

Java Keywords

- **Java** -- Application Program
- **JDK** -- Java Development Kit and it is Software Bundle
- **JRE** -- Java Runtime Environment which is having set of Lib and Bin
- **JVM** -- Java Virtual Machine to execute Java Program

32

32

Application Build:

- Java ->
 - Source File - FileName.java
 - Binary File - FileName.class

Source code is managed in 'src' directory

Binary code is managed in 'build/classes or release or target' directory
- Converting your source code into binary
 - Compilation:
 - --> Source Code --> Compiler --> Binary Code
- Binaries:- Executable, artifacts, outcomes

33

33

Web Project Build

- Two Applications:
 - 1) Standard Alone - Stand Alone Machine (Using Jdk)
 - Contains Java Classes
 - The packaging file format is .jar
 - 2) Web Apps - Hosted on App Server
- Web Application: Binary -> .war
 - src/ .java
 - build/classes/ .class
 - WebContent/ .css
 - .js
 - .html
 - .jsp
 - .xml
 - images

34

Web Application Structure

Directory or File	Description
MyWebApp	Public document root of Web application
WEB-INF	Private resources not served directly to clients
classes	Classes, such as servlets, filters, listeners
lib	Java libraries (JAR files)
web.xml	Optional Java EE deployment descriptor
weblogic.xml	Optional WebLogic deployment descriptor
index.html	Static and dynamic Web content
page1.jsp	
page2.jsp	

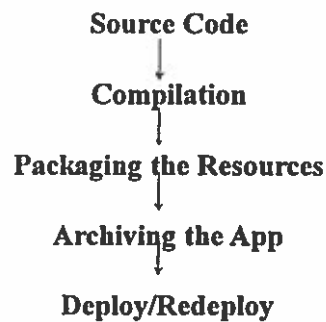


MyWebApp.war

35

35

Deployment Process



36

36

What is a Build Tool?

- A build tool is a tool that automates everything related to building the software project and it 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 or ZIP files.
- Installing the packaged code on a server, in a repository or somewhere else.

37

37

Database Design?

RISE 'N' SHINE TECHNOLOGIES

111

111

Table Name - Employee

Columns

Rows

Emp ID	Name	Department	Salary
1	x	DevOps	5Lakhs
2	y	AWS	4Lakhs
3	z	DevOps & AWS	6 Lakhs

RISE 'N' SHINE TECHNOLOGIES

112

112

Table Name - Address

Emp ID	Line 1	Line 2	City	State
1	11	L2	Hyd	TS
2	11	L2	Hyd	TS
3	11	L2	Hyd	TS

RISE 'N' SHINE TECHNOLOGIES

113

113

What is XML?

➤ A meta language that allows you to create and format your own document markups

➤ These files are

- easy to read
- unambiguous
- extensible
- platform-independent

➤ File Extension is

- .xml

RISE 'N' SHINE TECHNOLOGIES

114

114

XML Key Words:

- Attribute
 - Name
 - Value
- Element
 - can contain text, other elements or a combination
- Root Element

RISE 'N' SHINE TECHNOLOGIES

115

115

A well-formed XML document

➤ **Elements** have an open and close tag, unless it is an empty element

```
<tag> text </tag>
```

➤ if a tag is an **empty element**, it has a closing / before the end of the tag

```
<tag/>
```

➤ open and close tags are nested correctly

RISE 'N' SHINE TECHNOLOGIES

116

116

XML basics

the XML declaration

```
<?xml ?>
```

-not required, but typically used

-attributes include:

- version
- encoding: the character encoding used in the document

```
<?xml version="1.0" encoding="UTF-8">
```

RISE 'N' SHINE TECHNOLOGIES

117

117

XML basics

```
<!-- --> comments
```

- contents are ignored by the processor
- cannot come before the XML declaration
- cannot appear inside an element tag
- may not include double hyphens

RISE 'N' SHINE TECHNOLOGIES

118

118

Sample XML File:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<Employees>
  <Employee>
    <id>1</id>
    <name>x</name>
    <dept>DevOps</dept>
    <sal>5lakhs</sal>
    <Address>
      <line1>l1</line>
      <line2/>
      <city>hyd</city>
      <state>ts</state>
    </Address>
  </Employee>
```

RISE 'N' SHINE TECHNOLOGIES

119

119

Sample XML File: Cont...

```
<Employee>
  <id>2</id>
  <name>y</name>
  <dept>AWS</dept>
  <sal>5Lakhs</sal>
  <Address>
    <line1>l1</line>
    <line2>l2</line2>
    <city>hyd</city>
    <state>ts</state>
  </Address>
</Employee>
```

```
-----
-----
-----
```

```
</Employees>
```

RISE 'N' SHINE TECHNOLOGIES

120

120

Maven

44

44

What is Maven?

- It is a powerful build tool for Java software projects.
- Maven is developed in Java
- The Maven can be downloaded from:
- <http://maven.apache.org>

45

45

What is Maven?

- Maven provides developers a complete build lifecycle framework.
- Maven simplifies and standardizes the project build process.
- It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly.
- Maven increases reusability and takes care of most of build related tasks.

46

46

Maven Environment Setup

- System Requirement
 - JDK 1.5 or above.
 - Memory no minimum requirement.
 - Disk Space no minimum requirement.
 - Operating System no minimum requirement.

47

47

Java installation

Step 1: Set JAVA environment

- Set the JAVA_HOME environment variable to point to the base directory location where Java is installed.

Step 2 - verify Java installation

- Windows Open Command Console `c:\> java -version`
- Linux Open Command Terminal `$ java -version`

48

48

Maven Installation

- **Step 3: Download Maven archive**

Download Maven 3.3.3 from

-- <http://maven.apache.org/download.cgi>

- **OS Archive name**

- Windows `apache-maven-3.3.3-bin.zip`
- Linux `apache-maven-3.3.3-bin.tar.gz`

- **Step 4: Extract the Maven archive**

- Extract the archive, to the directory you wish to install Maven 3.3.3. The subdirectory `apachemaven-3.3.3` will be created from the archive.

49

49

Maven Installation

- **Step 5: Set Maven environment variables**
- Add M2_HOME, MAVEN_OPTS to environment variables.
- Windows: Set the environment variables using system properties.
 - *M2_HOME=... \apache-maven-3.3.3*
 - *MAVEN_OPTS=-Xms256m -Xmx512m*
- Linux: Open command terminal and set environment variables.
 - *export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.3*
 - *export MAVEN_OPTS=-Xms256m -Xmx512m*

50

50

Maven Installation

Step 6: Add Maven bin directory location to system path

- Now append M2 variable to System Path
- Windows
 - Append the string ;%M2% to the end of the system variable, Path.
- Linux
 - *export PATH=\$M2_HOME/bin:\$PATH*

51

51

Maven Installation

- **Step 8: Verify Maven installation**
- Now open console, execute the following **mvn** command.
- Windows:
 - Open Command Console `c:\> mvn --version`
- Linux:
 - Open Command Terminal `$ mvn --version`

52

52

Maven Core Concepts – POM File

- Maven is centered around the concept of POM files (Project Object Model).
- A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc.
- It contains a detailed description of your project, including information about versioning and configuration management, dependencies, application and testing resources, team members and structure, and much more.
- The POM file should be located in the root directory of the project it belongs to.

53

53

Maven POM Files

- All POM files require the **project** element and three mandatory fields: **groupId**, **artifactId**, **version**.
- Projects notation in repository is **groupId:artifactId:version**.
- Root element of POM.xml is **project** and it has three major sub-nodes :

54

Example POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId> org.rns.training</groupId>
  <artifactId>maven-training</artifactId>
  <version>1.0</version>

</project>
```

55

Elements of POM:

- **groupId:**
- This is an Id of project's group. This is generally unique amongst an organization or a project.
 - For example, a banking group `com.company.bank` has all bank related projects.
- **artifactId:**
- This is an Id of the project. This is generally name of the project.
 - For example, `consumer-banking`. Along with the `groupId`, the `artifactId` defines the artifact's location within the repository.
- **Version:**
- This is the version of the project. Along with the `groupId`, It is used within an artifact's repository to separate versions from each other.

56

56

Elements of POM:

- For example:
 `com.company.bank:consumer-banking:1.0`
 `com.company.bank:consumer-banking:1.1.`
- **name** - This is the human-readable name of the project.
- **url** - This is the url for the project website.
- **packaging** - This defines the “style” of project your building (e.g. ear, jar, war, etc.).
 - For more information on packaging, check out the respective plugins (maven-jar-plugin, maven-war-plugin, etc.)

57

57

Elements of POM

- **dependencies** - Specifies the dependencies of the project. This will be materialized from any defined maven repositories. More details later
- **repositories** - Specifies alternative locations for maven to look when materializing dependencies.
- **pluginRepositories** - Specifies alternative location for maven to look when materializing build plugins
- **build** - Specifies configuration on **how** to build the project
- **reports** - Specifies configuration on **what** reports to generate for the project.

58

58

Maven - Project Templates

- Maven provides a very large list of different types of project templates using concept of **Archetype**.
- Maven helps to quickly start a new java project using following command
 - mvn archetype:generate
- What is Archetype?
 - Archetype is a Maven plug-in whose task is to create a project structure as per its template.

We are going to use *quickstart* archetype plug-in to create a simple java application here.

59

59

Maven - Creating Project

- To create a simple java application, we'll use *maven-archetype-quickstart* plugin. In example below, We'll create a maven based java application project.

```

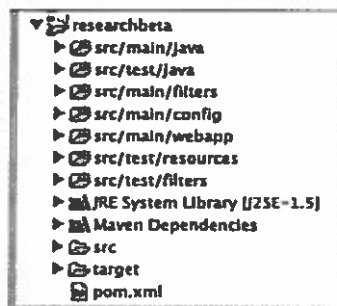
-mvn archetype:generate -DgroupId=com.rns.app -
-DartifactId=maven-app -DarchetypeArtifactId=maven-
archetype-quickstart -DinteractiveMode=false

```

60

60

Default Directory Layout

 <ul style="list-style-type: none"> researchbeta <ul style="list-style-type: none"> src/main/java src/test/java src/main/filters src/main/config src/main/webapp src/test/resources src/test/filters JRE System Library [J2SE-1.5] Maven Dependencies src target pom.xml 	<p>src/main/java: Your Java source code goes here (strangely enough!)</p> <p>src/main/resources: Other resources your application needs</p> <p>src/main/filters: Resource filters, in the form of properties files, which may be used to define variables only known at runtime</p> <p>src/main/config: Configuration file</p> <p>src/main/webapp: The Web application directory for a WAR project</p> <p>src/test/java: Unit tests</p> <p>src/test/resources: Resources to be used for unit tests, but will not be deployed</p> <p>src/test/filters: Resources filters to be used for unit tests, but will not be deployed</p>
---	---

61

61

Maven - Build & Test Project

- Let's open command console, go the project directory and execute the following **mvn** command.

> mvn clean package

- Now open command console, go the project\target\classes directory and execute the following java command.

> java com.rns.app.App

62

62

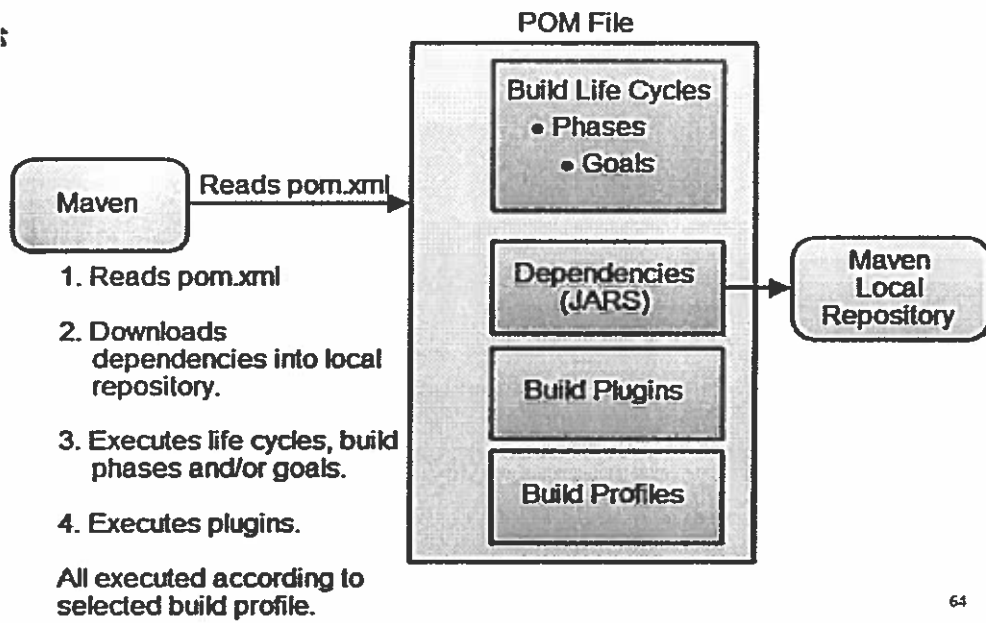
key learning concepts

- We give maven two goals, first to clean the target directory (clean) and then package the project build output as jar(package).
- Packaged jar is available in artifact_id\target folder as artifact_id-1.0.jar.
- Test reports are available in artifact_id\target\surefire-reports folder.
- Maven compiled source code file(s) and then test source code file(s).
- Then Maven run the test cases.
- Finally Maven created the package.

63

63

Maven Core Concepts – POM File



64

Maven core concepts - 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.
- Ex: clean, compile, testcompile, test, package, install, deploy
- When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal.

65

65

Build Life Cycles

- Maven has 3 built-in build life cycles. These are:
 - 1. default
 - 2. clean
 - 3. site
- Each of these build life cycles are executed independently of each other.
- You can get Maven to execute more than one build life cycle, but they will be executed in sequence, separately from each other, as if you had executed two separate Maven commands.

Ex : mvn clean package site

66

66

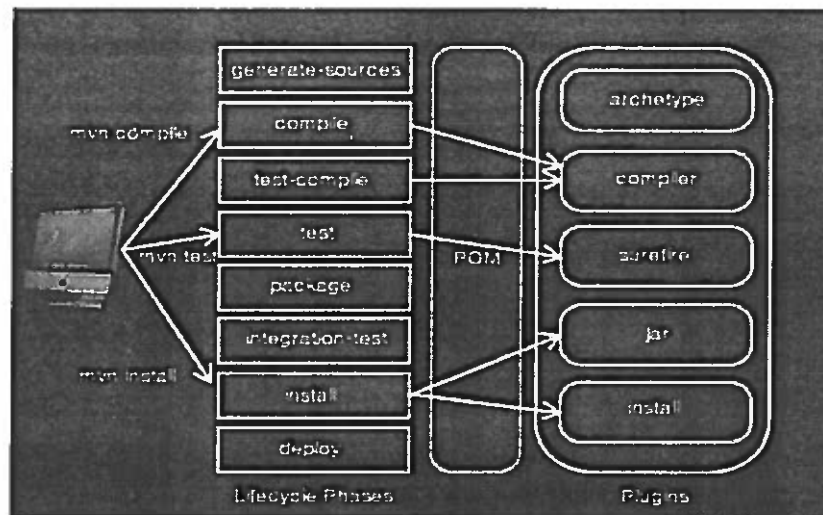
Build Life Cycles

- 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. In fact, site can generate a complete website with documentation for your project.

67

67

Maven Lifecycle



68

68

Maven lifecycle phases

Some of the more useful Maven lifecycle phases are the following:

- **generate-sources:** Generates any extra source code needed for the application, which is generally accomplished using the appropriate plug-ins
- **compile:** Compiles the project source code
- **test-compile:** Compiles the project unit tests
- **test:** Runs the unit tests (typically using JUnit) in the src/test directory
- **package:** Packages the compiled code in its distributable format (JAR, WAR, etc.)
- **integration-test:** Processes and deploys the package if necessary into an environment where integration tests can be run
- **install:** Installs the package into the local repository for use as a dependency in other projects on your local machine
- **deploy:** Done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and project.

69

Running Maven

- You can execute multiple build life cycles or phases by passing more than one argument to the mvn command. Here is an example:
 - mvn clean install
- This command first executes the clean build life cycle, which removes compiled classes from the Maven output directory, and then it executes the install build phase.

70

70

Demo

- How to generate Java Docs of the project?
 - mvn clean site
- How to run Junit Test Cases?
 - mvn clean test
- How to generate the Junit Test Case Reports?
 - mvn clean test site

71

71

Maven Settings File

- In the settings file you can configure settings for Maven across all Maven POM files. For instance, you can configure:
 - Location of local repository
 - Active build profile, etc....
- The settings files are called settings.xml and it is located at:
- The Maven installation directory: \$M2_HOME/conf/settings.xml

72

72

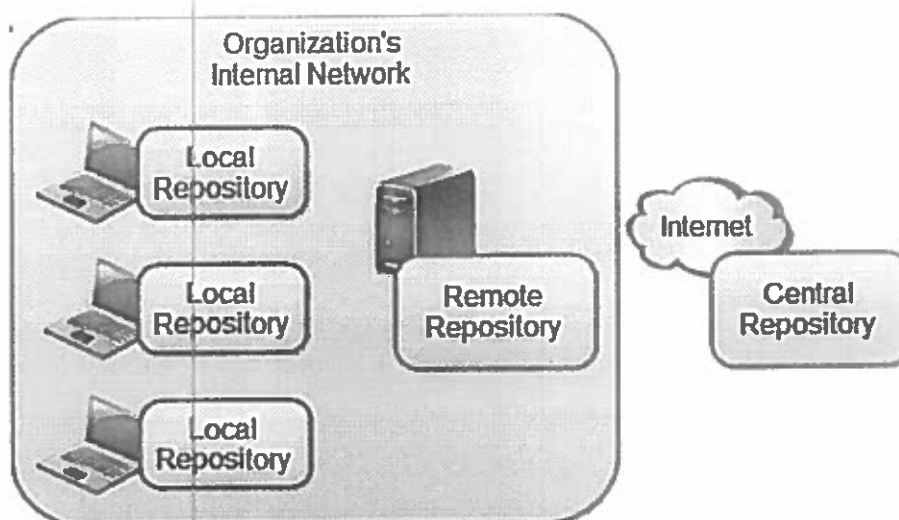
Maven - Repositories

- What is a Maven Repository?
- In Maven terminology, a repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.
- Maven repository are of three types
 - local
 - central
 - remote

73

73

Maven - Repositories



74

74

Local Repository

- Maven local repository keeps your project's all dependencies (library jars, plugin jars etc).
- When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies stored on remote machine every time a project is build.
- Maven local repository by default get created by Maven in %USER_HOME% directory.
- To override the default location, mention another path in Maven settings.xml file available at %M2_HOME%\conf directory.

75

75

Local Repository

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <localRepository>C:/MyLocalRepository</localRepository>

</settings>
```

76

76

Central Repository

- Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.
- When Maven does not find any dependency in local repository, it starts searching in central repository using following URL: <http://repo1.maven.org/maven2/>
- Key concepts of Central repository
 - This repository is managed by Maven community.
 - It is not required to be configured.
 - It requires internet access to be searched.
- To browse the content of central maven repository URL: <http://search.maven.org/#browse>

77

77

Remote Repository

- Sometime, Maven does not find a mentioned dependency in central repository as well then it stopped build process and output error message to console.
- To prevent such situation, Maven provides concept of **Remote Repository** which is developer's own custom repository containing required libraries or other project jars.

78

Remote Repository

- ```
<dependencies>
 <dependency>
 <groupId>com.companyname.common-lib</groupId>
 <artifactId>common-lib</artifactId>
 <version>1.0.0</version>
 </dependency>
</dependencies>
```
- ```
<repositories>
  <repository> <id>companyname.lib1</id>
    <url>http://download.companyname.org/maven2/lib1</url>
  </repository>
  <repository> <id>companyname.lib2</id>
    <url>http://download.companyname.org/maven2/lib2</url>
  </repository>
</repositories>
```

79

Maven Dependency Search Sequence

- **Step 1** - Search dependency in local repository, if not found, move to step 2 else if found then do the further processing.
- **Step 2** - Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4 else if found, then it is downloaded to local repository for future reference.
- **Step 3** - If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
- **Step 4** - Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference otherwise Maven as expected stop processing and throws error (Unable to find dependency).

80

Maven - Snapshots

- A large software application generally consists of multiple modules and it is common scenario where multiple teams are working on different modules of same application.
- For example consider a team is working on the front end of the application as app-ui project (app-ui.jar:1.0) and they are using data-service project (data-service.jar:1.0).
- Now it may happen that team working on data-service is undergoing bug fixing or enhancements at rapid pace and the they are releasing the library to remote repository almost every other day.

81

81

Maven - Snapshots

- Now if data-service team uploads a new version every other day then following problem will arise
 - data-service team should tell app-ui team every time when they have released an updated code.
 - app-ui team required to update their pom.xml regularly to get the updated version
- To handle such kind of situation, **SNAPSHOT** concept comes into play.

82

82

What is SNAPSHOT?

- **SNAPSHOT** is a special version that indicates a current development copy. Unlike regular versions, Maven checks for a new **SNAPSHOT** version in a remote repository for every build.
- Now data-service team will release **SNAPSHOT** of its updated code everytime to repository say data-service:1.0-SNAPSHOT replacing a older **SNAPSHOT** jar.

83

83

Snapshot vs Version

- In case of Version, if Maven once downloaded the mentioned version say data-service:1.0, it will never try to download a newer 1.0 available in repository. To download the updated code, data-service version is be upgraded to 1.1.
- In case of SNAPSHOT, Maven will automatically fetch the latest SNAPSHOT (data-service:1.0-SNAPSHOT) every time app-ui team build their project.

84

84

app-ui pom.xml

- app-ui project is using 1.0-SNAPSHOT of data-service

```

<dependencies>
  <dependency>
    <groupId>data-service</groupId>
    <artifactId>data-service</artifactId>
    <version>1.0-SNAPSHOT</version>
    <scope>test</scope>
  </dependency>
</dependencies>

```

85

85

data-service pom.xml

- Data-service project is releasing 1.0-SNAPSHOT for every minor change

```
<groupId>data-service</groupId>
```

```
<artifactId>data-service</artifactId>
```

```
<version>1.0-SNAPSHOT</version> <packaging>jar</packaging>
```

- Although, In case of SNAPSHOT, Maven automatically fetches the latest SNAPSHOT on daily basis. You can force maven to download latest snapshot build using -U switch to any maven command.

```
- > mvn clean package -U
```

86

86

Maven - Web Application

- how to manage a web based project using version control system **Maven**.
- Here you will learn how to create/build/deploy and run a web application:

87

87

Create Web Application

- To create a simple java web application, we'll use maven-archetype-webapp plugin. So let's open command console, and execute the following **mvn** command.

```
>mvn archetype:generate -DgroupId=com.rns.simpleweb -DartifactId=webapp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

- Maven will start processing and will create the complete web based java application project structure.

88

88

Create Web Application

- Maven uses a standard directory layout. Using above example, we can understand following key concepts.
- Webapp contains src folder and pom.xml
 - src/main/webapp contains index.jsp and WEB-INF folder
 - src/main/webapp/WEB-INF contains web.xml
 - src/main/resources it contains images/properties files .

89

89

Build Web Application

- Let's open command console, go the project directory and execute the following **mvn** command.
 - **>mvn clean package**
- Deploy Web Application
 - Now copy the **webapp.war** created in **C:\ > MVN > webapp > target >** folder to your webserver webapp directory and restart the webserver.
- Test Web Application
 - Run the web-application using URL : **http://<server-name>:<port-number>/webapp/index.jsp**
- Verify the output.

90

Tomcat App Server

Rise 'n' Shine Technologies

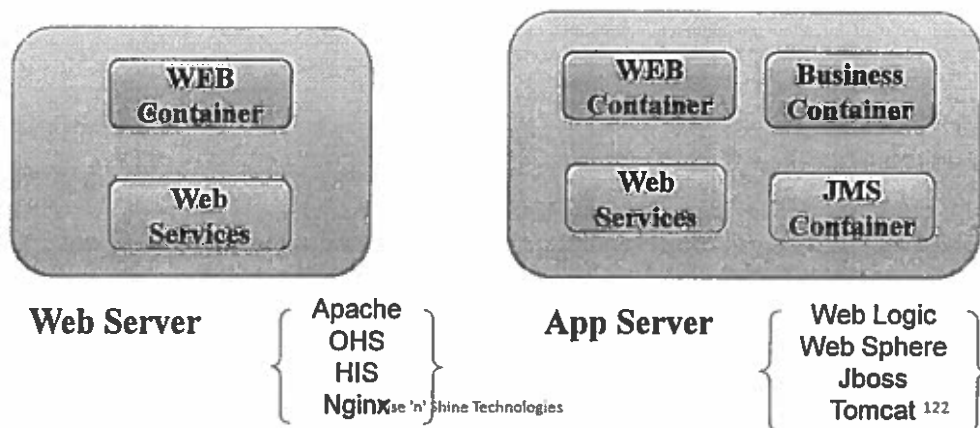
121

121

Types of Servers

1) Web Servers

2) Application Servers



122

Web Server	Application Server
A Web server handles the HTTP protocol. Protocol Dependent (Http)	Application Server handles any number of protocols (HTTP, TCP-IP, RMI (t3) ...etc). Protocol Independent.
Web Server is mostly designed to serve static content .	The application server is used to run business logic or dynamically generating presentation code.
It Serves Web Based Applications .	It serves Web Based Applications and Enterprise Based applications .
It provides only Web container (Servlet container + JSP container)	It provides Web Container + EJB Container + JMS Container
Apache, OHS, HIS, IIS, Sun one iplanet	Weblogic, Websphere, Oracle Application Server, JBOSS, GF

About Tomcat Server:

- --> Open Source
- --> Platform Independent
- --> Apache vendor
- --> Pre-requisite: Any OS is recommended
- --> 1 GB RAM

JDK Installation:

- To Install tomcat, first we need to install jdk. So To install java (jdk-7) follow the below steps :
- Step 1: Download the tar file of java
- Step 2: Now unpack the zip file in /u01/java using tar command


```
– > tar xzpfv jdk-7u9-linux-x64.tar.gz -C /u01/java/
```
- This will create the directory /u01/java/jdk1.7.0_09 . This will be our JAVA_HOME

Rise 'n' Shine Technologies

125

125

JDK Installation:

- Step 3: Now we set the Java Home and will put Java into the path of our users using below command


```
– > JAVA_HOME=/usr/java/jdk1.7.0_09/
```

```
– > export JAVA_HOME
```

```
– > PATH=$JAVA_HOME/bin:$PATH
```

```
– > export PATH
```
- To set the JAVA_HOME permanently, we need to add above commands to the ~/.bash profile of the user

Rise 'n' Shine Technologies

126

126

Tomcat Installation:

- Install Tomcat 8/9.
- Step 1: First download the zipped file using the below url
“<http://tomcat.apache.org/>”
- Step 2: We will install tomcat under /u01/tomcat directory , Now unzip the tomcat file using below command:
- `> unzip /u01/batch/apache-tomcat-8.0.29.zip`
- This will create a directory “/u01/batch/apache-tomcat-8.0.29”
- Then rename the this directory to tomcat using ‘mv’ command.

127

Tomcat Directory Structure

- CATALINA_HOME = /u01/batch/tomcat
 - webapps (This Application deployment directory)
 - bin (Start and Shutdown scripts of Tomcat)
 - Startup.sh
 - Shutdown.sh
 - logs
 - catalina.out (Server core log file)
 - localhost.dt.log (Access information of the server)
 - Conf
 - server.xml
 - tomcat-users.xml
 - lib (Library / Module files of Tomcat)
 - temp
 - work

Rise 'n' Shine Technologies

128

128

Tomcat Start/Shutdown Server

- Startup the Server
 - > go to bin and run the startup.sh
 - > sh ./startup.sh
 - > ps -aef | grep tom
 - > Access the url --> <http://ip:8080>
- Shutdown Tomcat Server:
 - go to bin and run the
 - ./shutdown.sh

Rise 'n' Shine Technologies

129

129

Configuration Files:

- **conf directory:**
 - - tomcat-users.xml
 - - server.xml (This is the complete configuration file)
 - - port customization (connector Element)
 - - open ports in the linux (netstat -nl | grep 8080)
 - - appBase attribute
 - (which can be used to customize the webapps directory)
 - -SSL configuration port should be enabled
 - Default https port – 8443
- **logs:**
 - -- tail -f catalina.out
 - -- localhost_access_log

Rise 'n' Shine Technologies

130

130

Types of Deployments

- Deployment Operation: Web Application:(.war file)
- Two types of Deployments:
 - 1. Console / Tomcat manager Deployment
 - -> On the home page of tomcat click on Tomcat manager line
 - -> Go to war deployment section
 - -> Browse your war file
 - 2. Direct / Hot Deployment
 - -> This is recommended when the large size of applications.
 - -> May be restart required.

Rise 'n' Shine Technologies

131

131

tmp Dir / Cache Dir

- tmp dir: this is extracted directory of the Tomcat web Apps Apps
- work dir: cache dir
- Clean and Start the Tomcat incase if the server throws any exception like 'lock exception' while starting server
- -- delete the content of the tmp and work directory and restart the server.
-
- -- rm -rf tmp/*
- -- rm -rf work/*

Rise 'n' Shine Technologies

132

132

Version Controlling Tool - Git

Rise 'n' Shine Technologies

133

133

Version Control

- Without Version Control
- What is version control
- Why use one

Rise 'n' Shine Technologies

134

134

Without Version Controlling System

- 100 developers...
- File Sharing:
 - No History
 - No Security
 - No Tracking
 - Maintenance Issues
 - Disk Space

Rise 'n' Shine Technologies

135

135

What is version control

- A repository of files with monitored access to keep track of who and what changes were made to files
 - Version tracking
 - Collaboration and sharing files
 - Historical information
 - Retrieve past versions
 - Manage branches
- Version Controlling Tools:
 - CVS, SVN, GIT, Perforce, Mercurial

Rise 'n' Shine Technologies

136

136

Why use one

- In code development, a version control system is, at this point, almost mandatory
 - Multiple developers working on more than one project
 - Must be able to roll back to any version
 - Must be able to tag releases, and milestones

Rise 'n' Shine Technologies

137

137

Tool Types

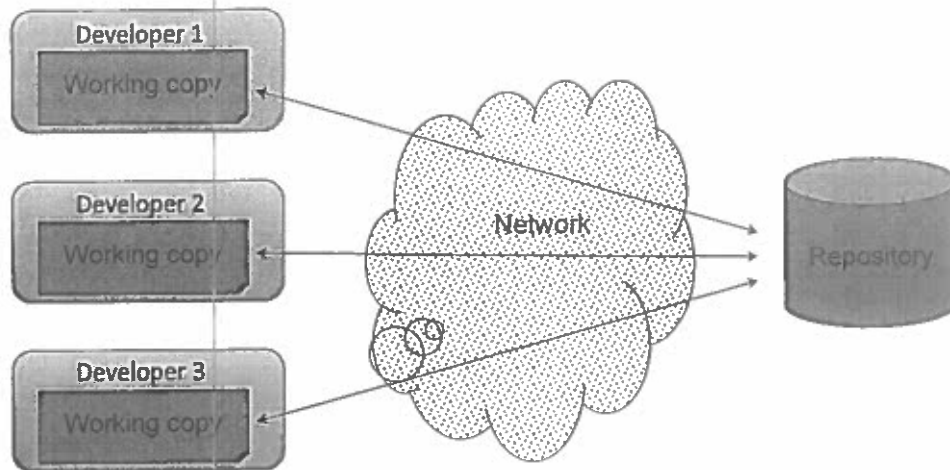
- Server Based Tools
 - CVS
 - SVN
 - Perforce
- Distributed VCS Tools
 - Git
 - Mercurial

Rise 'n' Shine Technologies

138

138

Client-Server Architecture

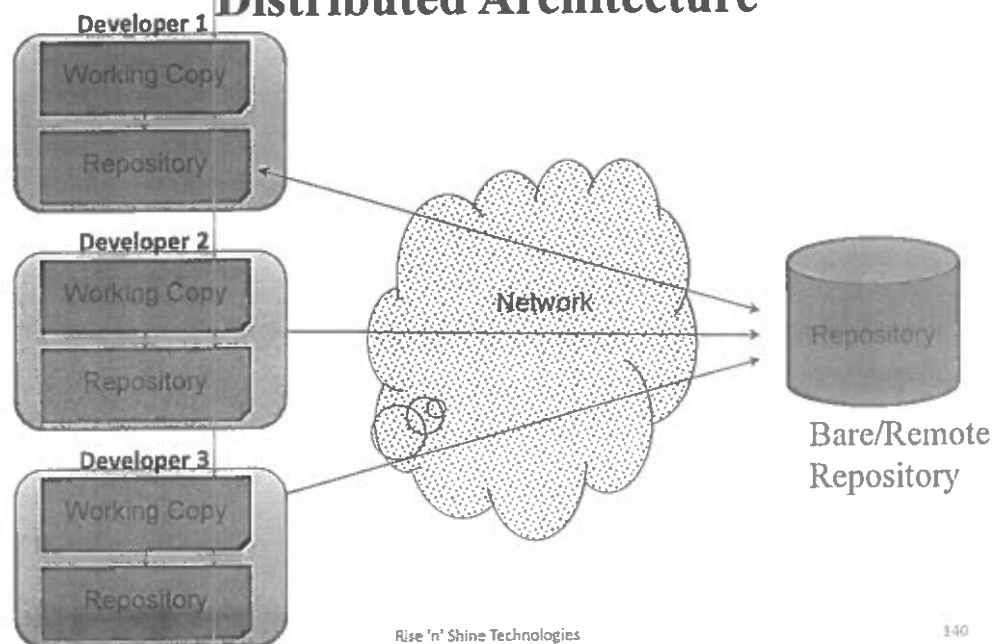


Rise 'n' Shine Technologies

139

139

Distributed Architecture



Rise 'n' Shine Technologies

140

140

Key Concepts

- The **repository** is where files are stored under Git/SVN on the server.
- The contents of a project in the repository are usually copied into a local directory. This is known as the **working directory**, and is separate from the repository.
- The operation of copying the project to the local directory is called "**check out / Pull** " and the reverse is "**check in / Push**".
- The connection to a Subversion/Git repository is usually given as a **URL**
- **Authentication** is how you prove who you are, such as through a username and password.
- **Authorization** deals with what you have access to and the permissions on what you can do. In this context, authorization is about your file read and write permissions.

Rise 'n' Shine Technologies

141

141

Repository versus Working Copy

- Project code is stored in a server in a data store referred to as a "repository."
- Developers "check out / Pull" copies of the project code into their local environments. These copies are referred to as "working copies."
- After making changes to a working copy, the developer "commits" changes to the repository.
- Other developers get these changes by "updating" their working copies.

Rise 'n' Shine Technologies

142

142

GIT - Overview

- Repository:
 - Git helps to create repositories
- Versions:
 - Different versions of same artifact can be stored
- Artifact:
 - Helps to manage changes in artifact
- Comparison:
 - GIT enables comparison of different versions of same artifact
- Collaboration:
 - Promotes Collaboration among developers

Rise 'n' Shine Technologies

143

143

Introduction - GIT

- It was born on 2005
- It is a Open Source Tool
- Platform Independent
- Distributed Version Controlling System
- Installation Windows/Linux/unix

Rise 'n' Shine Technologies

144

144

Git – Setup and Installation

- **Installing Git on Windows:**
 - Download git for Windows from git-scm.com and install it (Git Bash with default options)
- **Installing GIT on Unix:**
 - Go to git-scm.com/download and then select 'Linux' option then follow the instructions...
- **Verify Git Installation:**
 - which git
 - git version

Rise 'n' Shine Technologies

145

145

Git – Client Machine

- Client Machine (Developers Machine)
- -- ssh access to git repository from developers Machine
- -- Windows Client
 - Install the git-bash client software.
- -> Unix Client:
 - Install the git software which is installed in the Server Machine

Rise 'n' Shine Technologies

146

146

How to use GIT Help

- >> git help
- >> git help -a (List of all the commands)
- >> git help -g (Git Guides)
- Ex:
 - >> git help glossary
 - >> git help everyday
 - >> git help init (particular Command help)
- Note: If it is the windows we get help info in the browser and if it is Linux system then in the command prompt we get help info.

Rise 'n' Shine Technologies

147

147

Git Settings - Author and Email

- **To check the configuration list**
 - git config --global --list
- **To set the User Name**
 - git config --global user.name "dev1"
- **To set the User Email Id**
 - git config --global user.email "dev1@gmail.com"
 - git config --global --list
- **Where this configuration info is stored?**
 - \$USER_HOME/.gitconfig file.
 - vi ~/.gitconfig

Rise 'n' Shine Technologies

148

148

Creating Repositories

- Local Repository
 - From Scratch
 - Existing Project
 - Cloning from Remote Repositories
- Bare/Remote Repository
 - GIT Server
 - Git Hub

Rise 'n' Shine Technologies

149

149

Setting up Git repository

- 3 Ways of setting up Git Repository
- 1) From Scratch
 - create a repository from absolutely blank state
- 2) Existing Project
 - Convert an existing unversioned project to a Git Repo
- 3) By Copying from Remote Repository
 - Copy an existing Git repository from Git Server / Git HUB

Rise 'n' Shine Technologies

150

150

1 – From Scratch

- Initializing an Empty Repository

- Go to directory where the repository should be created
- > mkdir Git_Repo_From_Scratch
- > cd Git_Repo_From_Scratch
- > git init

Note: The above command creates a set of files and metadata to store the users data as a repository

Rise 'n' Shine Technologies

151

151

2 – Existing Project

- **Unversioned Project to Versioned:**
- → Create a project using mvn archetype
- mvn archetype:generate -DgroupId=com.rns.simpleweb -DartifactId=webapp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
- --> Go to project directory which is 'unversioned'.
- --> Then run the 'git init' command which is going to create .git dir and this dir is called as versioned dir.

Rise 'n' Shine Technologies

152

152

Creating Bare/Remote Repository on GitServer

- -- Login to Server
- – Go to directory where the repository should be created
- –> mkdir Git_Repo
- –> cd Git_Repo
- –> git init –bare
- Note: The above command creates a set of files and metadata to store the users data as a repository

Rise 'n' Shine Technologies

153

153

3(a) - Cloning Repository from GitServer

- -> Cloning Repository in the client Machine whether that is Win/Unix same process
- --> Create a directory (mkdir git_practise)
- --> git clone root@ip_address:/u01/batch/gitrepo
- --> cd gitrepo
- --> ls -la (which lists all the repo Meta data)

Rise 'n' Shine Technologies

154

154

What is GitHub

- **Web Based**
 - It is a Git Repository with a web Interface
- **Origin**
 - GitHub was born on 2008
- **Subscription**
 - It is having the both free and Paid plans
- **User**
 - Need to create a User account
- **Other Features**
 - Documentation and Bug Tracking
- **Largest Host**
 - It is the largest hosted Git Repository

Rise 'n' Shine Technologies

155

155

What is GitHub

- → Create a GitHub user account
- → Create a Repository in the GitHub
- ***What is fork and How to do it?***
 - Creating project from another existing project
 - It encourages outside contribution

Exercise:

-- fork the sample github project

Rise 'n' Shine Technologies

156

156

3(b) - Cloning Repository from GitHub

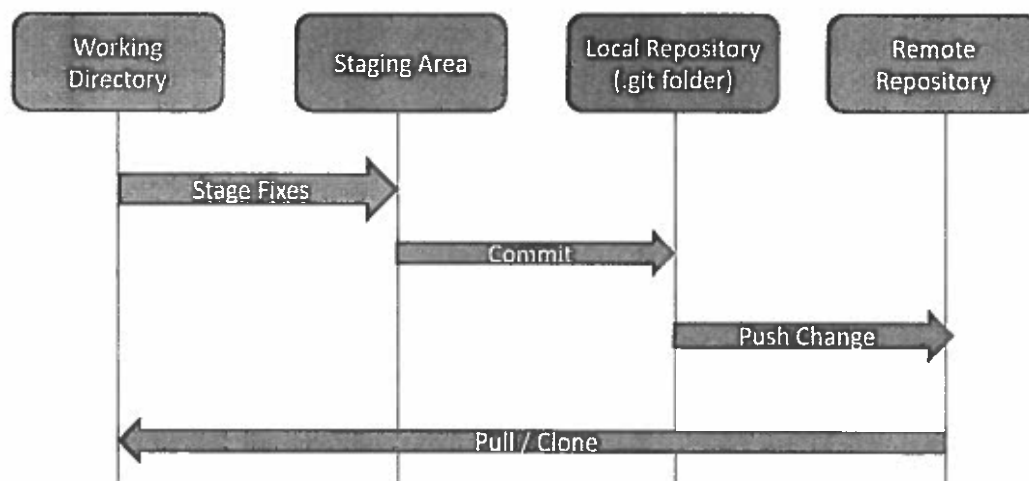
- -> Cloning Repository in the client Machine whether that is Win/Unix same process
- --> Create a directory (mkdir git_practise)
- --> git clone https://github.com/username/repo.git
- --> cd repo_dir
- --> ls -la (which lists all the repo Meta data)

Rise 'n' Shine Technologies

157

157

How GIT Works



Rise 'n' Shine Technologies

158

158

How GIT Works

- 3 main states of artifact/file
- Modified:
 - Here, the artifact/file goes through change made by the user
- Staged
 - Here, the user / developer adds the artifact/file to Git Index or staging area
- Committed
 - Here, the artifact/file gets safely stored in git database

Rise 'n' Shine Technologies

159

159

Git - Adding the Files

- **Process to add the files to Remote Repository:**
 - --> 1. Create the file
 - --> 2. Stage the File
 - --> 3. Commit the file
 - --> 4. Push to remote Repository
- **Commands:**
 - --> touch firstFile.txt
 - --> git status (It gives working tree status)
 - --> git add firstFile.txt (It stages the file)
 - --> git status
 - --> git commit firstFile.txt -m "This is my First commit"
 - --> git push origin master
 - --> git pull origin master

Rise 'n' Shine Technologies

160

160

Git Commands

- Revert Back the Staged File:
`> git rm --cached fileName`
- Staging and commit:
`> git commit -am "msg"`
- Deleting the file:
`> git rm file_name`
- View which branch you are in:
`> git branch`
- Information about each line modification done by author
`> git blame filename`
- Fetch the changes from Remote Repository:
`> git fetch origin master`
- Note: The git pull command performs a git fetch and git merge and the git fetch does not perform merge operations and it just fetches

161

161

Git - Parallel Development

- **Dev1 User:** (creation -> stage -> commit -> push)
`git clone dev1@IP_Address:repo`
`cd repo`
`touch hello.txt`
`git status -> gives you the status of ur working tree -> added/new created`
`git add hello.txt`
`git commit Hello.txt -m "my first commit"`
`git log -> shows the log of commits performed as of now`
- **Dev2:**
`cloned the repo`
`added some text to Hello.txt`
`git add Hello.txt`
`git commit -m "second commit"`
`git push`

Rise 'n' Shine Technologies

162

162

Git - Parallel Development Strategy

- **Dev1 User:**
 - >> vi hello.txt
 - >> git status -> gives you the status of ur working tree
 - >> git add hello.txt
 - >> git commit Hello.txt -m "my first commit"
 - >> git log --oneline
 - >> git push
- --> leads to conflicts to the files

Rise 'n' Shine Technologies

163

163

Merging the conflicts

- git mergetool
- enter
- open ups merge window --> select respective changes then save and quit.
- git add .
- git commit -m "After merging"
- git branch (make sure you are in master branch)
- git push origin master --> push the merged changes

Rise 'n' Shine Technologies

164

164

Git Status and Log Commands

- **Checking Repository Status:**
 - --> git status
 - --> git status --long
 - --> git status -s
 - **Checking Committed History:**
 - --> git log
 - --> git log --oneline
 - --> git log filename
 - --> git log --author author_name
 - --> git shortlog
 - --> git log -n 3 --oneline (Last 3 commits)
 - --> git log <since>..<<until>
- Ex: git log checksum1..Checksum2 --oneline*

165

165

GIT - Directory Structure

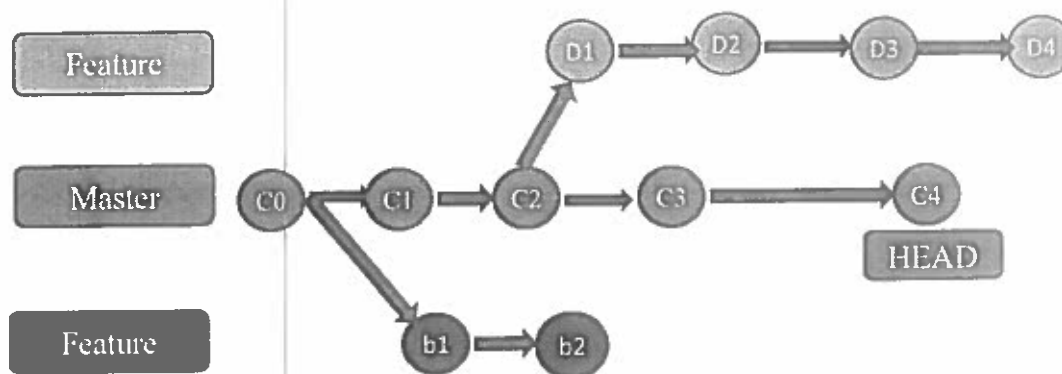
- Master → This is root directory and default one
- Branches → It contains the Major releases
- tags --> It contains References of Bug fixes code and always read only
- Note:
 - We create a branch when we want to develop a new feature or to do a bug fix
 - Unstable code is never committed to main or master branch

Rise 'n' Shine Technologies

166

166

Regular Git Branching



Rise 'n' Shine Technologies

167

167

Real World Branching Scenario

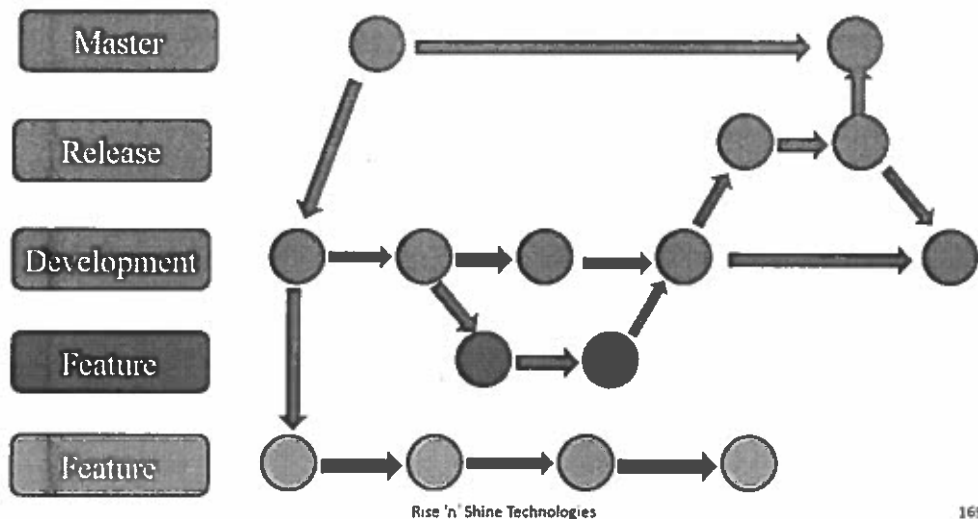
- **Master Branch:** Production ready copy
- **Development Branch:** It is a developers branch where continuous work will be done
- **Release Branch:** This branch is created from the Development Branch to make it ready for the Release and it is used for Bug tracking and documentation purpose.
- **Feature Branch:** Whenever the developers working on the new features, they use the feature branches and commit the code to that branch.

Rise 'n' Shine Technologies

163

168

Real World Branching Scenario



169

169

Git Branching

- -- Default branch is master
- -- How do we know branch name
 - >> git branch
- -- Create a new branch from master
 - >> git branch new-feature-1 (branch will be created from master)
 - >> git checkout new-feature-1 (to switch to new-feature-1 Branch)
- -- Create a branch and switch to it immediately
 - >> git checkout -b new-feature-2
- -- list all the branches including remote branches
 - >> git branch -a

Rise 'n' Shine Technologies

170

170

Git Branching

- -- Display info of Branches
 - >> git branch -v
- -- Display remote Branches
 - >> git branch -r
- -- Renaming a Branch
 - >> git branch -m new-feature-1 small-feature
- -- Perform some changes on branch and push it with below cmd
 - >> git push origin new-feature-1
- -- To pull the branch changes by other user
 - > git fetch origin new-feature-1 new-feature-1:new-feature-1

Rise 'n' Shine Technologies

171

171

Git Branching

- -- Delete a branch in local Repository
 - >> git branch -d new-feature-2
- Note: (Sometimes It throws error because new-feature-2 branch is not fully merged. If you would like to delete this branch forcefully then use '-D' option)
 - >> git branch -D new-feature-2
- -- Delete same in remote repo
 - > git push origin :refs/heads/new-feature-2

Rise 'n' Shine Technologies

172

172

Task on the Branches

- Go to the master branch and perform some commits.
- Then verify the all commits using git log command
- Then create a new branch based on master branch
 - >> git checkout -b new-branch-1 master
 - >> git log --oneline
- Then do some modifications in the new-branch-1 and then commit to this branch
- Then checkout to the master branch and verify the branch history using git log command
- Create a new branch 'new-branch-2' based on the 'new-branch-1'
 - >> git checkout -b new-branch-2 new-branch-1
- Verify the logs

Rise 'n' Shine Technologies

173

173

Git -- Tags

- -- In git tags are read only
- -- Snapshot of a particular revision number having a name and meant for read only
- -- Create a light weight tag
 - > git tag tag_1.0
- -- Show git tag revision
 - > git show tag_1.0
- -- List all tags
 - > git tag

Rise 'n' Shine Technologies

174

174

Git -- Tags

- -- Create an annotated tag with some message
 - > git tag tag_1.1 -m "Release 1.0 completed"
- -- Checkout a tag
 - > git checkout tag_1.0
- -- push the tag using below command as it will not be saved in server with default push
 - > git push origin tag_1.1
- -- Delete a tag
 - > git tag -d tag_1.0
 -
- -- Delete same in remote repo
 - > git push origin :refs/tags/tag_1.0

175

175

Directory & Branching Structure

- Releases:

|R1-----| |R2-----| |R3-----| ...R20

- Objectives:
 - 1. No data loss in production
- Branching Strategies:
 - 1. Sequential Branching Strategy
 - 2. Parallel Branching Strategy

Rise 'n' Shine Technologies

176

176

Sequential Branching Strategy - Branches

- Releases:
 - |R1-----||R2-----||R3-----| ...R20
- **master** --> Master Copy (Production Copy)
- **branches/** R1_release_branch
- branches/R2_release_branch
- branches/R3_release_branch
-
-
- branches/R20_release_branch

Rise 'n' Shine Technologies

177

177

Sequential Branching Strategy - Tags

-- Total 6 QA Builds within Release 1

R1 |---1B---2B---3B---4B---5B---6B| R2

-- Dev needs 3 Build QA src from Release 1, then how?

For each build, master code will be copied to the Tags

R1_QA1_tag
 R1_QA2_tag
 R1_QA3_tag

 R2_QA1_tag
 R2_QA2_tag

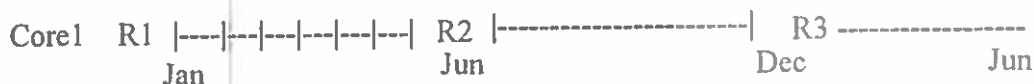
Rise 'n' Shine Technologies

178

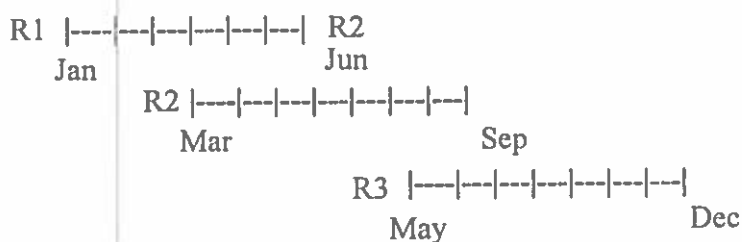
178

2. Parallel Branching Strategy

Sequential Branching Disadvantages



Parallel Branching Advantages



Rise 'n' Shine Technologies

179

179

2. Parallel Branching Strategy

- **Objectives:**
 - 1. No data loss in production
 - 2. Branch is the dev code where check in and check out
- **Before Deploying/Releasing:**
 - No data Loss / Overwriting code in prod environment
- **After Deploying/Releasing:**
 - Make your master = Prod Code base (Release branch)

Rise 'n' Shine Technologies

180

180

2. Parallel Branching Strategy

Objectives:

→ No Data Loss in Production Env

→ Master always should be equivalent to Prod Env

	Master		Prod
(Jan)	Core		Core
(Jun)	Core +R1		Core +R1
(Sep)	Core +R1 +R2		Core +R1 +R2
(Dec)	Core +R1 +R2 +R3		Core +R1 +R2 +R3

Rise 'n' Shine Technologies

181

181

2. Parallel Branching Strategy

- **Jan R1:** (Master and core should be same)
 - /master/Core
 - /branches/R1/Core
 - Master > git branch R1
- **Mar R2:** (master and core should be same)
 - /master/Core
 - /branches/R2/Core
 - Master > git branch R2
- **May R3:** (master and core should be same)
 - /master/Core
 - /branches/R3/Core
 - Master > git branch R3

Rise 'n' Shine Technologies

182

182

2. Parallel Branching Strategy

Jun: (R1 Release)

	Master		Branch R1
(Jan)	Core		Core
(Jun)	Core +R1		Core +R1

- Release Objective:**

– /master/Core+R1

Merging: Src → Dest
 git Checkout Destination
 Dest> git merge SRC

Merging: R1 → master
 > git checkout master
 > master>git merge R1

Rise 'n' Shine Technologies

183

183

2. Parallel Branching Strategy

Sep: (R2 Release)

	Master		Branch R2
(Jan)	Core		Not Yet Started
(Mar)	Core		Core +R2
(Sep)	Core +R1 +R2		Core +R2 +R1

Release Goal:

/master/Core + R1+R2

/branches/R2/Core + R1 + R2

Rise 'n' Shine Technologies

184

184

2. Parallel Branching Strategy

- **Step 1:** /branches/R2/Core+R1+R2
- Merge: master --> R2
- src: master
- dest: R2
- Git checkout R2
- R2> git merge master

- **Step 2:** /master/Core + R1+R2
- Merge-> R2-> master
- src: R2
- dest: master
- git checkout master
- master> git merge R2

Rise 'n' Shine Technologies

185

185

2. Parallel Branching Strategy

- **Dec: (R3 release)**

	master		Build R3
(Jan)	Core		No Branch
(May)	Core		Core +R3
(Dec)	Core +R1 +R2 +R3		Core +R3 +R1 +R2

Goal:

/master/Core + R1+R2+R3

/branches/R3/Core+R1+R2+R3

Rise 'n' Shine Technologies

186

186

2. Parallel Branching Strategy

- Step 1: /branches/R3/Core+R1+R2+R3
- Merge: master --> R3
- src: master
- dest: R3
- git checkout R3
- R3> git merge master

Step 2: /master/Core + R1+R2+R3

- Merge-> R3-> master
- src: R3
- dest: master
- git checkout master
- master> git merge R3

Rise 'n' Shine Technologies

187

187

Parallel Branching Exercises:

- Creating HelloWorld.java, src/Example.java
- Add and commit these changes to the Repository.
- Creating a Branch:
 - > git branch Release1
 - > git branch (view branches)
 - > git checkout Release1 (Switch to another branch)
 - > git branch

Rise 'n' Shine Technologies

188

188

Parallel Branching Exercises:

- Step 2: Modify the Helloworld.java and commit to the Release 1.
- > execute the required commands
- > git status
- > ls
- > git branch (it should be Release1)
- > git checkout master (Change to master repository)
- > cat HelloWorld.java (No changes in master)
- Step3: Create a new branch Release2 from the master and update the code to HelloWorld.java and commit the changes to the Release 2.
- -- View the git log messages (which will be displayed only that particular Release log)

Rise 'n' Shine Technologies

189

189

Git tags -- Exercise

- 1) Create a tag
- 2) After creating a tag modify some changes
src/Example.java and commit the changes to the Repository.
- 3) Then change back the Created tag and we will be having
previous changes but not latest changes.

Rise 'n' Shine Technologies

190

190

Important Commands

- -- To check the diff between staging area and git
- > git diff
- -- Diff between branches
- > git diff Release1 Release2
-
- -- Show changes of patchset
- > git show charset
- -- Remove untracked files (before add operation)
- -- The below command shows what are all the files will be deleted
- > git clean -n
- > git clean -f (delete them actually)

191

191

Important Commands

- -- Revert the unstaged changes with below command
- > git checkout filename
-

192

Git Stashing

- --stash stores the staged(git add) changes into a package and stores in memory, when ever u want to pull remote changes and want to apply the unstaged changes you can do this.
- --> git stash
- -- list the stashes
- git stash list
- -- Apply the stash when you wanted with below command
- git stash pop
- or you can use the id as well as shown below [pop is always latest stash]
- git stash apply stash@{0}

Rise 'n' Shine Technologies

193

193

Git Stashing

- -- This may lead to conflicts, resolve conflicts manually
- -- Once the conflicts are resolved commit the changes
- -- Delete the stash packet
- git stash drop stash@{0}

Rise 'n' Shine Technologies

194

194

Git Cmds - Remote Repositories

- --Default will be origin but we can add multiple repositories as well
- `git remote add reponame username@github.com/project/repo`
- `git remote show origin` (which repository it has been pointed)
- `git remote -v`



Introduction to Vagrant

Rise 'n' Shine Technologies

By
Venkat



We're sorry, but something went wrong.
We've been notified about this issue and we'll take a look at it shortly.



What is Vagrant?

- A tool to build development environments based on virtual machines
- Focused to create environments that are similar as possible or identical with production servers
- Created by Mitchell Hashimoto
- Written in Ruby
- Initially built on top of VirtualBox API, today offers VMWare Fusion support (as \$79 per licence)

What is Vagrant?

Let Vagrant handle the entire lifecycle of development machines

- ▶ Suspend, Halt, Resume virtual machines
- ▶ Destroy your virtual machines and delete its metadata
- ▶ SSH in to your machine
- ▶ Package up your machines entire state and the re distribute it to other development team members
- ▶ Vagrant is a must have tool in development environments in which it will resemble your actual production

Why should we use Vagrant?

- ▶ Traditionally we would have to manually install and setup things like, Apache and MySQL locally on our development workstations.
- ▶ This can be very complex with all amount of moving parts to keep track of
- ▶ Human error is easy to take place when setting up complete local development environments
- ▶ Vagrant is an awesome way of fixing these issues by automating our setup of all these services needed to develop our applications. Each project gets its own virtual machines.

Using Vagrant to Deploy

- ▶ Developers can now check out versions or repositories from a version control such as GIT and run 'vagrant up'.
- ▶ Now the developers can work on their own machines comfortably with their own, editors, browsers and tools.
- ▶ We have a consistent and stable development environment now
- ▶ The same scripts of configuration in using vagrant to deploy dev environments is the same used in production therefore the dev environment is as close as possible to the production limiting issues later on.

Prerequisites to Install Vagrant

➤ Tools Required:

➤ VirtualBox

➤ virtualization software

➤ Vagrant

➤ virtualization automation

➤ Steps:

➤ 1. Download VirtualBox first

➤ 2. Download Vagrant installer on Vagrant site (Debian, CentOS, Windows, OSX, other OS's)

➤ 3. Get a Vagrant box

Prerequisites to Install Vagrant

- You can actually use Vagrant with other “providers” instead of just VirtualBox
 - VMware
 - AWS EC2
 - And just about any other provider out there!
- Note: In this course we are going to be primarily using VirtualBox because it's free.
- *Vagrant offer an official VMware provider for cost
www.virtualbox.com/vmware



- Virtualization software from Oracle
- Free
- Runs on Windows, Mac, Linux
- Runs as an application
- Allows us to use local VMs
- Easy to install
- Works well with Vagrant

➤ <https://www.virtualbox.org/>



- Virtualization workflow software from HashiCorp
- Free, open-source
- Runs on Windows, Mac, Linux
- Easy to install
- Works well with Puppet, Chef, Shell
 - many other **provisioners**
- Works well with VirtualBox, VMware, Amazon Web Services
 - many other **providers**

➤ <https://www.vagrantup.com/>

Rise 'n' Shine Technologies

What is a Vagrant Box?

- Vagrant boxes are just 'Templates'
- Is a previously built Vagrant virtual machine image,
- ready-to-run
- Available in a lot of platforms (Linux, Windows)
- Manage Boxes such as:
 - Vagrant box list
 - Vagrant box add
 - Vagrant box remove


How to add a box?

➤ Great box repository:

➤ www.vagrantbox.es

➤ Run this command:

```
$ vagrant --version  
$ vagrant box add <name> <url> <provider> # virtualbox
```

- 
- To create a VM:
 - Add the Vagrant box
 - `$ vagrant box add hashicorp/precise64`
 - downloads a “base box”
 - boxes at <https://atlas.hashicorp.com/search>
 - Inside your project, create a Vagrantfile:
 - `$ vagrant init hashicorp/precise64`
 - builds a Vagrant file with the base box



➤ Vagrantfile

- lots of comments by default

- stock Vagrantfile without comments is:

- `Vagrant.configure(2) do |config| config.vm.box = "hashicorp/precise64"`

- End

- As of date the current config version of vagrant is "2"

- There are currently only 2 versions supported. "1" and "2"

- Version "1" config files for Vagrant version 1.0.X

- Version "2" config files for Vagrant version 1.1+ leading up to 2.0.x

Access Vagrant Box

- To access a VM:
 - vagrant ssh
 - connects to the VM via the forwarded SSH port
- requires an SSH client installed
 - Git (<https://msysgit.github.io/>)
 - openssh on Cygwin (<http://www.cygwin.com/>)
 - PuTTY (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>)
 - requires converting the key format



- vagrant up
 - starts the VM with the following steps
- imports the base box to VirtualBox
- makes sure the base box is up to date
- sets a unique name for the VM
- sets up networking (just NAT by default)
- sets up port forwarding (just SSH by default)
- boots VM
- replaces known, insecure SSH key with a new random key
- makes sure VirtualBox Guest Additions are installed
- mounts shared folders (/vagrant by default on the VM)
- provisions software (nothing by default)

Rebuild Vagrant Box

- To rebuild a VM:
 - vagrant destroy
 - deletes a VM
 - vagrant up
 - starts the VM

How do start to use it?

Simply run this command:

```
$ vagrant up
```

Easy:

\$ vagrant ssh

\$ vagrant halt

\$ vagrant reload

How to access it

- You need to set forwarding ports between guest and host to work (bind on 0.0.0.0!)
- Just add the following code in your Vagrantfile, restart server and access in browser:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # ...

  config.vm.network :forwarded_port, guest: 3000, host: 3000

  # ...
end
```

How to customize it

- You can change memory, CPU cores and other things in Vagrantfile
- Just see VBoxManage options
- Example:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # ...

  config.vm.provider :virtualbox do |vb|
    vb.customize : 'modifyvm', :id, '--memory', '1024'
    vb.customize : 'modifyvm', :id, '--cpus', '4'
  end

  # ...
end
```

I hate it!

- Of course, no! :)
- It's time to configure environment using available provisioners to install required software:
 - Shell

LAB 12.5

Create a single bash script that installs all you need:

```
#!/bin/bash

apt-get update

# base
apt-get install --yes python nginx mongodb-server redis-server

# others
apt-get install --yes curl tmux htop

(...)

# some additional configuration here

(over)
```

[REDACTED]

Easy:

```
$ vagrant up --provision
```

