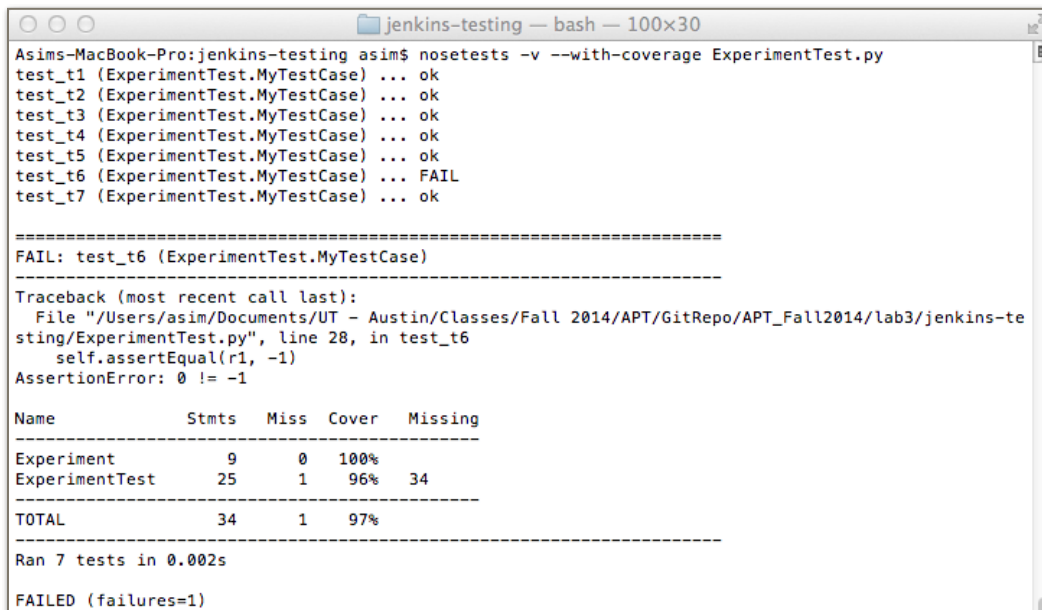


LAB 3 - Jenkins

The following document contains screenshots of the various steps carried out for setting up the lab for Jenkins. In addition to that, it contains the screenshots of expected deliverables of Jenkins showing the status with different iterations of the code.

Before installing Jenkins, a test run of the code provided in the lab using “nosetests” gave the following output



```
jenkins-testing — bash — 100x30
Asims-MacBook-Pro:jenkins-testing asim$ nosetests -v --with-coverage ExperimentTest.py
test_t1 (ExperimentTest.MyTestCase) ... ok
test_t2 (ExperimentTest.MyTestCase) ... ok
test_t3 (ExperimentTest.MyTestCase) ... ok
test_t4 (ExperimentTest.MyTestCase) ... ok
test_t5 (ExperimentTest.MyTestCase) ... ok
test_t6 (ExperimentTest.MyTestCase) ... FAIL
test_t7 (ExperimentTest.MyTestCase) ... ok

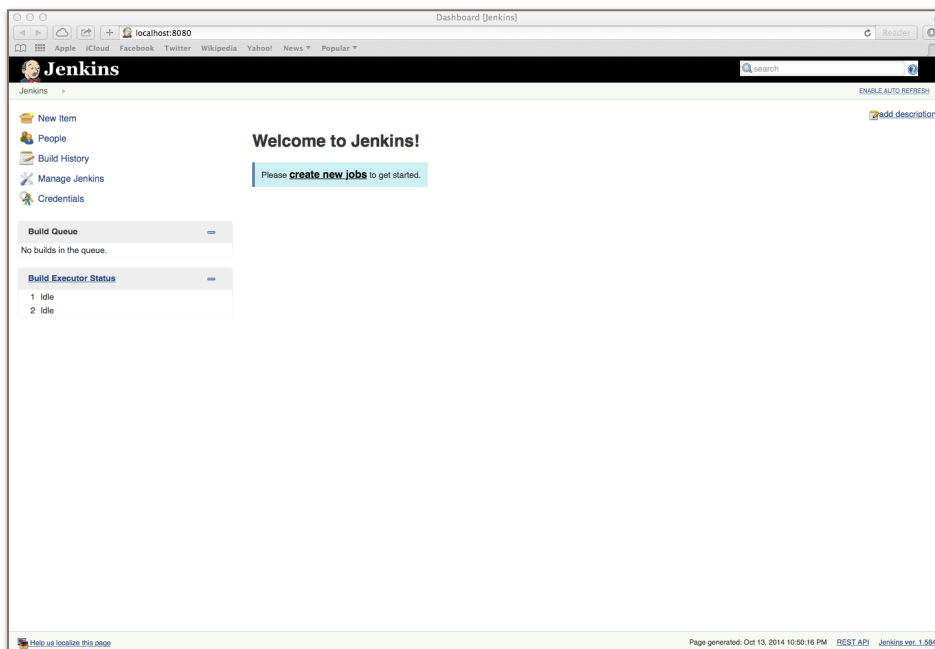
=====
FAIL: test_t6 (ExperimentTest.MyTestCase)
=====
Traceback (most recent call last):
  File "/Users/asim/Documents/UT - Austin/Classes/Fall 2014/APT/GitRepo/APT_Fall2014/lab3/jenkins-testing/ExperimentTest.py", line 28, in test_t6
    self.assertEqual(r1, -1)
AssertionError: 0 != -1

Name           Stmts   Miss  Cover   Missing
-----
Experiment       9      0   100%
ExperimentTest   25      1    96%   34
TOTAL            34      1    97%

Ran 7 tests in 0.002s

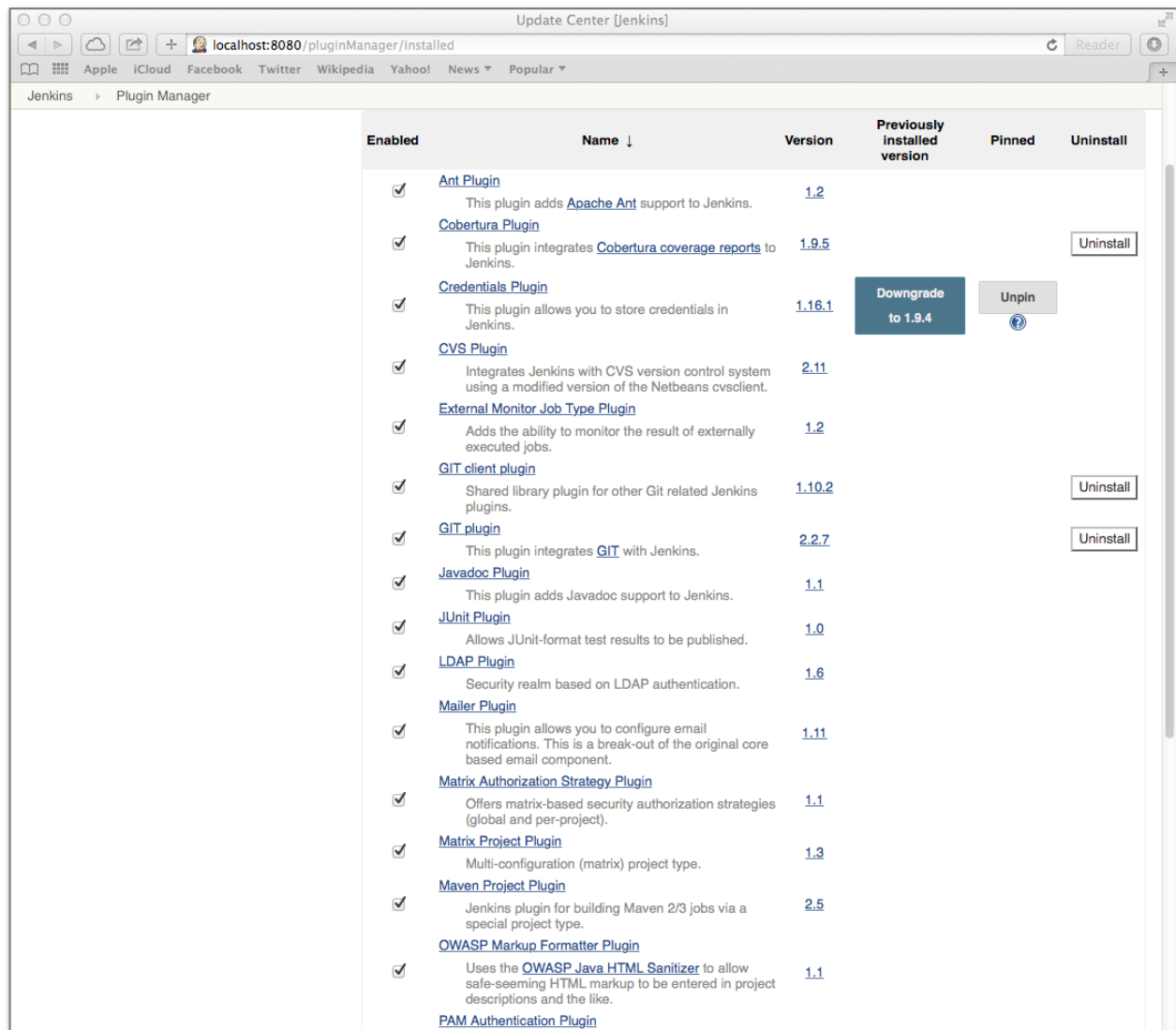
FAILED (failures=1)
```

After installing Jenkins for Mac, the following screen pops up indicating a successful installation. It started up on localhost:8080 port



In order to continue, we need to install the plugins required for Jenkins to perform the operations we require.

To do this, we access the Jenkins plugin repository and install the desired plugins, namely the “Jenkins GIT Plugin” and the “Jenkins Cobertura Plugin”



The screenshot shows the Jenkins Update Center interface. The browser address bar indicates the URL is `localhost:8080/pluginManager/installed`. The page title is "Update Center [Jenkins]". The breadcrumb navigation shows "Jenkins" > "Plugin Manager".

Enabled	Name ↓	Version	Previously installed version	Pinned	Uninstall
<input checked="" type="checkbox"/>	Ant Plugin This plugin adds Apache Ant support to Jenkins.	1.2			
<input checked="" type="checkbox"/>	Cobertura Plugin This plugin integrates Cobertura coverage reports to Jenkins.	1.9.5			<button>Uninstall</button>
<input checked="" type="checkbox"/>	Credentials Plugin This plugin allows you to store credentials in Jenkins.	1.16.1	<button>Downgrade to 1.9.4</button>	<button>Unpin</button>	
<input checked="" type="checkbox"/>	CVS Plugin Integrates Jenkins with CVS version control system using a modified version of the Netbeans cvsclient.	2.11			
<input checked="" type="checkbox"/>	External Monitor Job Type Plugin Adds the ability to monitor the result of externally executed jobs.	1.2			
<input checked="" type="checkbox"/>	GIT client plugin Shared library plugin for other Git related Jenkins plugins.	1.10.2			<button>Uninstall</button>
<input checked="" type="checkbox"/>	GIT plugin This plugin integrates GIT with Jenkins.	2.2.7			<button>Uninstall</button>
<input checked="" type="checkbox"/>	Javadoc Plugin This plugin adds Javadoc support to Jenkins.	1.1			
<input checked="" type="checkbox"/>	JUnit Plugin Allows JUnit-format test results to be published.	1.0			
<input checked="" type="checkbox"/>	LDAP Plugin Security realm based on LDAP authentication.	1.6			
<input checked="" type="checkbox"/>	Mailer Plugin This plugin allows you to configure email notifications. This is a break-out of the original core based email component.	1.11			
<input checked="" type="checkbox"/>	Matrix Authorization Strategy Plugin Offers matrix-based security authorization strategies (global and per-project).	1.1			
<input checked="" type="checkbox"/>	Matrix Project Plugin Multi-configuration (matrix) project type.	1.3			
<input checked="" type="checkbox"/>	Maven Project Plugin Jenkins plugin for building Maven 2/3 jobs via a special project type.	2.5			
<input checked="" type="checkbox"/>	OWASP Markup Formatter Plugin Uses the OWASP Java HTML Sanitizer to allow safe-seeming HTML markup to be entered in project descriptions and the like.	1.1			
	PAM Authentication Plugin				

Once we install the desired plugins, it is time to setup the configuration required for the plugins to work.

Configure System [Jenkins]

localhost:8080/configure

Jenkins > configuration

List of Maven installations on this system

Maven Project Configuration

Global MAVEN_OPTS

Local Maven Repository

☒ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

Download Preferences

Use Browser ☒

Jenkins Location

Jenkins URL

⚠ Please set a valid host name, instead of localhost

System Admin e-mail address

SSH Server

SSHD Port ☐ Fixed : ☒ Random ☐ Disable

Git plugin

Global Config user.name Value

Global Config user.email Value

Create new accounts base on author/commmitter's email ☐

CVS

Default Compression Level

Private Key Location

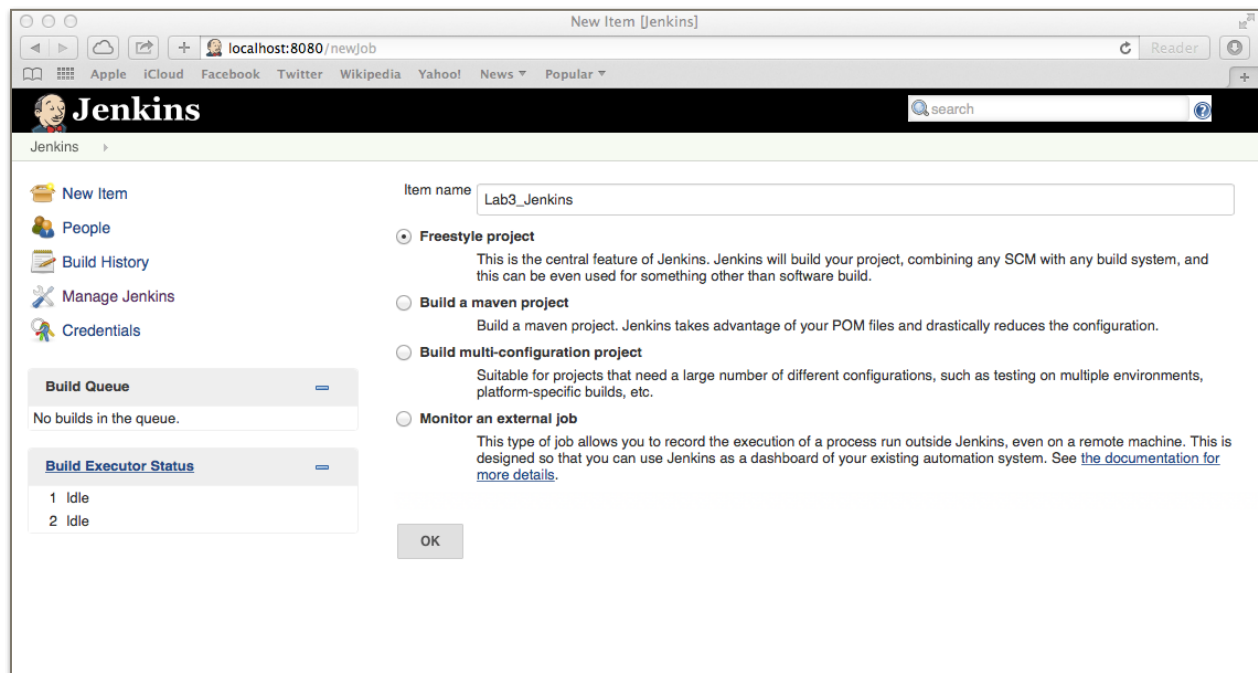
Private Key Password

Known Hosts Location

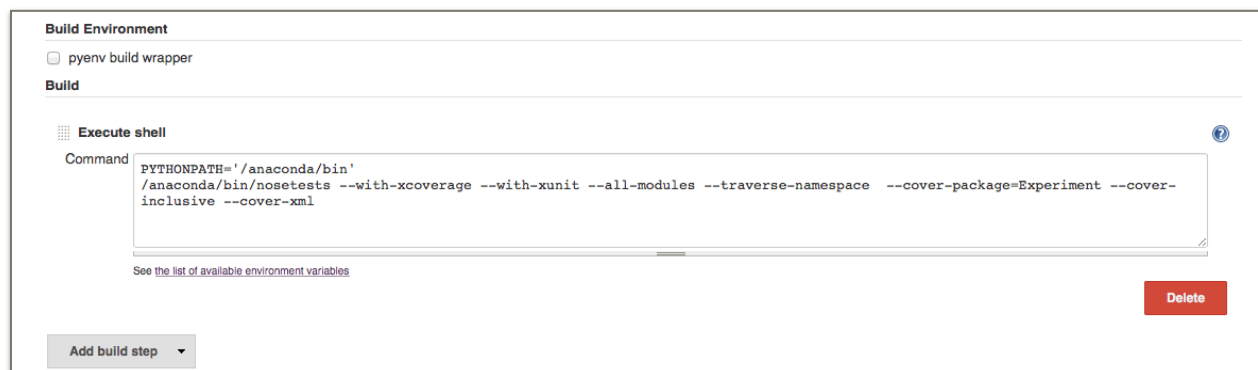
Authentication

Subversion

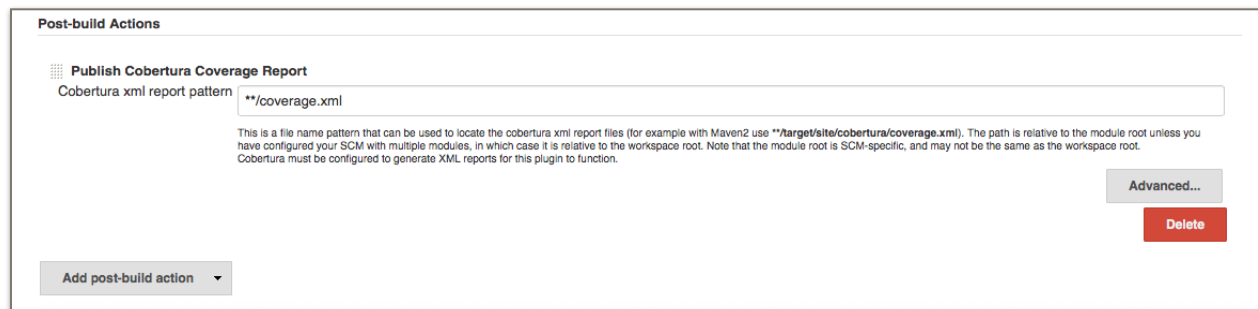
After the configurations are set, we configure the Job in Jenkins to point to the code we have already setup on Git.



We set up the Build command as follows. This takes care of executing the “nosetests” and outputting the results in a XML format

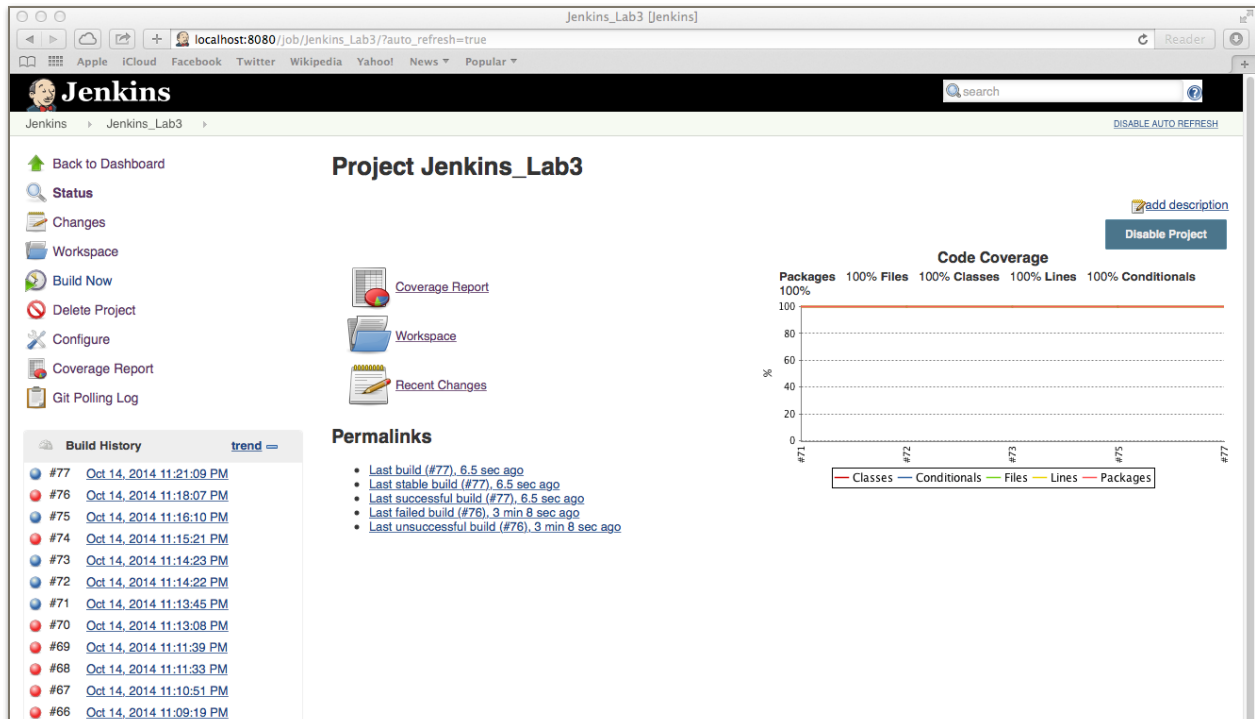


The XML file generated by the Execute Shell process in Build is then displayed using the Cobertura plugin as configured below:

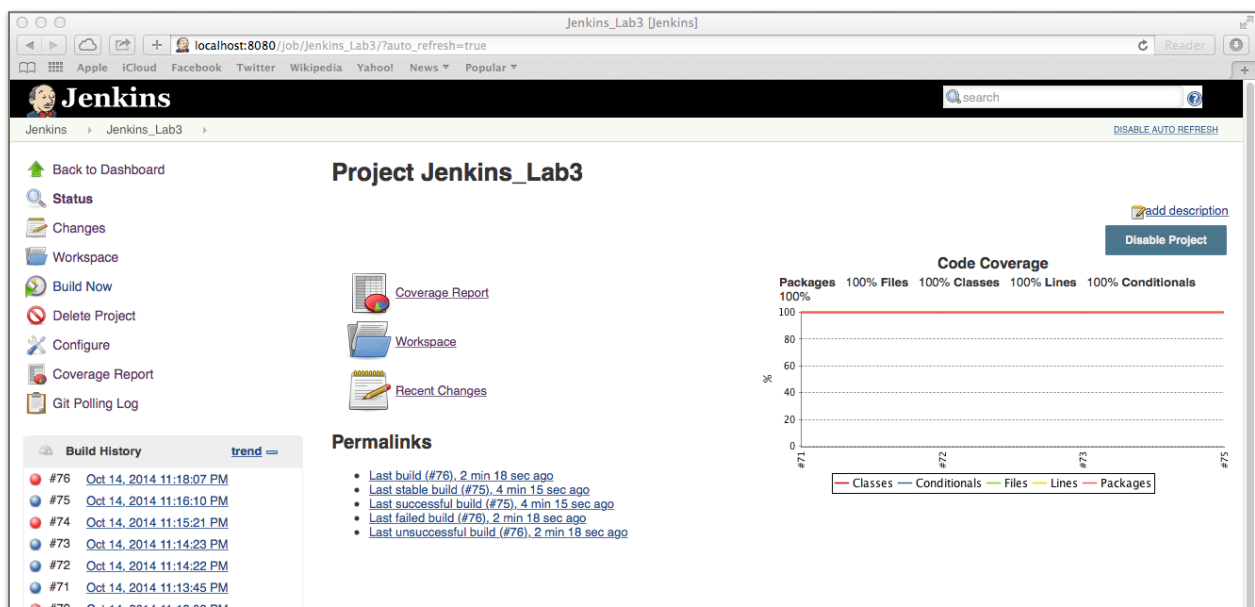


Now that the entire job has been successfully configured, we can then start seeing the status of the process whenever any code changes. Basically a code change triggers a build and the output of the build is displayed in the Jenkins UI.

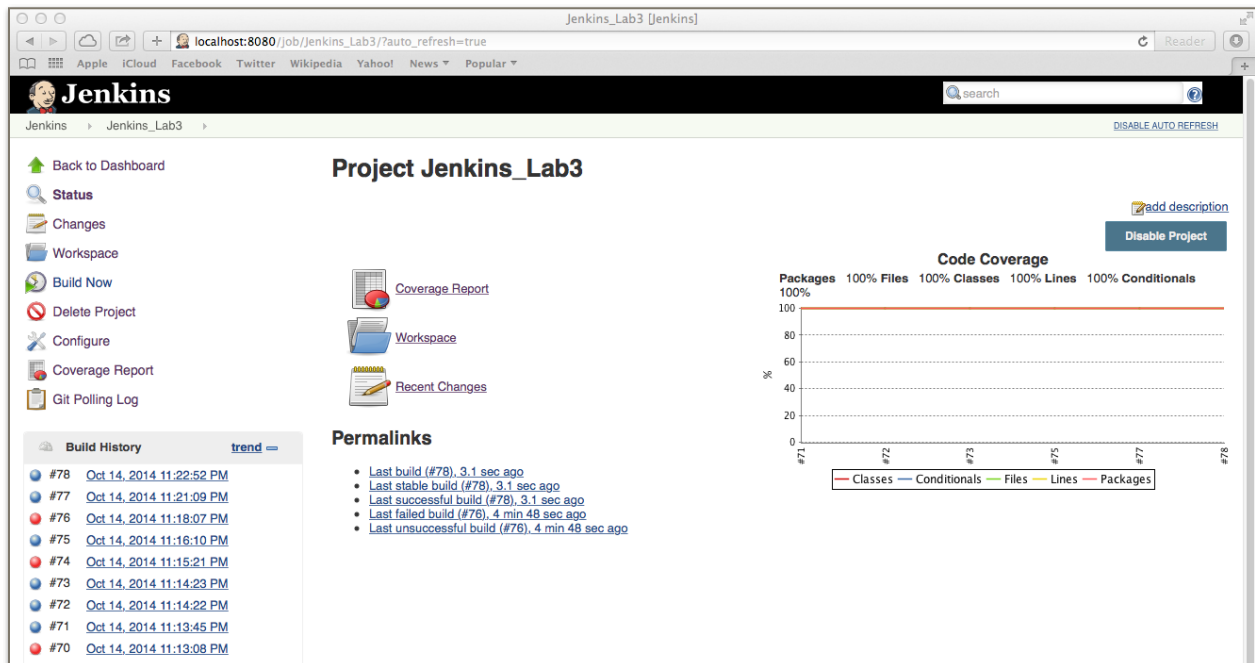
Below is an example of how it looks when the Build is Successful. In this case, the dots in the Build History section indicate the status of the individual builds. So Blue indicates success and Red indicates Failure



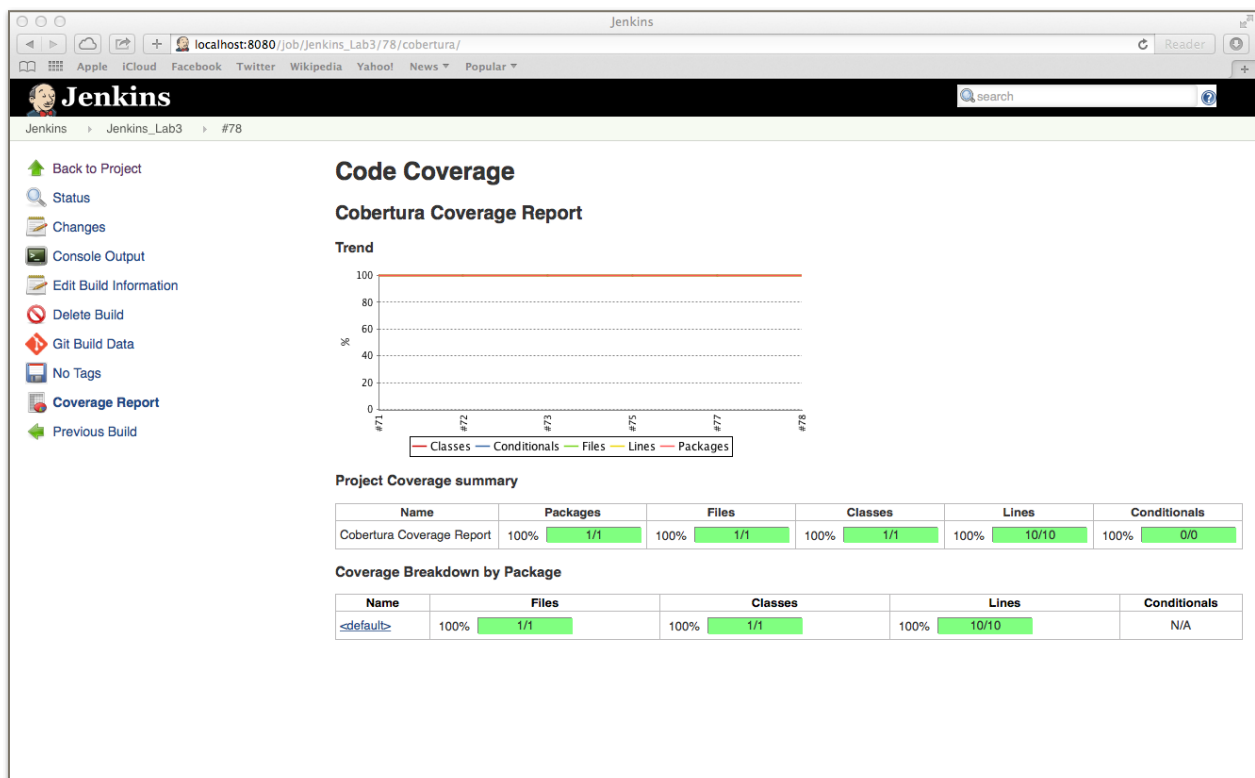
Just to indicate that the status can be Failure too, we modify the code to break the compilation process to simulate the build failing. This is captured in Jenkins as shown below:



When the error is fixed and the build runs again, then we notice that the build process again changes to blue indicating the same.



By clicking on the Coverage Report link, we can also observe the coverage of the test cases that were executed.



Finally the overall status of the Job can be observed by accessing the home page of Jenkins where it is displayed as given below:

The screenshot displays the Jenkins Dashboard interface. The browser's address bar shows 'localhost:8080'. The Jenkins logo is at the top left, and a search bar is at the top right. On the left sidebar, there are links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'Credentials'. The main content area features a table of jobs. The table has columns for 'S' (Success), 'W' (Warning), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. A single job, 'Jenkins_Lab3', is listed with a success status, a last success time of '4.4 sec - #84', and a last failure time of '8 min 7 sec - #76'. Below the table, there are links for 'Legend', 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'. On the left, there are sections for 'Build Queue' (showing 'No builds in the queue') and 'Build Executor Status' (showing '1 Idle' and '2 Idle'). The footer of the page indicates 'Page generated: Oct 14, 2014 11:26:14 PM' and 'Jenkins ver. 1.584'.

S	W	Name	Last Success	Last Failure	Last Duration
		Jenkins_Lab3	4.4 sec - #84	8 min 7 sec - #76	0.97 sec

Therefore from the various screenshots that are attached, we can see how Jenkins helps us with the Continuous Integration process.