes a while, so it will make things run so much more smoothly at deployment time if you only build once. You can keep coding and push changes all the time without worrying about promoting code that's not ready. You'll want to practice continuous integration to make packing easier, and one of its fundamental principles is that you build only once. Practices like this allow you to integrate constantly in a shared repository, verify the process with automated tests, and fix things quickly if needed.
3. **Deploy the same way all the time:** There's no value in applying the previous tips if you don't deliver the same way in all circumstances. You packed once to deploy the same code everywhere, and you need to deliver the same way. By doing this, your production deployments will not have any issue. For this, you need to have production-like environments. This doesn't necessarily mean that you'll have the same data everywhere—that can be expensive and a possible security risk. It just means that if you have a load balancer in the production because you need to be able to scale out/in, you also have to have one in development. If you're hosting production in the cloud, you need to do the same for development.
4. **Deploy using feature flags in your application:** Using feature flags will increase your confidence to deploy. And the benefit is even bigger when you combine them with strategies like blue/green

and canary deployments. These strategies will help you to deploy without customers noticing there was a change.

5. **Deploy in small batches, and do it often:** Deploying in small batches and doing it frequently becomes natural. It will take some time to get there, especially because deployments should be predictable and reliable. But it'll be worth it. Now, automated testing is a prerequisite. It doesn't matter how much effort you put into automating your deployments—if at the end of the day you spend a lot of time manually testing your application, you won't see the benefit. So try to spend time automating your test cases. Otherwise, there's no point in doing automated deployments.

3.1 Maven Prepare Goal

Preparing a release goes through the following phases:

- The main aim is to isolate each unit of the system to identify, analyze and fix the defects.
- No uncommitted changes should exist in the sources.
- No SNAPSHOT dependencies should exist.
- Change the version in the POMs from SNAPSHOT to a new version (you will be asked for the new version to use).
- Any new changes made to code should be reverted before you start a prepare process, `release:rollback` can revert back the changes for you.
- Run the tests on modified POMs to confirm everything is working fine.
- Tag the source code with a version name in the SCM.

Facilitator Notes:

Discuss about Maven prepare goal.

Maven Goals: A **Maven** plugin is a container for/supplier of **goals**. Code implemented in **goals** is the real workhorse. Each of a plugin's **goals** can be assigned/bound to any of the lifecycle phases.

Goals are executed in phases which help determine the order goals get executed in. The best understanding of this is to look at the default maven lifecycle bindings which shows which goals get run in which phases by default. The compile phase goals will always be executed before the test phase goals which will always be executed before the package phase goals and so on.

Default maven lifecycle reference - http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Built-in_Lifecycle_Bindings

Maven Prepare Goal

- To prepare a release, below command should be executed:

`mvn release:prepare`

- If any error is encountered while running above command, or prepare process is cancelled, then run below command to pick up from where the last last execution failed:

mvn release:prepare -Dresume=false

Or you can use **mvn release:clean release:prepare**

- If you would have made any changes to source code, it should be reverted before starting the prepare process. To revert back the changes run **release:rollback**
- For multi-module projects you might need to enter the version number for each project. If you would like to get the version from Parent POM for every project, then set `autoVersionSubmodules` to true, you will be asked only once to enter the version.
- To generate release POM's run this command - **mvn release:prepare-with-pom**
- If you do not specify any tag name, default tag `artifactId-version` will be used. However, you can specify the exact tag name by specifying **tagNameFormat in pom**. Example configuration:

```
<project>
[... ]
<build>
[... ]
<plugins>
[... ]
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.5.3</version>
  <configuration>
    <tagNameFormat>v@{project.version}</tagNameFormat>
  </configuration>
</plugin>
[... ]
</plugins>
[... ]
</build>
[... ]
</project>
```

3.2 Maven Perform Goal

Performing a release runs the following phases:

- Checkout from a repository URL with optional tag.
- The project can be released by running the predefined Maven goals (deploy and site-deploy).
- To perform a release, execute this command:
mvn release:perform – It depends on **release.properties** from previous release preparation.
- If release properties are not used, you might need to specify the goal and optional tag to perform the release.

Facilitator Notes:

Discuss about maven perform goal.

- A new Maven instance, will be forked by release:perform command to build the checked-out project. The new forked out Maven instance will use the same maven profiles and system configuration used by running the release:perform goal.
- It is possible to force profile names enabled during release by setting a comma separated in the releaseProfiles parameter. Then the goals and profiles required to release the project can be configured in the POM.

Example plugin configuration to perform a release:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.5.4</version>
  <configuration>
    <!--
      During release:perform, enable the "release" profile
    -->
    <releaseProfiles>release-product</releaseProfiles>
  </configuration>
</plugin>
```

- Once release is completed successfully, release.properties and other files used for released will be automatically cleaned.

3.3 Maven Clean Goal

Cleaning a release goes through the following phases:

- Delete the release descriptor (release.properties)
- Delete backup POM files
- To clean a release, execute this command:
mvn release:clean
- This plugin can be used to cleanup a failed or abandoned release or to perform a dry run

Facilitator Notes:

Discuss about maven clean goal.

- Maven uses this plugin to clean up after a release preparation. This is done after a successful release:perform.
- Full name of release clean plugin : org.apache.maven.plugins:maven-release-plugin:2.5.3:clean
- This plugin cleans only current working copy and it will not rollback previous steps.

3.4 Maven Rollback Goal

To rollback a release, the following requirement must be met:

- You haven't run release:clean on the project. This means that the backup files and the release descriptor from the previous release command still exists.
- The following phases are executed when a release is rolled back:
 - ↳ All project POMs are reverted back to their pre-release state locally, and also in the SCM if the previous release command was able to successfully make changes in the SCM to the POMs. This is done by using the backup files created during release:prepare.
 - ↳ The created branch/tag in SCM for the release is removed.

Facilitator Notes:

Discuss about the maven rollback goal.

Review copy only not to be circulated without prior permissions from Xebia

- To rollback a release, execute this command:

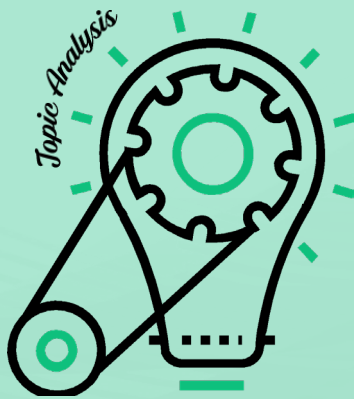
```
mvn release:rollback
```

Note: If the process is cancelled or an error occurs, then running the rollback command will be executed where the last step left off.

- The created branch/tag in SCM for the release is removed.

Note: This feature is not yet implemented, so branch/tag should be manually removed from your SCM. For more info see MRELEASE-229

What did You Grasp?



1. Maven release clean goal will cleanup _____?
 A) Target directory
 B) Backup POM files
 C) Project artifacts
 D) Noe of the above
2. State True or False
 Maven release perform depends on release prepare?
 A) True
 B) False



3. State True or False
 To run maven rollback goal, maven clean goal should be executed.
 A) True
 B) False
4. From which of the following phases does the Maven rollback goal use files to perform a rollback?
 A) Release:clean
 B) Release:perform
 C) Release:prepare
 D) Release:rollback

Facilitator Notes:

Tell the participants that they will be going through a knowledge check question.

Answer:

1. b. Backup POM files
2. a. True
3. b. False
4. c. Release:prepare

In a nutshell, we learnt:



1. Release lifecycle is a standard approach to carry out development of a product.
2. Different stages of a release cycle is Pre-Alpha, Alpha, Beta, Release candidate and General availability.
3. A source code repository is a web hosting and file archive facility where a large amount of source code, is maintained, either publicly or privately.
4. Git is the widely used source code repository.
5. The main aim of release is to isolate each unit of the system to identify, analyze and fix the defects.
6. To run maven rollback goal you haven't run release:clean on the project.

Facilitator Notes:

Share the module summary with the audience.

Ask the participants if they have any questions. They can ask their queries by raising their hands.