Continuous Integration with Jenkins-CI

Labs

Table of Contents

Lab 1 - Jenkins

In this lab you will verify that Jenkins Continuous Integration is already installed and you will configured it. There are different ways Jenkins can be installed and run. If you are running a production installation of Jenkins on a Windows box, it is essential to have it running as a Windows service. This way, Jenkins will automatically start whenever the server reboots, and can be managed using the standard Windows administration tools.

One of the advantages of running Jenkins on an application server such as Tomcat is that it is generally fairly easy to configure these servers to run as a Windows service. However, it is also fairly easy to install Jenkins as a service, without having to install Tomcat.

At the end of this lab you will be able to:

1. Verify Jenkins is running

Part 1 - Configure Jenkins

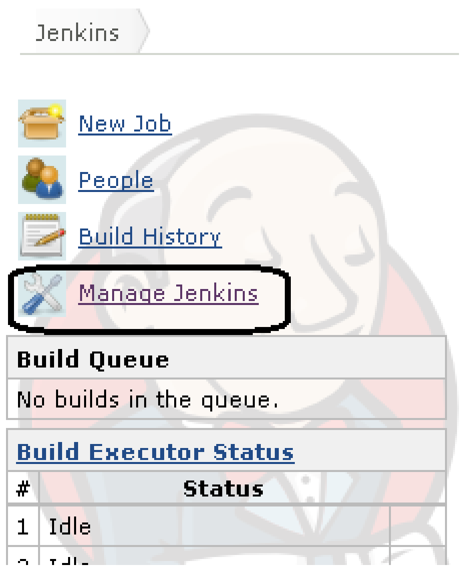
After the Jenkins installation, you can configure few other settings to complete the installation before creating jobs.

You will be setting JDK HOME, Maven Installation directory and SVN plugin.

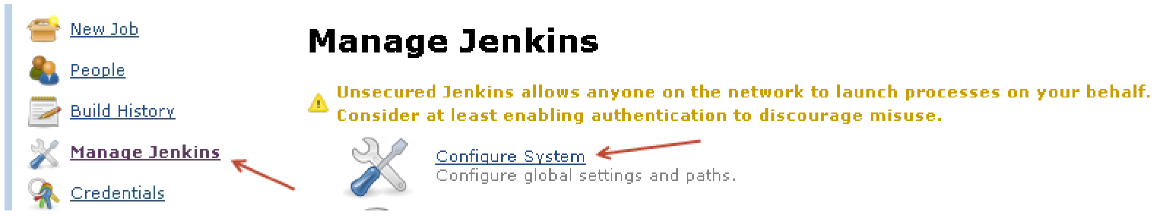
__1. To connect to Jenkins, open Firefox and enter the following URL .

```
http://localhost:8080/
```

__2. Click on the **Manage Jenkins** link.

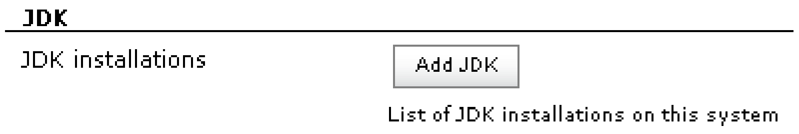


__3. From the menu, click **Manage Jenkins** → **Configure System**.



Unsecured Jenkins allows anyone on the network to launch processes on your behalf. Consider at least enabling authentication to discourage misuse.

__4. Scroll down and find JDK section, Click **Add JDK**.



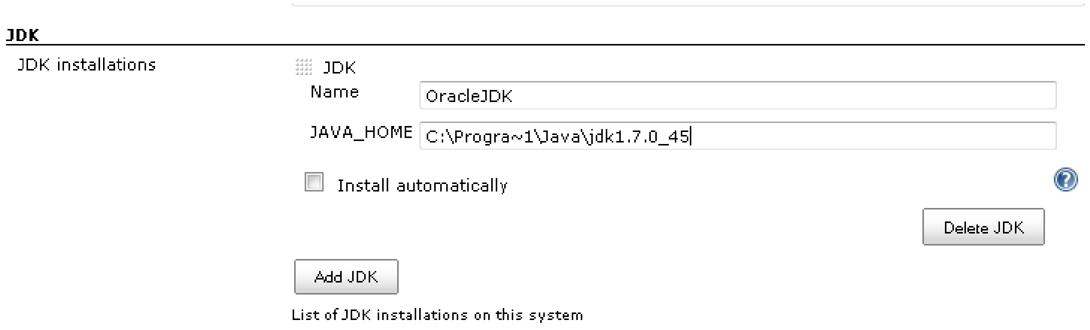
__5. Enter **OracleJDK** for JDK name.

__6. Don't check the 'Install automatically' option. Uncheck if already checked.

__7. Enter JAVA_HOME value as **C:\Progra~1\Java\jdk1.7.0_45**

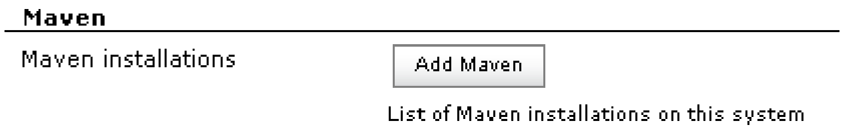
Note. You may need to use another path if Java was installed in a different folder, contact your instructor or search for the right path and use it as JAVA_HOME.

__8. Verify your settings look as below:



Name: OracleJDK
JAVA_HOME: C:\Progra~1\Java\jdk1.7.0_45
☐ Install automatically

__9. In the Maven section, click **Add Maven**.

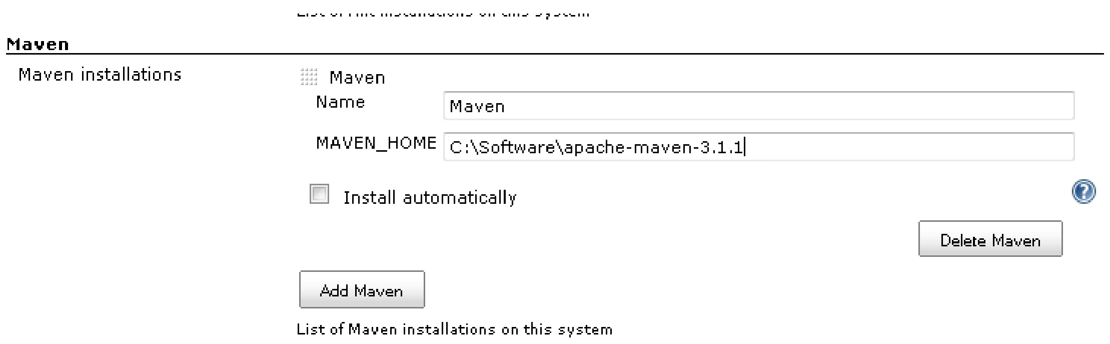


__10. Enter **Maven** for Maven name.

__11. Uncheck the 'Install automatically' option.

__12. Enter **C:\Software\apache-maven-3.1.1** for MAVEN_HOME.

__13. Verify your settings look as below:


Maven

Maven installations

Maven

Name: Maven

MAVEN_HOME: C:\Software\apache-maven-3.1.1

☐ Install automatically

__14. Scroll down and click **Save**.

No special configuration is needed for SVN, because Jenkins uses native Java libraries to interact with Subversion repositories. You will provide the SVN URL to access the source code when you setup jobs.

Part 2 - Review

In this lab you configured the Jenkins Continuous Integration Server.

Lab 2 - Maven

In this lab, we're going to configure Apache Maven. We'll setup the Eclipse installation to use that version of Maven as its implementation.

Although most developer's use of Maven will be through their IDE, it's a good idea to know a little bit of command-line usage. That knowledge will come in handy when you go to setup continuous integration, and also when you want to do a quick build, perhaps for generating deployment or QA artifacts (although that might also be done through continuous integration). Also, it's a good idea to set the Maven implementation, so you have control over the exact Maven version that gets run.

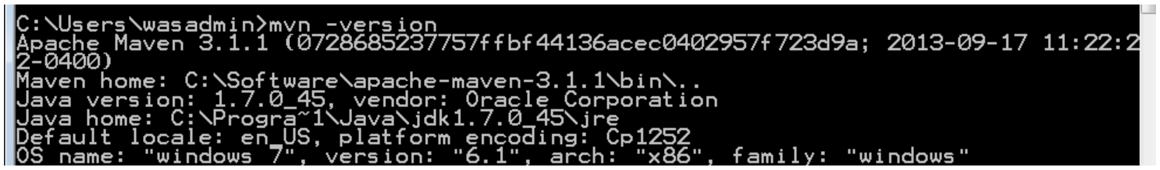
Part 1 - Test Maven from the Command Line

__1. Open a command prompt window (in the start menu, click **All Programs --> Accessories --> DOS Prompt**.

__2. In the command window, type:

```
mvn -version
```

__3. You should see output similar to:



```
C:\Users\wasadmin>mvn -version
Apache Maven 3.1.1 (0728685237757ffbf44136acec0402957f723d9a; 2013-09-17 11:22:2
2-0400)
Maven home: C:\Software\apache-maven-3.1.1\bin\..
Java version: 1.7.0_45, vendor: Oracle Corporation
Java home: C:\Progra~1\Java\jdk1.7.0_45\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "x86", family: "windows"
```

We have Maven available from the command line.

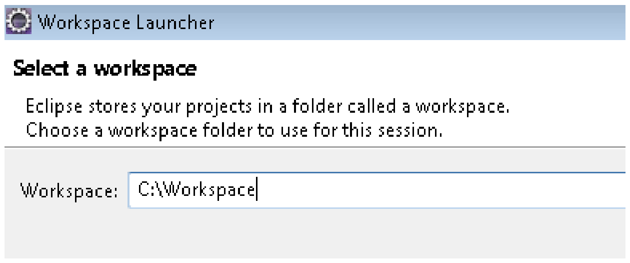
Part 2 - Setup Maven in Eclipse

We have Maven already installed and reachable from the command line, but in reality, we are probably going to use Eclipse as the primary interface to it. So there's a little configuration to do.

__1. Use Windows Explorer to navigate to **C:\Software\eclipse**, and then double-click on '**eclipse.exe**' to start up the IDE.

__2. You may see a security dialog. If so, un-check the box for "Always ask before opening this file" and then click **Run**.

__3. Change the workspace to **C:\Workspace** and click **OK**.



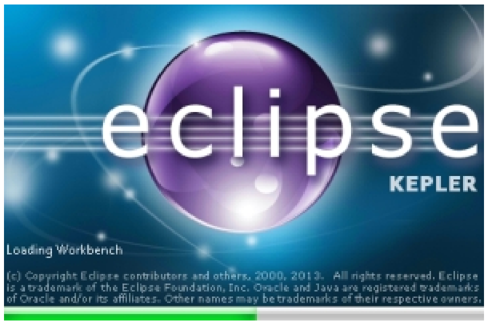
Workspace Launcher

Select a workspace

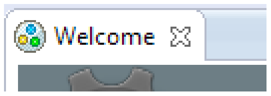
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Workspace

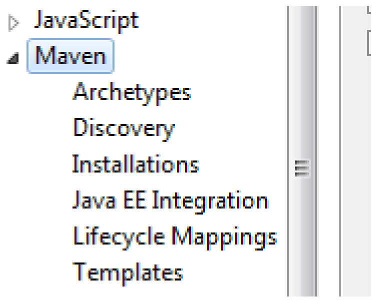
__4. Eclipse Kepler will start launching.


eclipse
KEPLER
Loading Workbench
(c) Copyright Eclipse contributors and others, 2000, 2013. All rights reserved. Eclipse is a trademark of the Eclipse Foundation, Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

__5. Close the **Welcome** screen by clicking the 'X' next to **Welcome** on the tab.

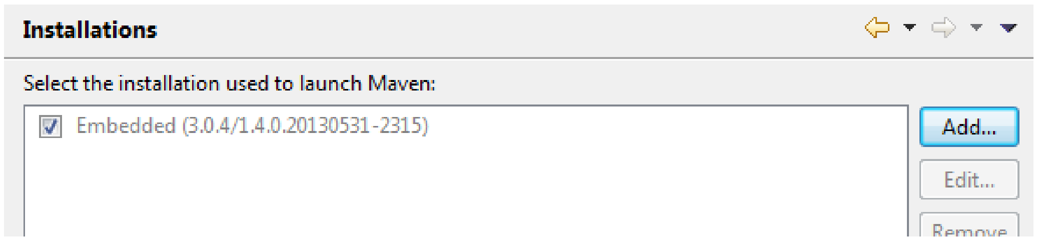


__6. From the main menu, select **Window --> Preferences** and then expand the tree node for **Maven**.



__7. Click on the tree node for **Installations**.

__8. On the right-hand side of the **Preferences** window, click on the **Add...** button.



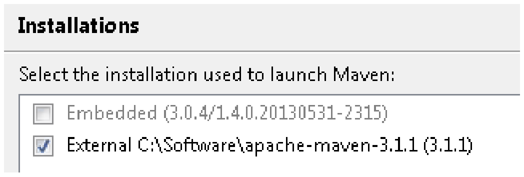
Installations

Select the installation used to launch Maven:

☑ Embedded (3.0.4/1.4.0.20130531-2315)

__9. Navigate to **C:\Software\apache-maven-3.1.1**, ensure the node is selected, and click **OK**.

__10. In the **Preferences** dialog, verify Maven path and click **OK**.



Installations

Select the installation used to launch Maven:

- [ ] Embedded (3.0.4/1.4.0.20130531-2315)
- [✓] External C:\Software\apache-maven-3.1.1 (3.1.1)

That's it! We're now setup to use Maven in the IDE.

Part 3 - Review

In this lab, we verified Apache Maven was already unpacked, then set up the environment variables that will be required to execute Maven. Once we had the command-line setup complete, we started Eclipse and set up Eclipse to use the same installation of Maven that we have on the command line.

Although there is a version of Maven embedded in Eclipse, it is not generally the latest version released, so it's usually a good idea to install Maven separately and then configure Eclipse to use that version.

Lab 3 - Create a Jenkins Job

In this lab you will create and build a job in Jenkins.

Jenkins supports several different types of build jobs. The two most commonly-used are the freestyle builds and the Maven 2/3 builds. The freestyle projects allow you to configure just about any sort of build job, they are highly flexible and very configurable. The Maven 2/3 builds understand the Maven project structure, and can use this to let you set up Maven build jobs with less effort and a few extra features.

At the end of this lab you will be able to:

1. Create a Jenkins Job without repository
2. Set-up Subversion
3. Create a Jenkins Job with SVN repository

Part 1 - Create a Jenkins Job without repository

This job simply compile source code and run unit tests.

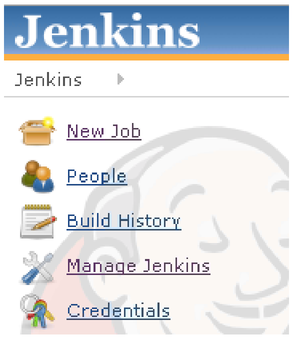
In this part you will not check out code from source control, instead you copy the code to job directory then build the project.

__1. Make sure Jenkins is started. Since we configured as windows service it will be started every time you start the machine.

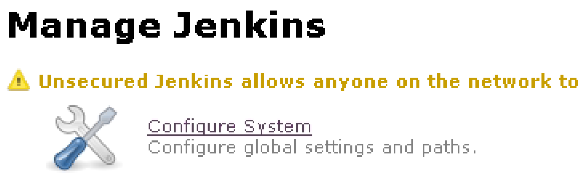
__2. Go to the Jenkins console:

```
http://localhost:8080
```

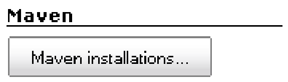
__3. On the Menu click **Manage Jenkins**.



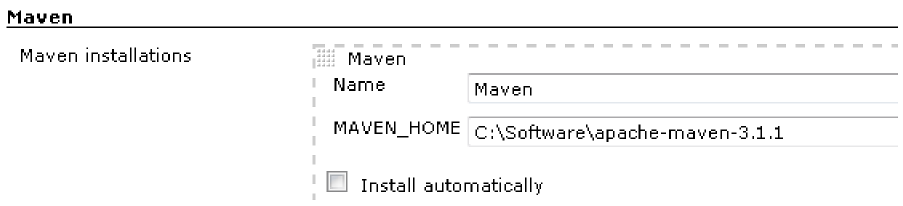
__4. Click **Configure System**.


Manage Jenkins
Unsecured Jenkins allows anyone on the network to
Configure System
Configure global settings and paths.

__5. Click the **Maven Installations** button



__6. Make sure the setting to look like below, if not change them and save.



Maven

Maven installations

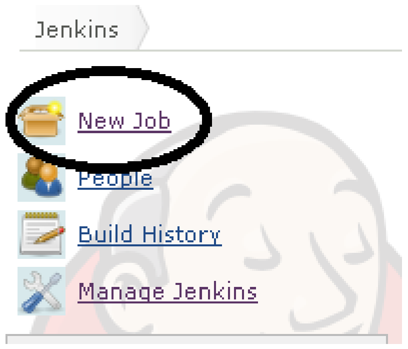
Maven

Name: Maven

MAVEN_HOME: C:\Software\apache-maven-3.1.1

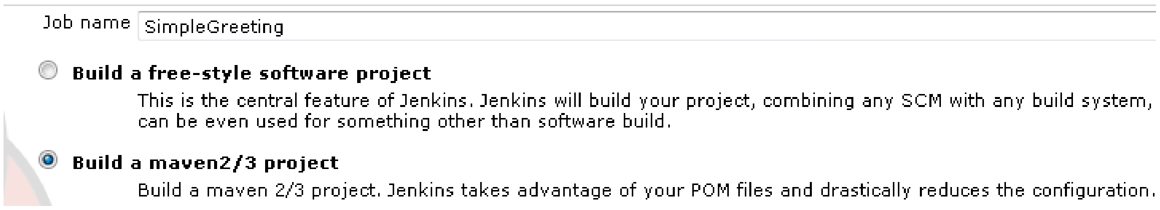
Install automatically

__7. Back in the main menu, click **New Job** link.



__8. Enter **SimpleGreeting** for job name.

__9. Select **Build a maven2/3 project**.



Job name SimpleGreeting

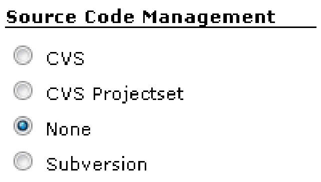
Build a free-style software project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, can be even used for something other than software build.

[✓ Build a maven2/3 project]
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

__10. Click **OK**, will add a new job.

After the job is created, you will be on the job configuration page.

__11. Scroll down a bit and make sure “Source Code Management” is selected None. Meaning in this part of the lab, you are not getting the source code from repository instead you will manually put the code in the Jenkins workspace.

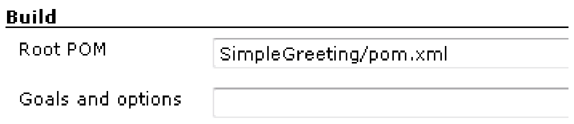


Source Code Management

CVS
CVS Projectset
[✓ None]
Subversion

__12. Scroll down and find **Build** section.

__13. Enter **SimpleGreeting/pom.xml** as **Root POM**.



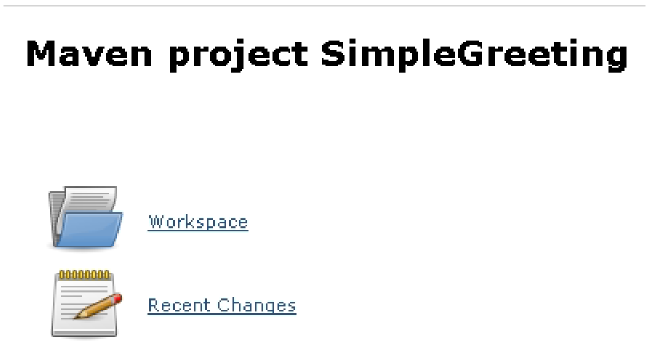
Build

Root POM SimpleGreeting/pom.xml

Goals and options

__14. Click **Save**.

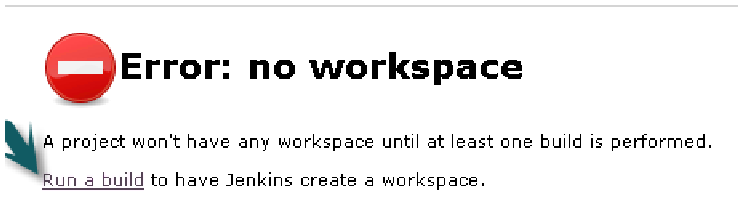
__15. You will see the Job screen:



Maven project SimpleGreeting

__16. Click **Workspace**.

__17. You will see an error, click **Run a Build**.



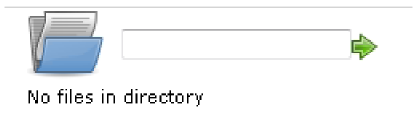
Error: no workspace

A project won't have any workspace until at least one build is performed.

Run a build to have Jenkins create a workspace.

__18. A blank page will open, simple click the **back** button in your browser to be again in Jenkins console.

__19. Refresh the page, you will see that the directory is empty.

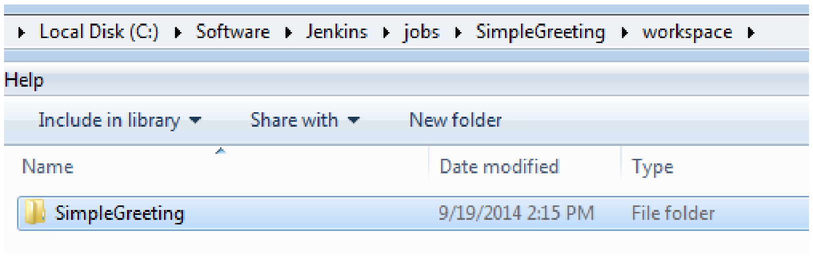


No files in directory

__20. Using a File browser, navigate to the workspace directory **C:\Software\Jenkins\jobs\SimpleGreeting\workspace**

Note. If you don't find the file search for the SimpleGreeting folder under C:\Software\Jenkins or ask your instructor.

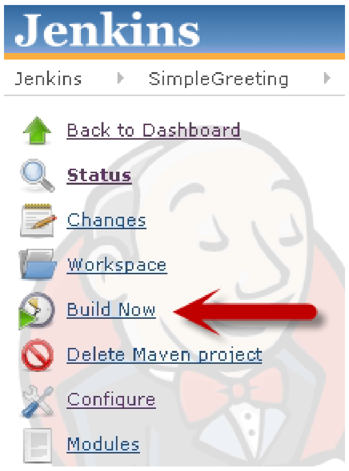
__21. Now copy **SimpleGreeting** folder from "**C:\LabFiles\Create A Jenkins Job**" to the workspace folder **C:\Software\Jenkins\jobs\SimpleGreeting\workspace**



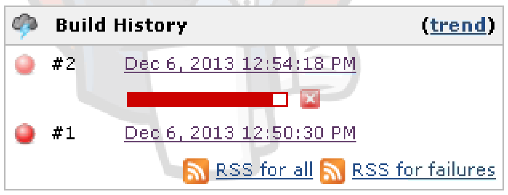
Local Disk (C:) ▸ Software ▸ Jenkins ▸ jobs ▸ SimpleGreeting ▸ workspace ▸
Help
Include in library
Share with
New folder

Name	Date modified	Type
SimpleGreeting	9/19/2014 2:15 PM	File folder

__22. Back in Jenkins click **Build Now** to build the project.



__23. You will see the progress indicator.


Build History (trend)

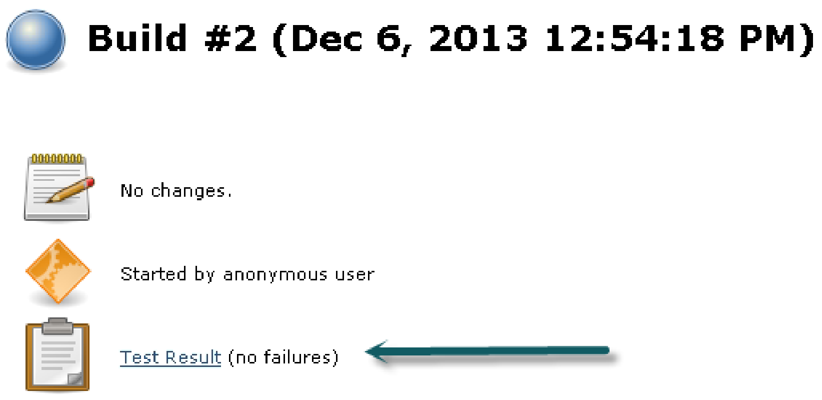
#2 Dec 6, 2013 12:54:18 PM

#1 Dec 6, 2013 12:50:30 PM

RSS for all RSS for failures

__24. Wait until is done (few seconds) and then click the link (#2). Note, build #1 created the workspace.

__25. You will see details of the build. Make sure result shows 'no failures'.

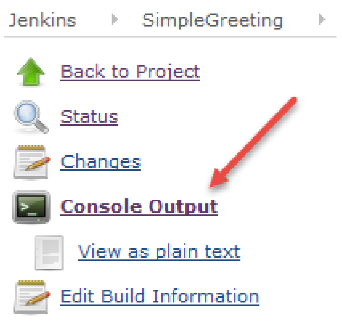

Build #2 (Dec 6, 2013 12:54:18 PM)

No changes.

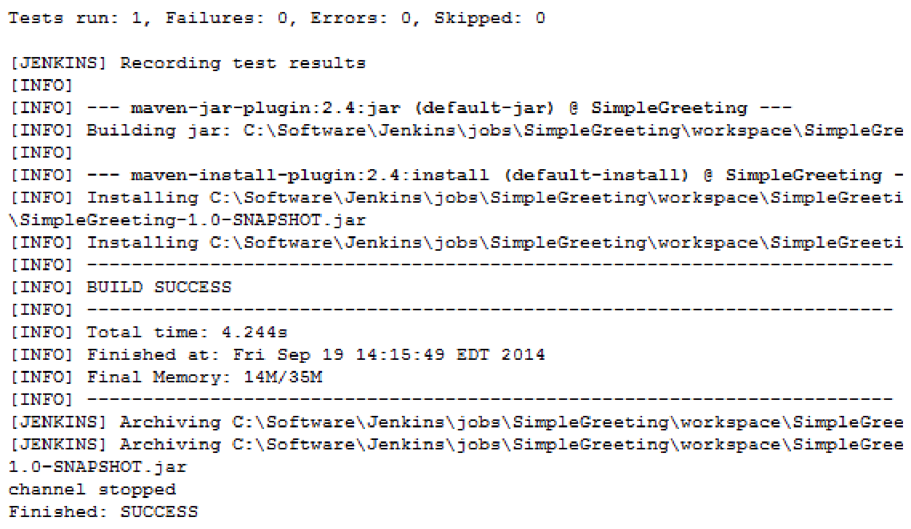
Started by anonymous user

Test Result (no failures)

__26. Click the **Console Output** from the left menu.



__27. At the end of the console you will also see the build success and successful build finish.



```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ SimpleGreeting ---
[INFO] Building jar: C:\Software\Jenkins\jobs\SimpleGreeting\workspace\SimpleGre
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ SimpleGreeting -
[INFO] Installing C:\Software\Jenkins\jobs\SimpleGreeting\workspace\SimpleGreeti
\SimpleGreeting-1.0-SNAPSHOT.jar
[INFO] Installing C:\Software\Jenkins\jobs\SimpleGreeting\workspace\SimpleGreeti
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.244s
[INFO] Finished at: Fri Sep 19 14:15:49 EDT 2014
[INFO] Final Memory: 14M/35M
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving C:\Software\Jenkins\jobs\SimpleGreeting\workspace\SimpleGree
[JENKINS] Archiving C:\Software\Jenkins\jobs\SimpleGreeting\workspace\SimpleGree
1.0-SNAPSHOT.jar
channel stopped
Finished: SUCCESS
```

You have created a project and built it successfully.

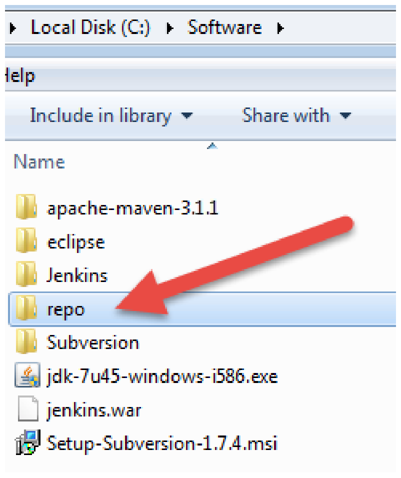
Part 2 - Subversion

__1. Open a command prompt and change to **C:\Software\Subversion\bin**

__2. Run the following command to create a repository:

```
svnadmin create c:\Software\repo
```

__3. The **C:\Software\repo** folder will be created.



```
apache-maven-3.1.1
eclipse
Jenkins
repo
Subversion
jdk-7u45-windows-i586.exe
jenkins.war
Setup-Subversion-1.7.4.msi
```

__4. Now you will add users to your server, using Notepad open the **C:\Software\repo\conf\svnserve.conf** file.

__5. Un-comment the following lines in **[general]** section:

```
anon-access = read
auth-access = write
password-db = passwd
```

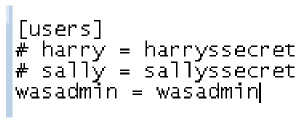
The above lines store the user credentials in the passwd file.

__6. Save and close the file.

__7. Open **C:\Software\repo\conf\passwd** file using Notepad.

__8. Under [**Users**] section, add the following user and password:

```
wasadmin = wasadmin
```



```
[users]
# harry = harryssecret
# sally = sallyssecret
wasadmin = wasadmin
```

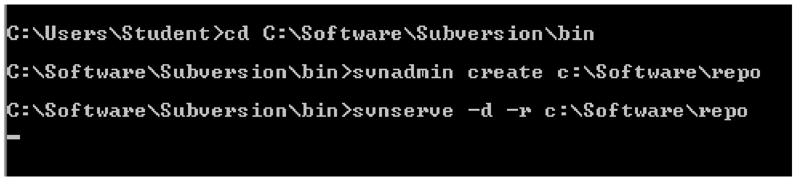
__9. Save and close the file.

__10. Back in the last command prompt used enter the following to start a subversion server:

```
svnserve -d -r c:\Software\repo
```

__11. If a Firewall windows open, allow to continue.

__12. The cursor won't do anything. Don't close the command window.



```
C:\Users\Student>cd C:\Software\Subversion\bin

C:\Software\Subversion\bin>svnadmin create c:\Software\repo

C:\Software\Subversion\bin>svnserve -d -r c:\Software\repo
_
```

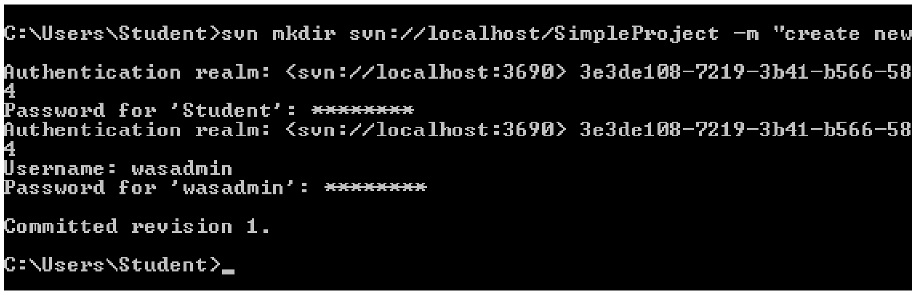
Next you will create a project folder in subversion.

__13. Open up a new command prompt. We can't use the old one as our daemon will shutdown. Once you're in the new window, type the following:

```
svn mkdir svn://localhost/SimpleProject -m "create new project"
```

__14. It may ask for your computer's password. If it does, you can simply type in your computer's password. Once you hit return, you can then go ahead and type in your subversion user-name and then your subversion password. Your subversion user-name and password is exactly what you typed in the passwd file. If you are not sure ask your instructor.

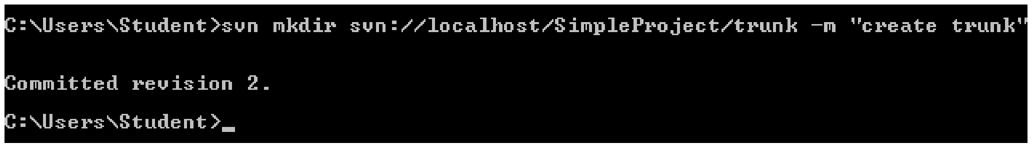
__15. You should get the following commit message:



```
C:\Users\Student>svn mkdir svn://localhost/SimpleProject -m "create new

Authentication realm: <svn://localhost:3690> 3e3de108-7219-3b41-b566-58
4
Password for 'Student': ********
Authentication realm: <svn://localhost:3690> 3e3de108-7219-3b41-b566-58
4
Username: wasadmin
Password for 'wasadmin': ********

Committed revision 1.

C:\Users\Student>_
```

__16. Create a trunk folder entering the following code so that all the active code will be in that folder:

```
svn mkdir svn://localhost/SimpleProject/trunk -m "create trunk"
```



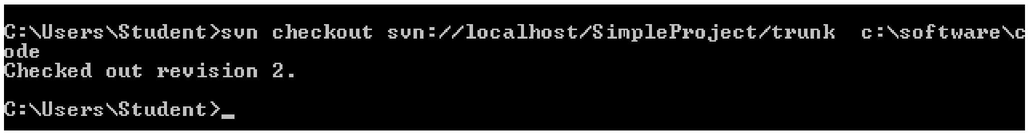
```
C:\Users\Student>svn mkdir svn://localhost/SimpleProject/trunk -m "create trunk"

Committed revision 2.

C:\Users\Student>_
```

__17. Under **C:\Software** create a folder called **code**

__18. To link our directory with all of our code to the SVN repository and tells it to start keeping it under version control, enter the following command on the prompt:

```
svn checkout svn://localhost/SimpleProject/trunk  c:\software\code
```

__19. You should see the message as shown below:



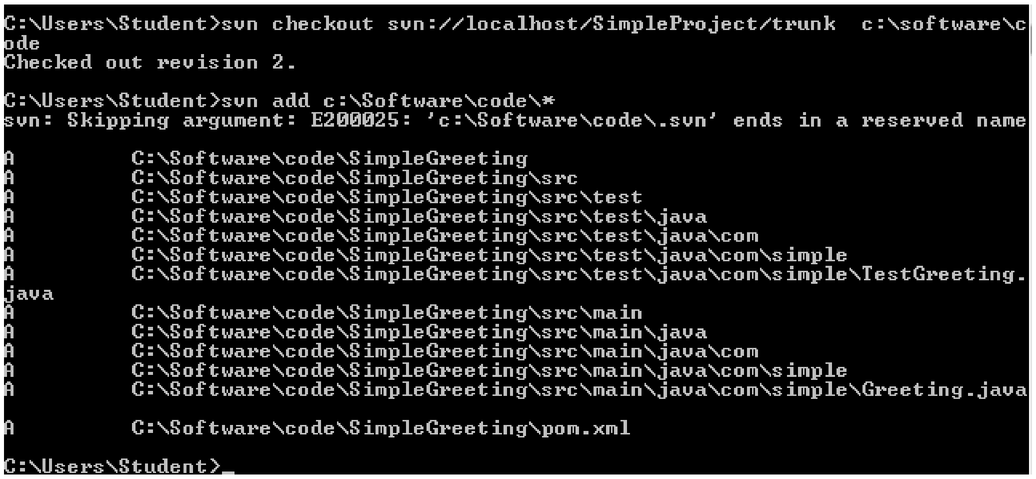
```
C:\Users\Student>svn checkout svn://localhost/SimpleProject/trunk  c:\software\c
ode
Checked out revision 2.

C:\Users\Student>_
```

__20. Now copy pom project **SimpleGreeting** from **C:\LabFiles\Create A Jenkins Job** folder and put it into **C:\Software\code** folder.

__21. Add the code to the repository by running the following command:

```
svn add c:\Software\code\*
```

__22. You should see the command prompt showing the activity:



```
C:\Users\Student>svn checkout svn://localhost/SimpleProject/trunk  c:\software\c
ode
Checked out revision 2.

C:\Users\Student>svn add c:\Software\code\*
svn: Skipping argument: E200025: 'c:\Software\code\.svn' ends in a reserved name

A         C:\Software\code\SimpleGreeting
A         C:\Software\code\SimpleGreeting\src
A         C:\Software\code\SimpleGreeting\src\test
A         C:\Software\code\SimpleGreeting\src\test\java
A         C:\Software\code\SimpleGreeting\src\test\java\com
A         C:\Software\code\SimpleGreeting\src\test\java\com\simple
A         C:\Software\code\SimpleGreeting\src\test\java\com\simple\TestGreeting.
java
A         C:\Software\code\SimpleGreeting\src\main
A         C:\Software\code\SimpleGreeting\src\main\java
A         C:\Software\code\SimpleGreeting\src\main\java\com
A         C:\Software\code\SimpleGreeting\src\main\java\com\simple
A         C:\Software\code\SimpleGreeting\src\main\java\com\simple\Greeting.java

A         C:\Software\code\SimpleGreeting\pom.xml

C:\Users\Student>_
```

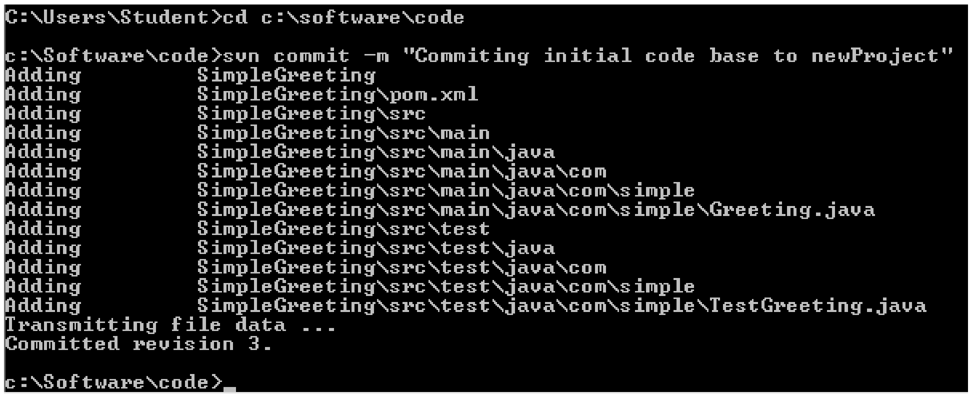
You will see the letter A beside each file telling you that these files are ready to be added to the repository.

__23. Enter the following to change folder.

```
cd c:\software\code
```

__24. Make sure you are at **c:\software\code** folder, then run the **commit** command:

```
svn commit -m "Commiting initial code base to newProject"
```



```
C:\Users\Student>cd c:\software\code

c:\Software\code>svn commit -m "Commiting initial code base to newProject"
Adding         SimpleGreeting
Adding         SimpleGreeting\pom.xml
Adding         SimpleGreeting\src
Adding         SimpleGreeting\src\main
Adding         SimpleGreeting\src\main\java
Adding         SimpleGreeting\src\main\java\com
Adding         SimpleGreeting\src\main\java\com\simple
Adding         SimpleGreeting\src\main\java\com\simple\Greeting.java
Adding         SimpleGreeting\src\test
Adding         SimpleGreeting\src\test\java
Adding         SimpleGreeting\src\test\java\com
Adding         SimpleGreeting\src\test\java\com\simple
Adding         SimpleGreeting\src\test\java\com\simple\TestGreeting.java
Transmitting file data ...
Committed revision 3.

c:\Software\code>_
```

You will see some output here as the code starts being transferred to the repository.

__25. Do a status command; you won't see any output as there are no changes and everything is synced up.

```
svn status
```

```
c:\Software\code>svn status

c:\Software\code>_
```

Part 3 - Create a Jenkins Job with SVN repository

In this part Jenkins will check out the code from SVN and build maven build.

__1. Go to the Jenkins console:

```
http://localhost:8080
```

__2. On the left menu, click **New Job** link.

__3. Enter **SimpleGreetingCodeFromSVN** for job name.

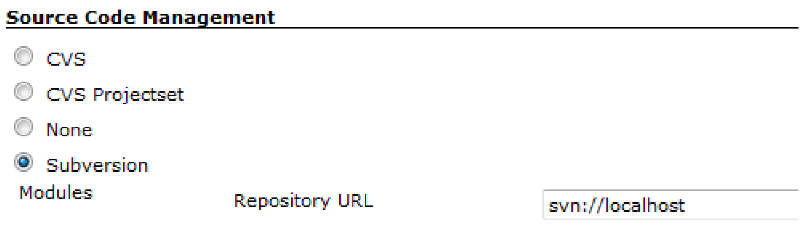
__4. Select **Build a maven2/3 project**.

__5. Click **OK**, Jenkins will add a new job.

After the job is created, you will be on the job configuration page.

__6. Scroll down a bit and from "Source Code Management" select **Subversion**.

__7. Meaning you are getting the source code from repository. Enter **svn://localhost** as URL Repository.

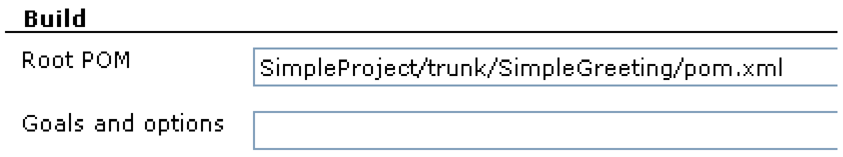


Source Code Management

- ( ) CVS
- ( ) CVS Projectset
- ( ) None
- (✓) Subversion

Modules — Repository URL: svn://localhost

__8. For build option we will set the pom.xml, enter **SimpleProject/trunk/SimpleGreeting/pom.xml**



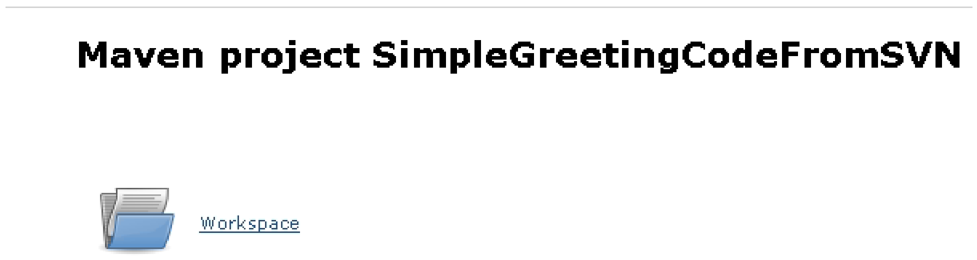
Build

Root POM: SimpleProject/trunk/SimpleGreeting/pom.xml

Goals and options:

__9. Click **Save**.

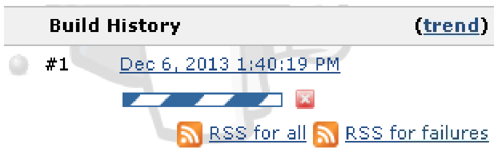
__10. You will see the Job screen:



Maven project SimpleGreetingCodeFromSVN

__11. Lets click on **Build Now** to kick a manual build.

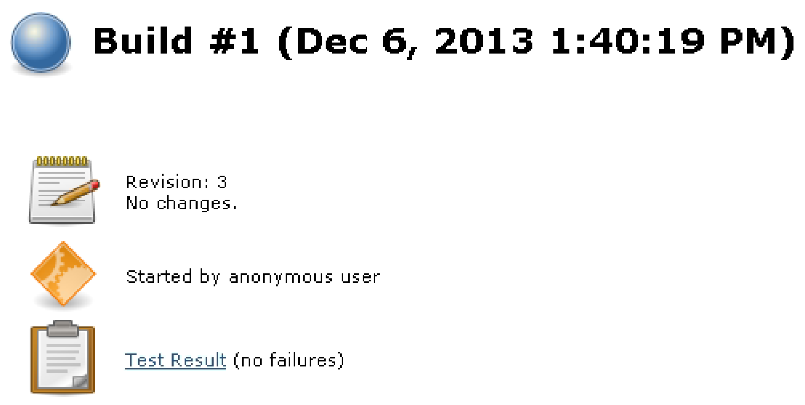
__12. Build will run.



Build History (trend)

#1 Dec 6, 2013 1:40:19 PM

__13. Click the link of the build to see the build results.



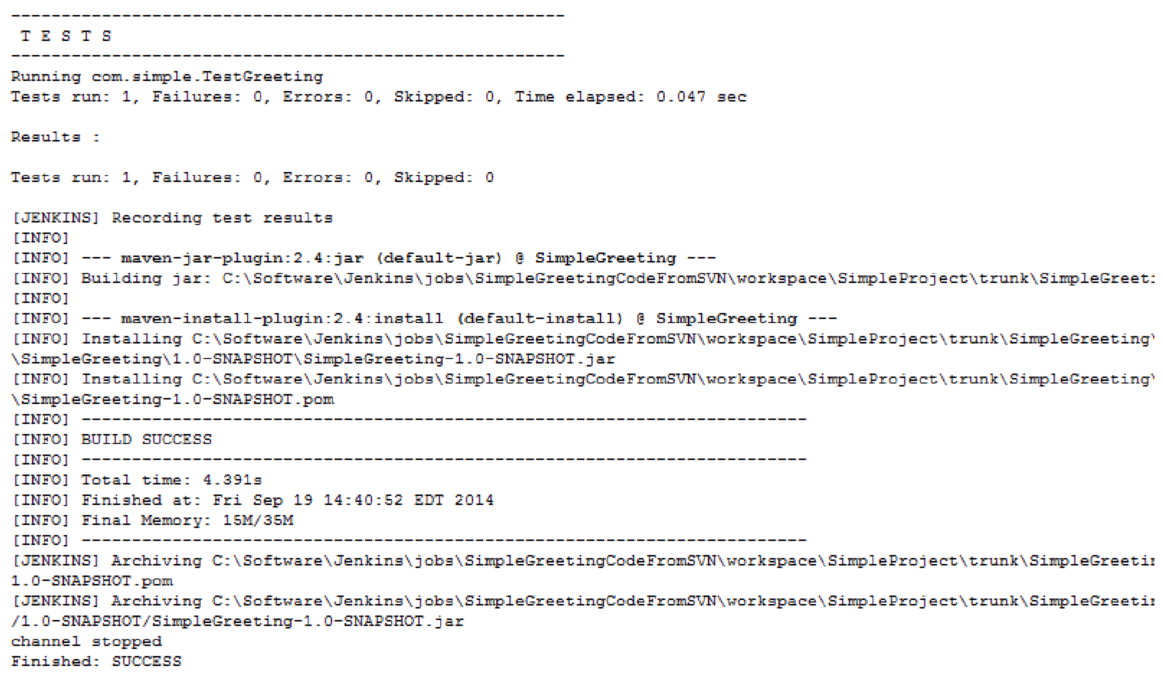
Build #1 (Dec 6, 2013 1:40:19 PM)

Revision: 3
No changes.

Started by anonymous user

Test Result (no failures)

__14. Click Console Output from the left menu to find out the build status, you will see different activities took place.



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.simple.TestGreeting
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.047 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ SimpleGreeting ---
[INFO] Building jar: C:\Software\Jenkins\jobs\SimpleGreetingCodeFromSVN\workspace\SimpleProject\trunk\SimpleGreet:
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ SimpleGreeting ---
[INFO] Installing C:\Software\Jenkins\jobs\SimpleGreetingCodeFromSVN\workspace\SimpleProject\trunk\SimpleGreeting\
\SimpleGreeting\1.0-SNAPSHOT\SimpleGreeting-1.0-SNAPSHOT.jar
[INFO] Installing C:\Software\Jenkins\jobs\SimpleGreetingCodeFromSVN\workspace\SimpleProject\trunk\SimpleGreeting\
\SimpleGreeting-1.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.391s
[INFO] Finished at: Fri Sep 19 14:40:52 EDT 2014
[INFO] Final Memory: 15M/35M
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving C:\Software\Jenkins\jobs\SimpleGreetingCodeFromSVN\workspace\SimpleProject\trunk\SimpleGreeti:
1.0-SNAPSHOT.pom
[JENKINS] Archiving C:\Software\Jenkins\jobs\SimpleGreetingCodeFromSVN\workspace\SimpleProject\trunk\SimpleGreeti:
/1.0-SNAPSHOT/SimpleGreeting-1.0-SNAPSHOT.jar
channel stopped
Finished: SUCCESS
```

__15. Close all browsers and command prompt windows opened.

Part 4 - Review

In this lab

- How to Set-up SVN and check-in and check-out codes.
- How to create a Jenkins Job without repository
- How to create a Jenkins Job with SVN repository

Lab 4 - Add Development Metrics

We are going to integrate code coverage metrics using the Cobertura plugin.

Code coverage is an indication of how much of your application code is actually executed during your tests—it can be a useful tool in particular for finding areas of code that have not been tested by your test suites. It can also give some indication as to how well a team is applying good testing practices such as Test-Driven Development or Behavior-Driven Development.

At the end of this lab you will be able to:

1. Install the Jenkins Cobertura Plugin
2. Configuring build tools
3. Run the code coverage without Jenkins

Part 1 - Install the Jenkins Cobertura Plugin

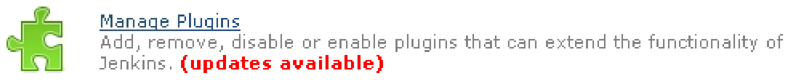
__1. Make sure Jenkins is started. Since we configured as windows service it will be started every time you start the machine.

__2. Go to the Jenkins console at:

```
http://localhost:8080
```

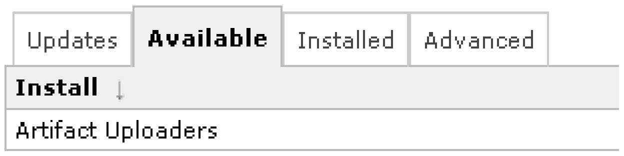
__3. To install a new plugin, click **Manage Jenkins** on the left.

__4. Click on the **Manage Plugins** entry.

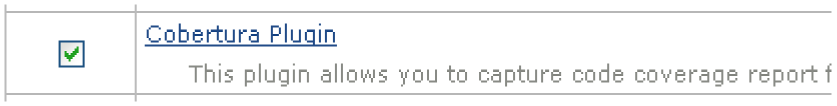


Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins. (updates available)

__5. Click on **Available** tab and scroll down until you find the **Cobertura Plugin** entry.

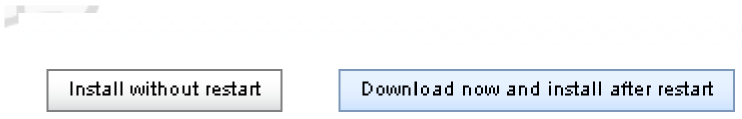


__6. Check check-box next to **Cobertura Plugin**.

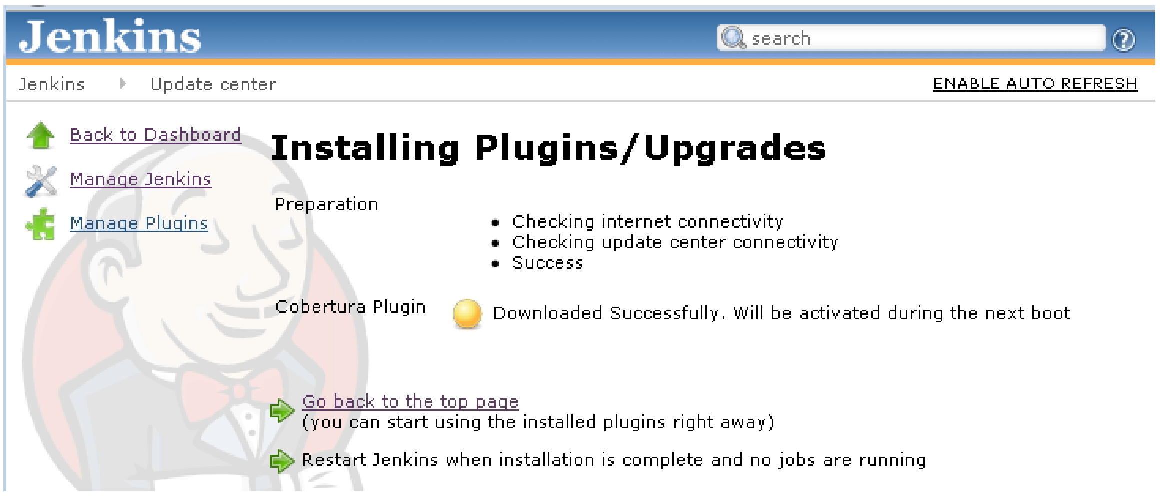


Cobertura Plugin
This plugin allows you to capture code coverage report f

__7. Click on the **Download now and Install after restart** button at the bottom of the screen.



__8. After plugin is downloaded, you would see the screen as shown below:



Installing Plugins/Upgrades

Preparation
- Checking internet connectivity
- Checking update center connectivity
- Success

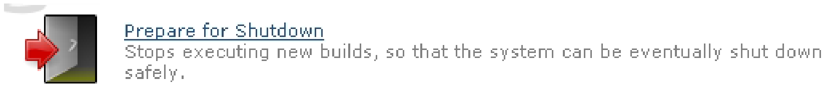
Cobertura Plugin ● Downloaded Successfully. Will be activated during the next boot

Go back to the top page
(you can start using the installed plugins right away)

Restart Jenkins when installation is complete and no jobs are running

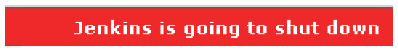
__9. We need to shut down Jenkins and start it again. Click **Manage Jenkins** link.

__10. Scroll down and click **Prepare for Shutdown**.



Prepare for Shutdown
Stops executing new builds, so that the system can be eventually shut down safely.

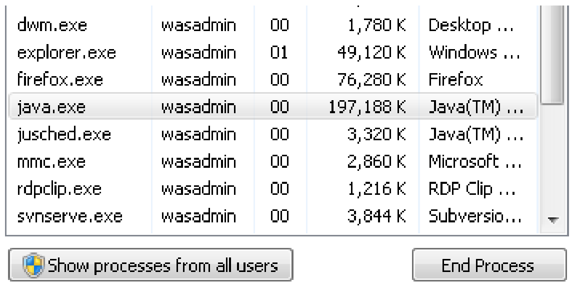
__11. A message will appear.



Jenkins is going to shut down

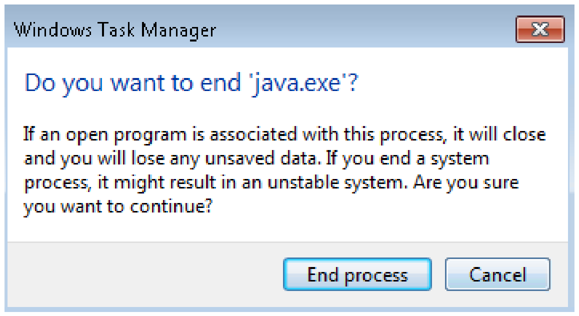
__12. Close the browser.

__13. Open the **Task Manager**, select the **Processes** tab, then select **Java** and click **End Process**. If Java is not present then continue to the next step.



```
dwm.exe        wasadmin   00     1,780 K   Desktop ...
explorer.exe   wasadmin   01    49,120 K   Windows ...
firefox.exe    wasadmin   00    76,280 K   Firefox
java.exe       wasadmin   00   197,188 K   Java(TM) ...
jusched.exe    wasadmin   00     3,320 K   Java(TM) ...
mmc.exe        wasadmin   00     2,860 K   Microsoft ...
rdpclip.exe    wasadmin   00     1,216 K   RDP Clip ...
svnserve.exe   wasadmin   00     3,844 K   Subversio...
```

Confirm you want to end the process.

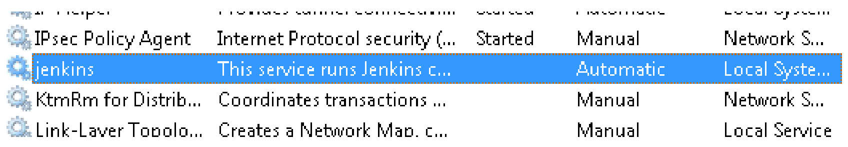


Windows Task Manager

Do you want to end 'java.exe'?

If an open program is associated with this process, it will close and you will lose any unsaved data. If you end a system process, it might result in an unstable system. Are you sure you want to continue?

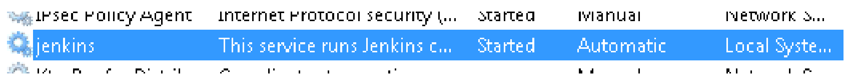
__14. Open the **Services** window and locate Jenkins.



```
IPsec Policy Agent     Internet Protocol security (...   Started   Manual      Network S...
jenkins                This service runs Jenkins c...              Automatic   Local Syste...
KtmRm for Distrib...   Coordinates transactions ...                Manual      Network S...
Link-Layer Topolo...   Creates a Network Map, c...                 Manual      Local Service
```

__15. Jenkins should be stopped, right click on it and select **Start**. If is still running then right click on it and select **Restart**. If you don't have privileges to restart Jenkins then restart the computer, and go to step 18.

__16. Verify it started successfully.



```
IPsec Policy Agent     Internet Protocol security (...   Started   Manual      Network S...
jenkins                This service runs Jenkins c...    Started   Automatic   Local Syste...
```

__17. Close **services** window.

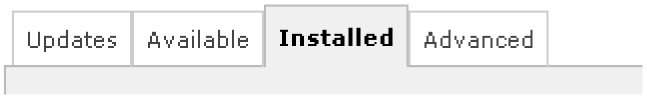
__18. Open Jenkins in a new a browser:

```
http://localhost:8080
```

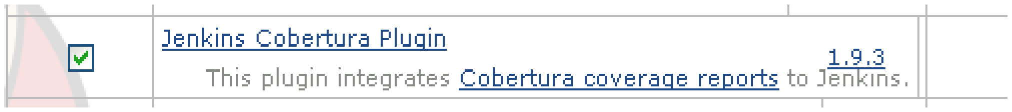
__19. Click **Manage Jenkins**.

__20. Click **Manage Plugin**.

__21. Click **Installed** tab.



__22. Scroll down and look for the newly installed plugin:



This plugin integrates Cobertura coverage reports to Jenkins.

This means the Cobertura Plugin is installed successfully.

__23. Click **Go Back to Dashboard**.

Part 2 - Configuring build tools

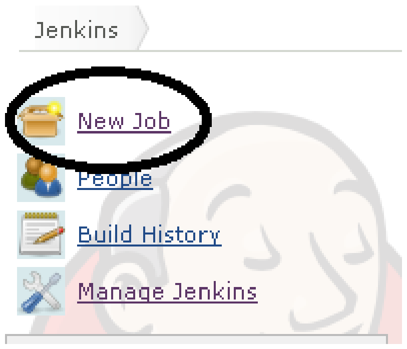
Configure Jenkins to run "cobertura" goal. Create a new build job to run the Cobertura coverage.

__1. Since code will be accessed from SVN repository, make sure SVN (Subversion server) is running. If it is not running, open a new command prompt, enter the following command.

```
svnserve -d -r c:\Software\repo
```

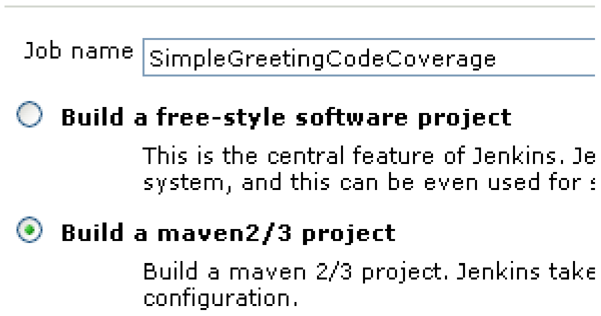
__2. Don't close the window, just minimize it.

__3. On the left menu on Jenkins, click **New Job** link.



__4. Enter **SimpleGreetingCodeCoverage** for job name.

__5. Select **Build a maven2/3** project.


Job name SimpleGreetingCodeCoverage

○ Build a free-style software project

This is the central feature of Jenkins. Je
system, and this can be even used for s

◉ Build a maven2/3 project

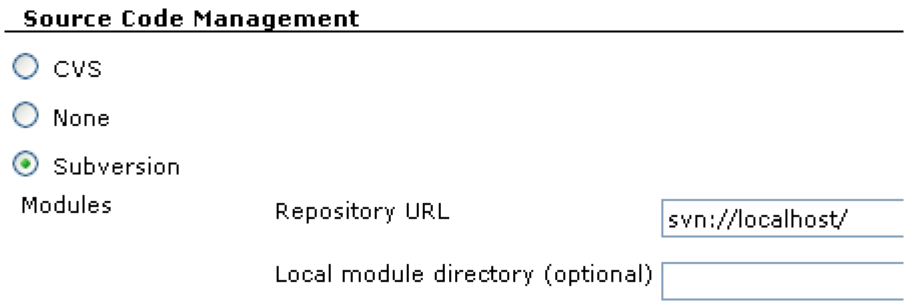
Build a maven 2/3 project. Jenkins take
configuration.

__6. Click **OK**, Jenkins will add a new job.

After the job is created, you will be on the job configuration page.

__7. Scroll down a bit and from "Source Code Management" select **Subversion**. Meaning you are getting the source code from repository.

__8. Enter **svn://localhost** as URL Repository


Source Code Management

○ CVS

○ None

◉ Subversion

Modules

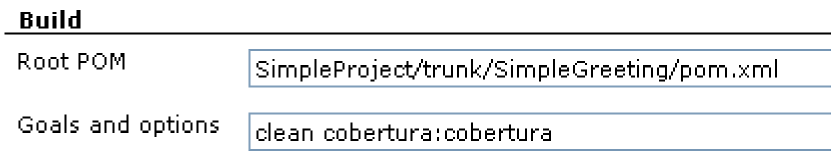
Repository URL svn://localhost/

Local module directory (optional)

__9. Scroll down and under **Build** section we will set the pom.xml, enter the followings:

Root POM : SimpleProject/trunk/SimpleGreeting/pom.xml

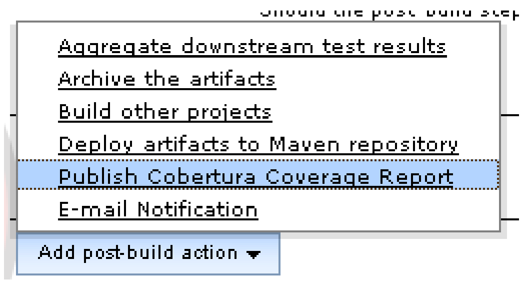
Goals and options: clean cobertura:cobertura


Build

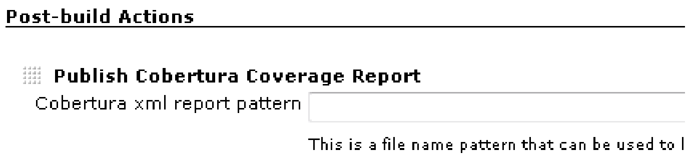
Root POM SimpleProject/trunk/SimpleGreeting/pom.xml

Goals and options clean cobertura:cobertura

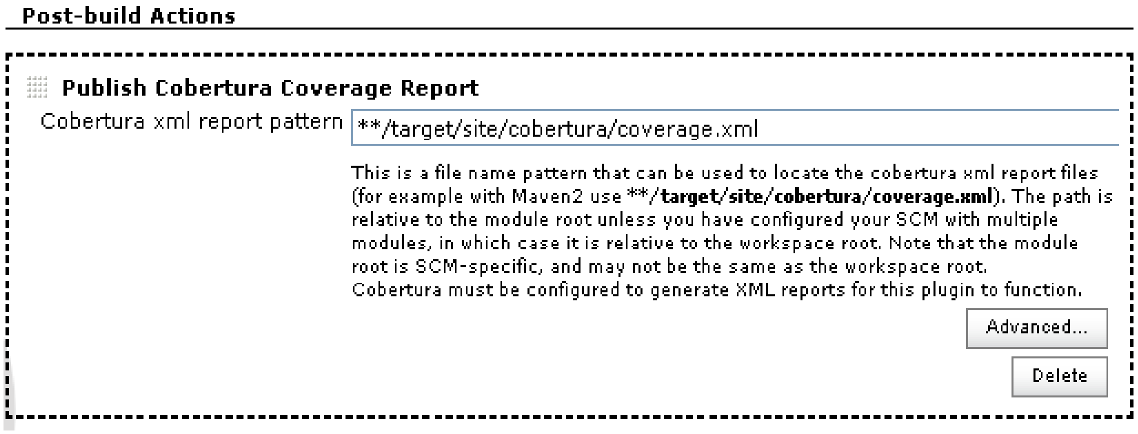
__10. Scroll down and from the **Add post-build Action** drop down, select **Publish Cobertura Coverage Report**.


Aggregate downstream test results
Archive the artifacts
Build other projects
Deploy artifacts to Maven repository
Publish Cobertura Coverage Report
E-mail Notification
Add post-build action

__11. As soon as you select the above drop down, you see the following fragment as below:


Post-build Actions
Publish Cobertura Coverage Report
Cobertura xml report pattern
This is a file name pattern that can be used to l

__12. Enter ****/target/site/cobertura/coverage.xml** for *Cobertura XML report pattern.*



Post-build Actions

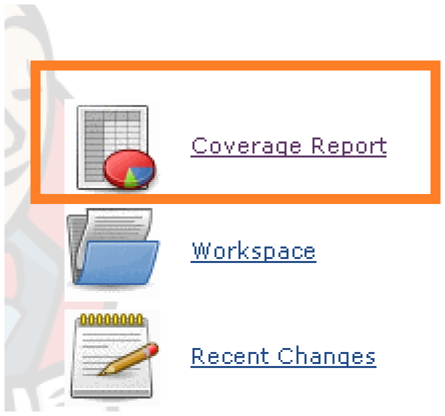
Publish Cobertura Coverage Report

Cobertura xml report pattern: **/target/site/cobertura/coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use ****/target/site/cobertura/coverage.xml**). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root. Cobertura must be configured to generate XML reports for this plugin to function.

__13. Click **Save**.

__14. You will see the project screen view and **Coverage Report** is activated shown as below:



Now lets update the **SimpleGreeting** Project pom file in SVN to handle report metrics.

__15. First make sure SVN server is running. If not run the following command to start:

```
svnserve -d -r c:\Software\repo
```

__16. Using a file browser navigate to **C:\Software\code\SimpleGreeting\**

__17. Open **pom.xml** using an editor.

__18. Add the code below in bold in the pom.xml.

```
...
 <build>
  <plugins>

    <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>cobertura-maven-plugin</artifactId>
       <version>2.5</version>
       <configuration>
         <formats>
             <format>html</format>
             <format>xml</format>
         </formats>
       </configuration>
    </plugin>
```

```
<plugin>
       <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>2.3.2</version>
          <configuration>
             <source>1.7</source>
             <target>1.7</target>
          </configuration>
    </plugin>
  </plugins>
 </build>

 <reporting>
   <plugins>
      <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>cobertura-maven-plugin</artifactId>
      </plugin>
   </plugins>
 </reporting>

</project>
```

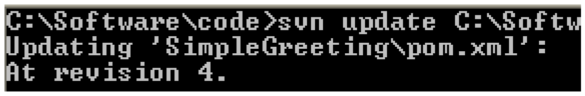
__19. Save and close the file.

__20. Open a command prompt.

__21. Change the directory to **C:\Software\code**

__22. Enter the following code to update the code to SVN.

```
svn update C:\Software\code\SimpleGreeting\pom.xml
```

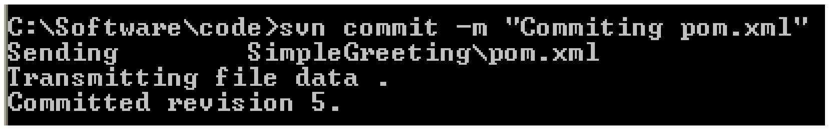


```
C:\Software\code>svn update C:\Softw
Updating 'SimpleGreeting\pom.xml':
At revision 4.
```

Note, revision # may be different.

__23. Enter the following command to commit the changes.

```
svn commit -m "Commiting pom.xml"
```

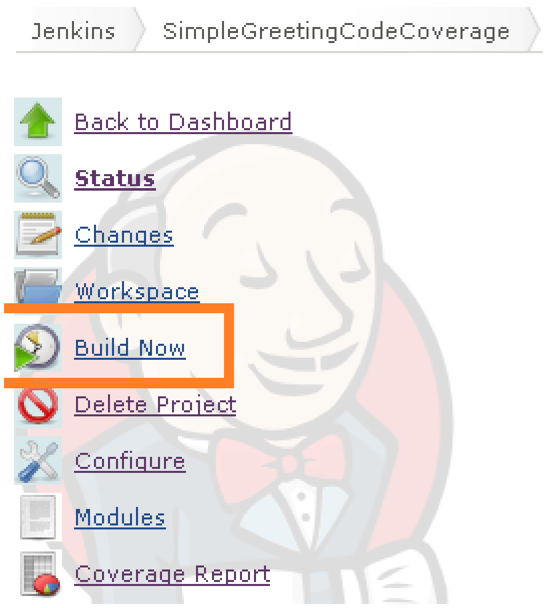


```
C:\Software\code>svn commit -m "Commiting pom.xml"
Sending        SimpleGreeting\pom.xml
Transmitting file data .
Committed revision 5.
```

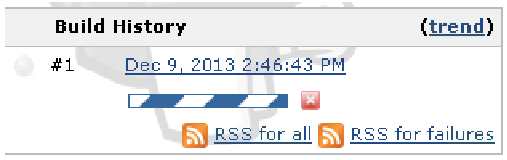
Note, revision # may be different.

Changes were committed successfully.

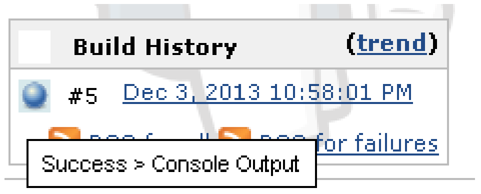
__24. Back to Jenkins page, click the **Build Now** to kick a manual build.



__25. Build should be in progress:

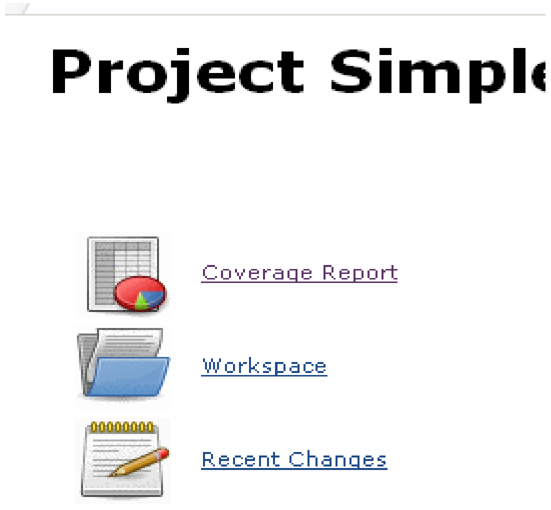


__26. Once the build is successful, move the mouse over to console like below and make sure it is indeed successful:

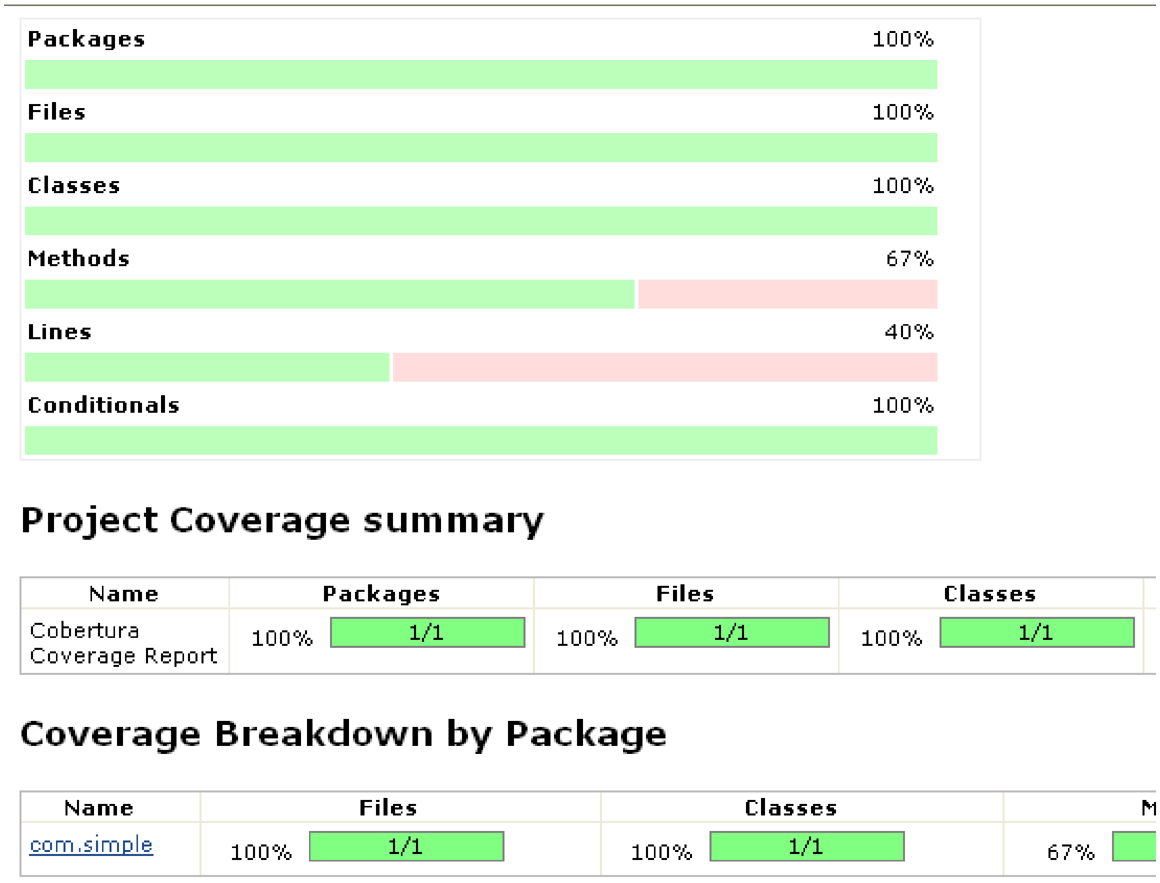


Note, Build # may vary depending on how many times you build.

__27. Now let's click on the **Coverage Report**.



__28. You will see the coverage report:



Packages	100%
Files	100%
Classes	100%
Methods	67%
Lines	40%
Conditionals	100%

Project Coverage summary

Name	Packages		Files		Classes	
Cobertura Coverage Report	100%	1/1	100%	1/1	100%	1/1

Coverage Breakdown by Package

Name	Files		Classes		M
com.simple	100%	1/1	100%	1/1	67%

Part 3 - Run the code coverage without Jenkins.

Cobertura is a free code coverage tool that calculates the percentage of code accessed by tests. It can be used to identify which parts of your Java program are lacking test coverage.

Cobertura produce code coverage report. Code coverage was among the first methods invented for systematic software testing.

This Lab describes how to generate a test coverage report for a site using the Cobertura Maven Plugin. You already entered the code in the pom in the previous part.

Generate Cobertura report:

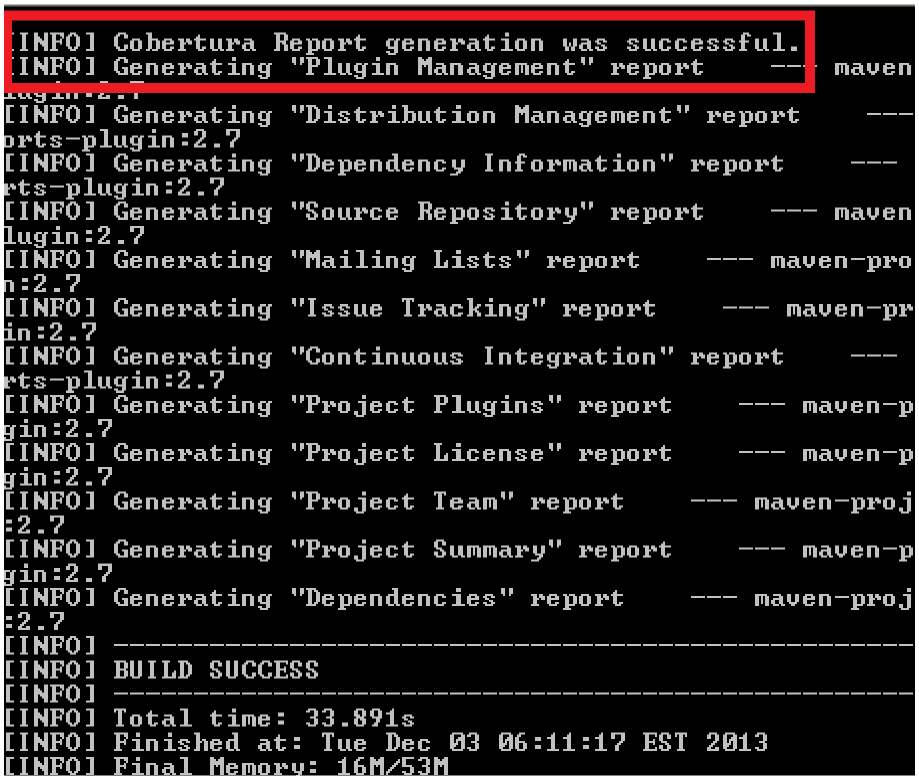
__1. Open a command prompt.

__2. Change the directory to **C:\Software\code\SimpleGreeting**

__3. Run following command to generate report:

```
mvn clean site
```

__4. You should see the successful report generation on the console:

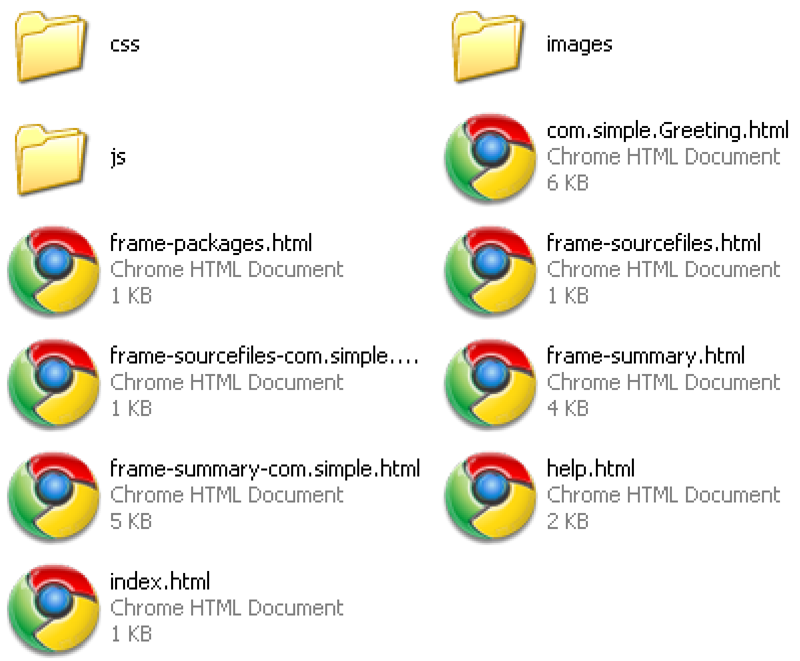


```
[INFO] Cobertura Report generation was successful.
[INFO] Generating "Plugin Management" report    --- maven
lugin:2.7
[INFO] Generating "Distribution Management" report    ---
orts-plugin:2.7
[INFO] Generating "Dependency Information" report    ---
rts-plugin:2.7
[INFO] Generating "Source Repository" report    --- maven
lugin:2.7
[INFO] Generating "Mailing Lists" report    --- maven-pro
n:2.7
[INFO] Generating "Issue Tracking" report    --- maven-pr
in:2.7
[INFO] Generating "Continuous Integration" report    ---
rts-plugin:2.7
[INFO] Generating "Project Plugins" report    --- maven-p
gin:2.7
[INFO] Generating "Project License" report    --- maven-p
gin:2.7
[INFO] Generating "Project Team" report    --- maven-proj
:2.7
[INFO] Generating "Project Summary" report    --- maven-p
gin:2.7
[INFO] Generating "Dependencies" report    --- maven-proj
:2.7
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------
[INFO] Total time: 33.891s
[INFO] Finished at: Tue Dec 03 06:11:17 EST 2013
[INFO] Final Memory: 16M/53M
```

Lets see the report site HTML files generated.

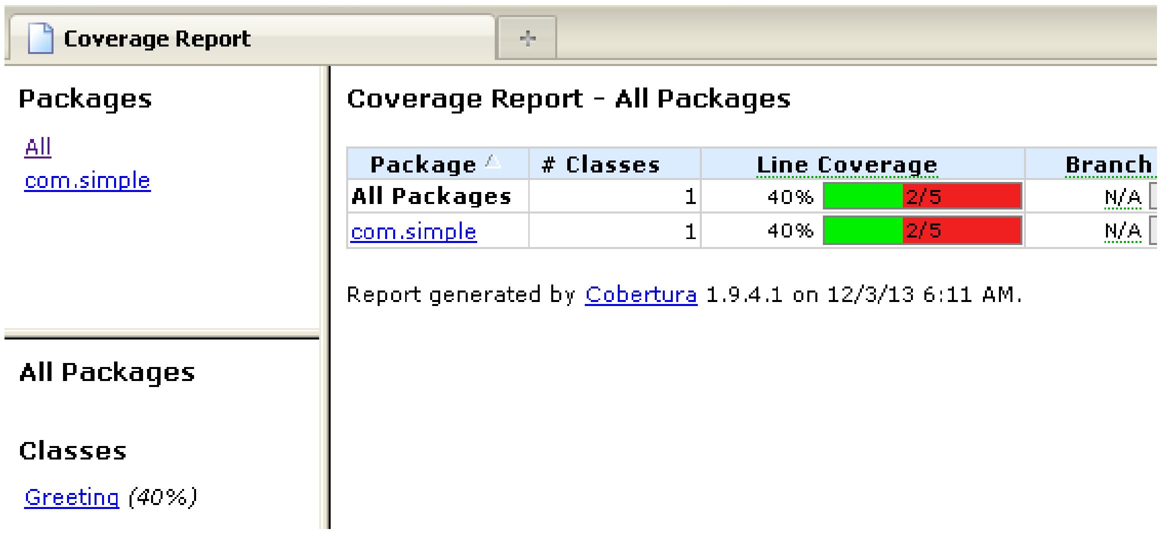
__5. Go to **C:\Software\code\SimpleGreeting\target\site\cobertura**.

__6. You should be able to see the generated HTML reports.



css
images
js
com.simple.Greeting.html
Chrome HTML Document
6 KB
frame-packages.html
Chrome HTML Document
1 KB
frame-sourcefiles.html
Chrome HTML Document
1 KB
frame-sourcefiles-com.simple....
Chrome HTML Document
1 KB
frame-summary.html
Chrome HTML Document
4 KB
frame-summary-com.simple.html
Chrome HTML Document
5 KB
help.html
Chrome HTML Document
2 KB
index.html
Chrome HTML Document
1 KB

__7. Open **index.html** to see the code coverage report as shown below:



Packages

All
com.simple

All Packages

Classes

Greeting (40%)

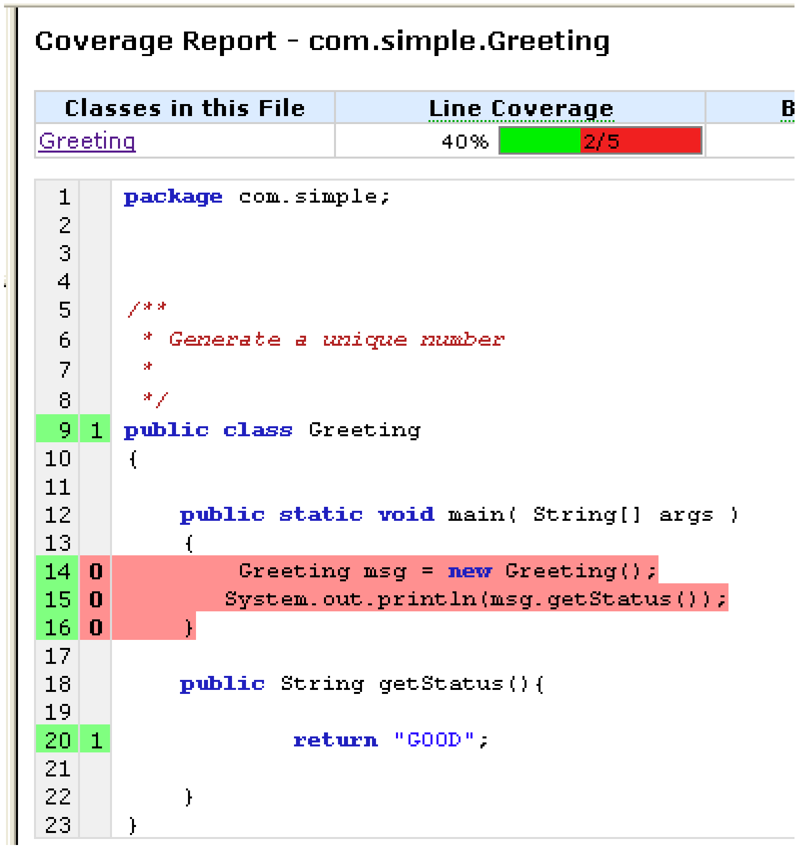
Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch
All Packages	1	40%	2/5	N/A
com.simple	1	40%	2/5	N/A

Report generated by Cobertura 1.9.4.1 on 12/3/13 6:11 AM.

__8. Click on **Greeting** on the above index page.

__9. You will see the code details:



Coverage Report - com.simple.Greeting

Classes in this File	Line Coverage	B
Greeting	40% 2/5	

```
package com.simple;



/**
 * Generate a unique number
 *
 */
public class Greeting
{

    public static void main( String[] args )
    {
        Greeting msg = new Greeting();
       System.out.println(msg.getStatus());
    }

    public String getStatus(){

            return "GOOD";

    }
}
```

Cobertura Features

- Instruments Java bytecode after it has been compiled.
- Can generate reports in HTML or XML.
- Shows percent of lines coveraged and branches coveraged for each class, package, and for the overall project. package, and for the overall product.
- Can sort HTML results by class name, percent of lines covered, percent of branches covered, etc. And can sort in ascending or descending order.

__10. Close all open browsers and command prompt windows.

Part 4 - Review

In this lab

- Learned how to configure Jenkins to generate report
- Learned hoe to run maven locally create a report site.

Lab 5 - Configure Jenkins Security

In this lab we will configure Jenkins to authenticate users against its internal user database, and enforce capability limitations on users.

At the end of this lab you will be able to:

1. Enable security and select the internal user database
2. Create users
3. Assign global privileges to users
4. Assign project-specific privileges to users.

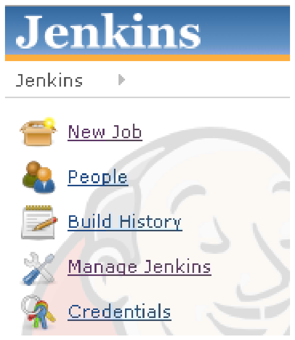
Part 1 - Enable Jenkins Security

__1. Make sure Jenkins is started. Since we configured as windows service it will be started every time you start the machine.

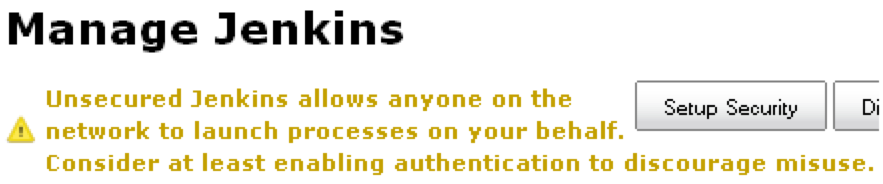
__2. Go to the Jenkins console:

```
http://localhost:8080
```

__3. On the Menu click **Manage Jenkins**.

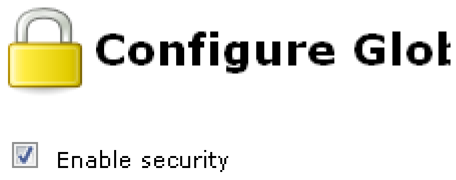


__4. Click **Setup Security**.

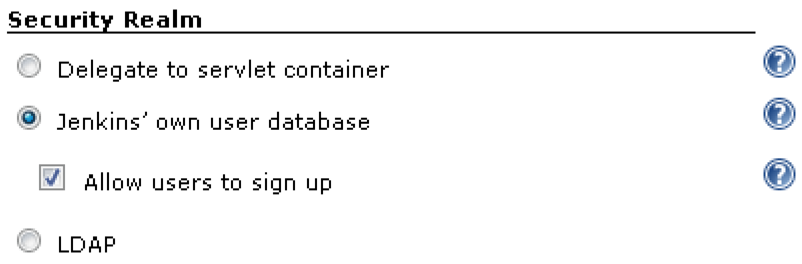


Unsecured Jenkins allows anyone on the network to launch processes on your behalf. Consider at least enabling authentication to discourage misuse.

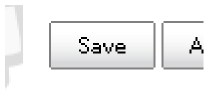
__5. Jenkins will display the **Configure Global Security** page. Click the check box for **Enable Security**, so it is selected:



__6. As soon as you click the security checkbox, Jenkins will display additional security options. Under the **Security Realm** heading, select **Jenkins own user database**. Leave **Allow users to sign up** selected.



__7. Click the **Save** button at the bottom of the page.

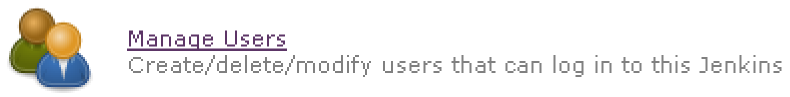


At this point, we have enabled security, but we haven't created any users, nor have we actually applied an authorization requirement. Let's go ahead and add a couple of users.

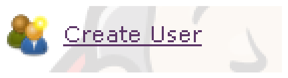
Part 2 - Create an Administrative User

There are two ways to add users: We can do it through Jenkins' management console, or we can allow users to sign up themselves. Let's first use the management console to add an administrative user so we can lock down the security.

__1. In the "Manage Jenkins" page (the last part of the lab should have left you here), click on **Manage Users**.



__2. Click on **Create User**.

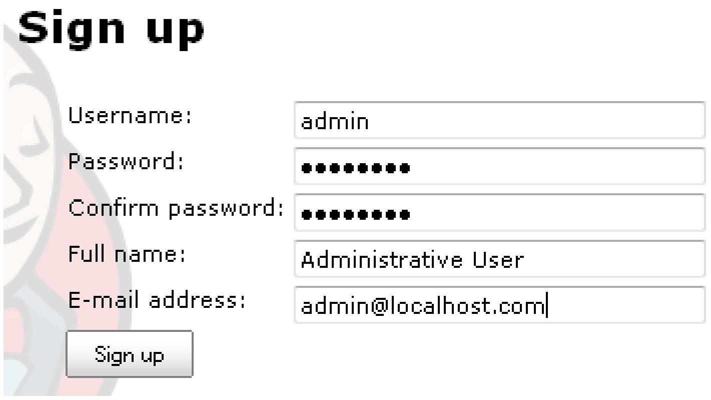


__3. The system displays the **Sign Up** page. Enter the following information in the appropriate fields:

Username:	admin
Password:	password
Confirm Password:	password
Full name:	Administrative User
E-mail address:	admin@localhost.com

Note that the e-mail address is not actually verified to be a valid email address, but Jenkins will reject it unless it is in the usual email format.

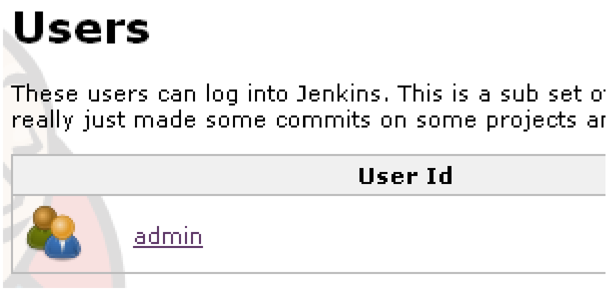
__4. When the page looks like below, click on the **Sign up** button.



Sign up

Username:	admin
Password:	●●●●●●●●
Confirm password:	●●●●●●●●
Full name:	Administrative User
E-mail address:	admin@localhost.com

__5. The system will display the list of current users, including the 'admin' user that you just created.



Users

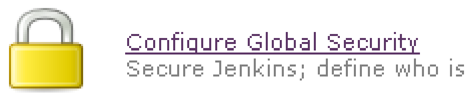
These users can log into Jenkins. This is a sub set o
really just made some commits on some projects ar

	User Id
	admin

__6. Click on **Manage Jenkins** to return to the management console.

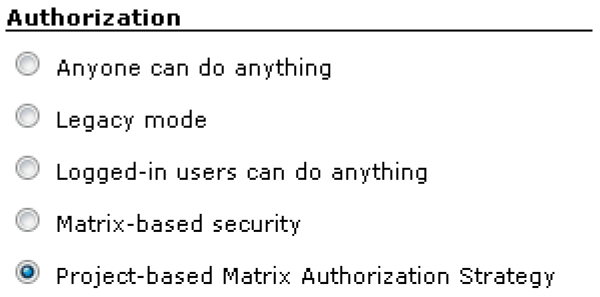
Part 3 - Enable Authentication

__1. Click on the link for **Configure Global Security**.



Configure Global Security
Secure Jenkins; define who is

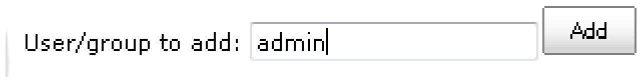
__2. Under the **Authorization** heading, select **Project-based Matrix Authorization Strategy**.



Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- [✓ Project-based Matrix Authorization Strategy]

__3. When you click on the radio button above, Jenkins will display a list of global authorizations. We need to enter the 'admin' user here with full permissions, and then we'll add other users to individual projects. In the field labeled **User/group to add:**, enter **admin**, and then click **Add**.

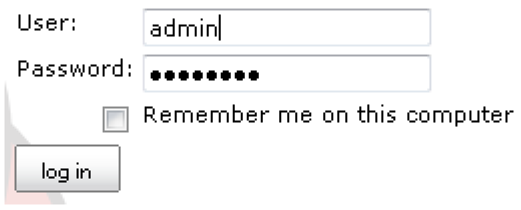


User/group to add: admin

__4. Jenkins displays the newly-added user in the list of users. Now we need to select all the permissions. You could click each permission box listed for 'admin' individually, but to save a little time, if you scroll the window horizontally all the way over to the right-hand side, you'll find a button that will select all the permissions in one operation . Find that button and click it.

__5. Click **Save**.

__6. Since we have altered the authorization strategy, Jenkins resets its security system, which requires us to log in again. At the login screen, enter **admin** as the userid and **password** as the password, then click **log in.**



User: admin
Password: ●●●●●●●●
☐ Remember me on this computer

__7. If for whatever reason, Jenkins doesn't let you log in, check with your instructor – you may need to reset the security system by editing the configuration file manually and then start over with the security setup.

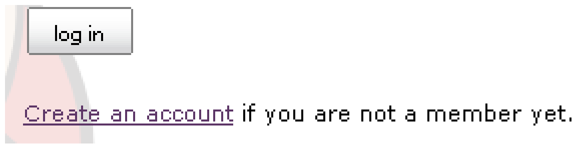
__8. Click the **log out** link at the upper-right corner of Jenkins' home page.

__9. Jenkins should display the login page again.

Part 4 - Create a Self-Signed-Up User

When we enabled Jenkins security, we left the checkbox selected for "Allow users to sign up". As a result, there is a **Create an Account** link on the login page. Let's create a user that way, and then we'll go back and grant them privileges on a project.

__1. Click on **Create an account** in the login page.

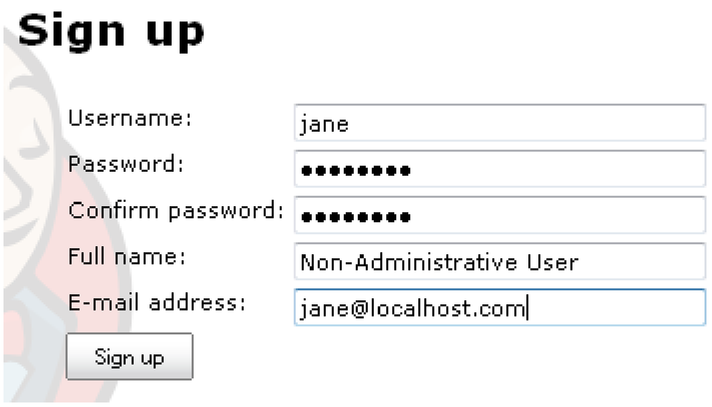


__2. Jenkins will display the **Sign up** page. Enter the following information in the appropriate fields:

Username:	jane
Password:	password
Confirm Password:	password
Full name:	Non-Administrative User
E-mail address:	jane@localhost.com

Note that the e-mail address is not actually verified to be a valid email address, but Jenkins will reject it unless it is in the usual email format.

__3. When the page looks like below, click on the **Sign up** button.

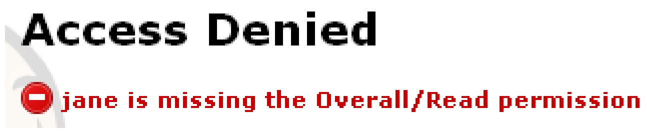


Sign up

Username: jane
Password: ●●●●●●●●
Confirm password: ●●●●●●●●
Full name: Non-Administrative User
E-mail address: jane@localhost.com

__4. Jenkins will display the **Success** window. Click on the link to go to **the top page**.

__5. Since the new user has no permissions, access is denied.



Access Denied

jane is missing the Overall/Read permission

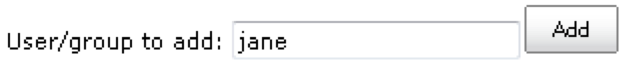
__6. Click on the **log out** link, and then log back in using 'admin/password'.

__7. We should see the main dashboard page. Click on the **SimpleGreetingCodeFromSVN** job (we created this job in an earlier lab).

__8. Click **Configure**

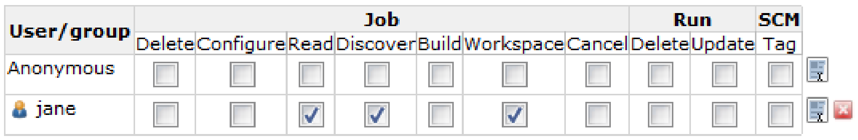
__9. In the configuration page, click on the checkbox marked **Enable project-based security**, to select it.

__10. When you select the checkbox, Jenkins will display a matrix of users and permissions. In the field marked **User/group to add:**, enter **jane**, and then click **Add**.



User/group to add: jane

__11. In the row labeled **jane**, select the checkboxes for **Read, Discover** and **Workspace**.



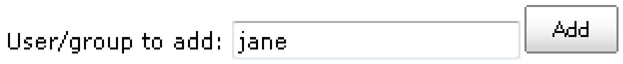
User/group	Job							Run		SCM
	Delete	Configure	Read	Discover	Build	Workspace	Cancel	Delete	Update	Tag
Anonymous	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
jane	☐	☐	✓	✓	☐	✓	☐	☐	☐	☐

__12. Click **Save**.

__13. We also need to grant jane the 'overall read' permission. Click **Back to Dashboard.**

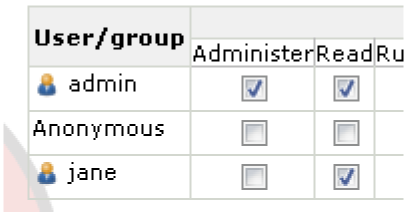
__14. Click the **Manage Jenkins** link, and then select **Configure Global Security**.

__15. Under the **Authorization** heading, in the field marked **User/group to add:** enter **jane** and then click **Add**.



User/group to add: jane

__16. In the row labeled **jane**, select the checkbox for **Read**.



User/group	Administer	Read	Ru
admin	✓	✓	
Anonymous	☐	☐	
jane	☐	✓	

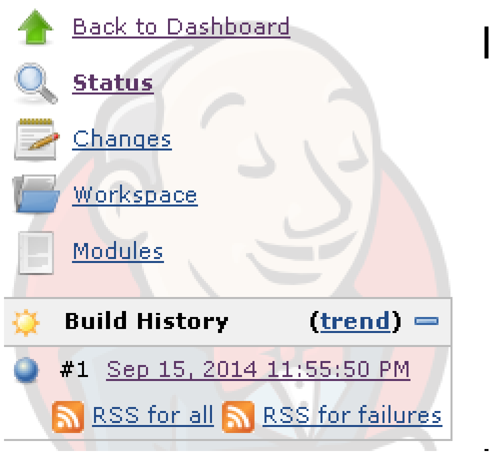
__17. Click **Save**.

__18. Log out, and then log back in as 'jane/password'.

__19. Note that Jenkins only displays the 'SimpleGreetingCodeFromSVN' job, even though we have a total of three jobs created. This is because Jane only has read and discover access to one project.

__20. Click on the **SimpleGreetingCodeFromSVN** job.

__21. Notice that Jane doesn't have full privileges – there is no **Build Now** or **Configure** option on the project.



__22. Log out of Jenkins.

__23. Close the browser.

Part 5 - Review

In this lab, we went through a series of steps to enable and configure Jenkins security. We saw how to create users administratively, and how to configure users who sign up using the self-sign-up screen.