StormTest and Jenkins/Hudson

Running StormTest Scripts from the Jenkins CI tool

S3 Group Document ID: ST-14002

Revision Date: January 2014

Product Version: 1.0

Author: Stephen Dolan

Contact us at http://www.s3group.com/tv-technology or stormtest-support@s3group.com

Contents

1 Introduction

Jenkins is an open source project providing continuous integration services for software development. It's designed to automatically execute builds, but can also be used to execute tests on those new builds or to do anything else that can be done from a script.

Note that Hudson is a fork of Jenkins (maintained by Oracle) and provides the same functionality. For the purposes of this document, Hudson and Jenkins are equivalent and either tool can be used.

2 Installation

Jenkins can be run on Windows, Linux and MacOS servers. It also supports master/slave servers, allowing a mix of environments. To use StormTest with Jenkins, you must have at least one Windows Jenkins server setup.

This document will show you how to setup a Windows master server and a Windows slave server. However, the same instructions can be followed if you already have a Linux or MacOS master server.

2.1 Choose your machines

StormTest scripts will only run on a Windows PC.

As a result, if you already have a non-Windows (i.e. Linux) Jenkins master, then you will need to choose a Windows machine to act as your Windows Jenkins slave. This can be any StormTest server but does not have to be – it could be a standalone PC.

If you do not have any Jenkins server, choose a Windows PC (such as your StormTest server) to be your Jenkins master.

You must now install the necessary software on the PC you've chosen

2.2 Get the Installers

You will need both the StormTest client software and the Jenkins software installer. Since Jenkins uses Java, you will also need the Java installer if the chosen PC does not have Java already.

The latest versions of required software can be downloaded from:

- StormTest: https://larisa.engage.s3group.com/frs/?group_id=6
- Jenkins: http://mirrors.jenkins-ci.org/windows/latest
- Java: http://java.com/getjava

2.3 Java

Install java, if necessary, following the instructions with the java installer.

2.4 StormTest

If the PC is a StormTest server, no additional StormTest software is required.

If the PC is not a StormTest server, install the StormTest Client software on it.

2.5 Jenkins Master

If you are creating a Jenkins master, run the Jenkins installer that you downloaded earlier. This will install Jenkins as a windows service.

2.5.1 Configuration

The following description assumes that your Jenkins master server has just been installed and has the default settings. If you already have a Jenkins master, please consult the administrator of the server before altering settings.

2.5.1.1 Initial setup of Jenkins master

Open the URL to the Jenkins master server. It will be of the form:

http://stormtest33:8080/

(in this case, the Jenkins master is installed on a StormTest server called 'stormtest33')

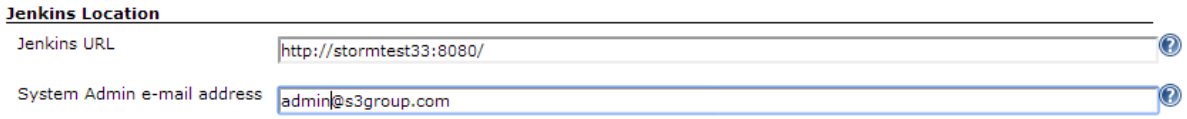
Click on 'manage jenkins' and 'configure system'.

Set the number of executors to a value that reflects the maximum number of jobs that you wish to run in parallel on this server.

This will be influenced by the type of jobs that will be run (for example build jobs could be very resource intensive).

You should also at this point give the server a 'label'. Since it is a Windows server, give it a label 'windows'. This will allow us to target StormTest jobs at Windows servers later on.

Set the URL and the system admin email address to appropriate values



Also, set the E-mail notification settings if you wish to receive emails from Jenkins.

Click on 'Save'

2.6 Jenkins Slave

Creating a slave is necessary where the master is not using Windows.

1. On the **master**, go to Manage Jenkins -> Manage Nodes
2. Choose New Node and enter a node name
3. Select Dumb Slave and press OK
4. Configure the slave:
    a. Set the number of executors
    b. Set the workspace root. This is the location on the slave where the workspace will be stored (e.g. D:\jenkins)
    c. Set labels to be 'windows'

d. Change launch method to be “Launch slave agents via java web start”
e. Press ‘OK’

5. Go to the **slave** machine
6. Open a browser to the master’s URL
7. Go to Manage Jenkins -> Manage Nodes and click on the new slave node
8. Click on the **Launch** button

9. Execute the .jnlp file that is downloaded.
10. The slave should now be running. However, every time the machine is restarted, you will need to manually re-start the slave server.
11. If you wish to run the slave as a windows service, choose that option from the ‘file’ menu:

2.7 Recommended plugins

There are a couple of Jenkins plugins that we strongly recommend that you install:

- Green Balls. Otherwise, passing jobs are marked blue, which is just confusing.
- Jenkins Workspace Cleanup Plugin

To install them:

1. Go to Manage Jenkins->Manage Plugins
2. Click on the ‘available’ tab
3. Select the plugin you wish to install

3 Creating a job

To execute a StormTest script, you need to create a Jenkins job. There are an infinite number of ways that a job can be configured. What is shown in this document is just one way which we have found to work well, but it is certainly not the only way.

3.1 Test runner

To properly reflect the result of a StormTest script in Jenkins, the return value from the script must be converted to a value understood by Jenkins. This can be done using a windows batch file. StormTest PASS value (111) is converted to a return value of 0, which Jenkins sees as a pass. Any other value is converted to 1.

Create a file called run_script.bat and paste into it the following:

```
@echo off

c:\python27\python %1 %2 %3 %4 %5 %6 %7 %8 %9

if %ERRORLEVEL% EQU 111 goto pass

:fail
exit /b 1

:pass
exit /b 0
```

Now save this file and then use Developer Suite to copy it to the *public* directory on your StormTest test repository.

3.2 Optional Test Changes

When a StormTest script runs, its output (log files, screen captures etc) go into the job workspace directory. They could be deleted or overwritten on subsequent test runs. If you choose to, you can archive the files when the job is complete for later checking.

To archive the artefacts, you must tell Jenkins which files to keep. This can be as simple as specifying all the files in the test output directory. However, since the name of this directory is different for each test run, it means that you need to edit this build step for every test you add.

A simpler way is to make sure that all tests output their logs to the same directory.

We therefore strongly recommend that all StormTest scripts call the SetDirectories() API to set the output directory. For example:

```
Stormtest.SetDirectories('output')
```

3.3 Create a job

1. Go to the Jenkins URL and click New Job
2. Give the job a name and choose 'build a free-style software project'
3. If you wish to only store a limited number of test outputs, choose 'discard old builds' and then set the number of days to keep test results or the absolute number of test results to store.
4. Choose 'restrict where this project can be run' and enter the label 'windows'. It should now tell you how many Jenkins servers are available with that label.
5. Under **Source Code Management**, select Subversion. Enter the repository URL:
   `svn://stormtest16/stormtest/public`
   where 'stormtest16' is replaced with the name of your **config** server. Note that this assumes you are only running tests in the public area of your repository. Adjust the URL if you want tests from other areas
6. Set 'local module directory' to 'public'
7. Set 'check-out strategy' to be 'emulate clean checkout'
8. (Optional): set a **build trigger** to automatically kick off the test. This could be done periodically or on checkin to subversion or could be linked to other Jenkins jobs. More detail on this feature of Jenkins is beyond the scope of this document.
9. Under **Build Environment**, select 'delete workspace before build starts'. Then select the advanced button and enter the following pattern for files to be deleted:
   `/output/**/*`
10. Under **Build** choose 'add build step->execute windows batch command'
11. In the command box, enter:
    public\run_script.bat public\tests\sometest.py –s opt1 –y opt2 10
    replacing the path, test and options above with whatever is appropriate for your test
12. Under **Post-build Actions**, choose 'archive the artefacts' and enter the files to archive as:
    `/output/**/*`
13. Click 'save'

To run the test manually, select **Build Now**.

To create another test run, you can radically shorten the process by copying the first job. When creating it, choose 'copy existing job' and enter the name of the first job you created. Then go down to the **Build** step and change the python file name to the name of the test you wish to execute. Everything else stays the same.

3.4 Finished

At this point, you are finished. You have a StormTest script running under Jenkins and can add more test scripts.

There is however more setup that can be done to make StormTest and Jenkins work better together. For information on this, please read the FAQ later in this document.

4 Using Jenkins

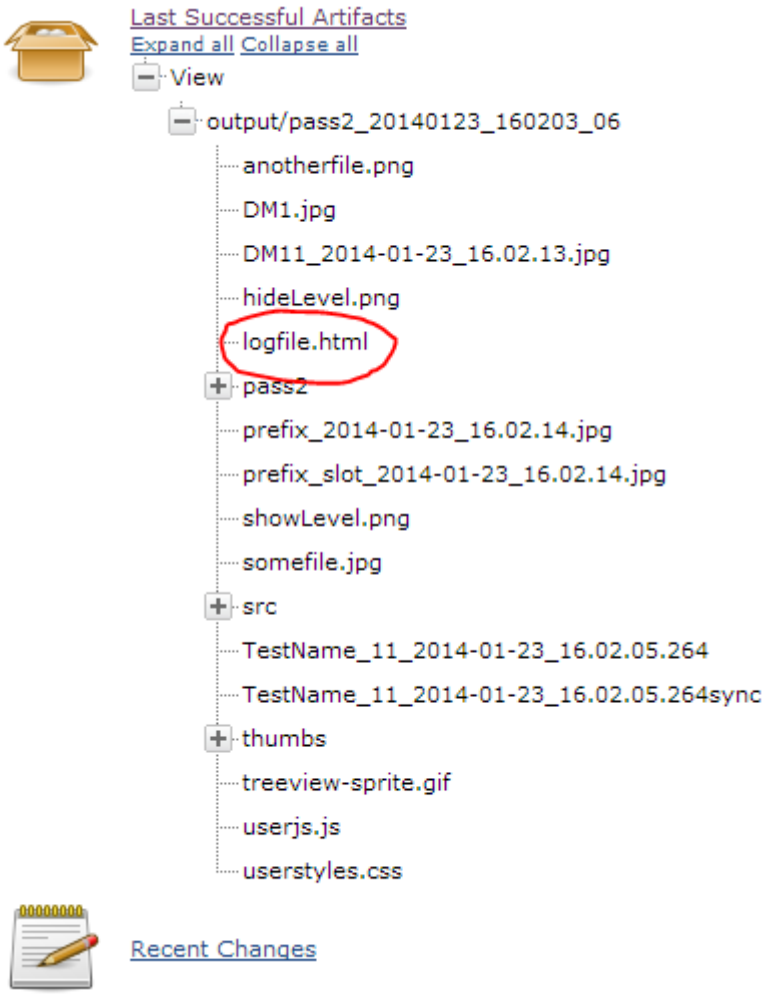
Jenkins is a dynamic expandable tool. For further documentation on how to use it, please refer to the Jenkins wiki:

https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins

To view the console output from a job, click on the instance of the job and then on 'console output'

If the test is in progress, the console output will be automatically updated while the test executes.

Once the job is complete, the archived artefacts from the last successful test run are shown on the job page



Clicking on logfile.html will allow you to view the StormTest log output.

4.1 Alternating between Jenkins and StormTest

Any tests executed with Jenkins will also appear in the StormTest Developer Suite as long as the final API call in the test is ReturnTestResult() (which should always be the case).

StormTest will automatically detect that the test was run using Jenkins and will alter the log file URL that is written to the database to point to the location of the job on the Jenkins server.

As a result, it is not possible to open the log files in the StormTest log viewer directly from Developer suite, although you can right click on open the URL to the Jenkins job from Developer Suite.

If however, you are viewing results using StormTest Dashboard, then it will locate the html log file on the Jenkins server and automatically display it in the dashboard.

5 Frequently Asked Questions

5.1 How do I run a suite of tests using Jenkins?

This can be done one of three ways.

1. You can create a different job for each and every test in your suite and link them together using build triggers.
2. You can create a 'master' script that will run all the tests in the test suite as a single Jenkins job.
3. There is a 3rd option also which requires changes to the tests themselves and is described below.

Option 2 is quicker to configure from a Jenkins point of view, but has 2 disadvantages:

1. To change the tests that are executed, you need to edit a python file and check it into svn.
2. The result shown in Jenkins reflects the entire test suite run, so you have to somehow aggregate multiple test result into one. This makes the dashboard and trends in Jenkins much less useful

Option 1 may take longer to setup, but it gives better visibility of results and is relatively easy to add and remove tests from the test run. We will describe options 1 and 2 here and then go into more detail on the third option.

5.1.1 Option 1 – multiple jobs

To add a test to the test run for option 1, you simply need to edit the job and click 'build after other projects are built' in the **Build Triggers** section. Then choose the job that will trigger this job.

(in the above capture, when stormtest_job1 completes, it will trigger the execution of the job being edited)

This allows you to create a 'pipeline' of jobs, running them in series.

Alternatively, all jobs can be triggered by a single 'master' job (which will enter them all into the queue) and then the number of executors will determine how many tests are run in parallel at a time.

5.1.2 Option 2 – single job

This requires a 'master' python file which spawns processes to run each test in the suite. The creation of this file is beyond the scope of this document

5.1.3 Option 3 – junit test results

There is a third option that is more 'Jenkins friendly'. This option is more complex to setup and requires your tests to be altered, however it does give you much more interesting results in Jenkins.

It relies on your test producing output in junit xml format. This is easiest to achieve by using the junit-xml python module.

5.1.3.1 Installing junit-xml

The junit-xml module will need to be installed on all your Jenkins server as well as the PC you are developing your tests on.

Before it can be installed, you must first install the 'distribute' utility. To do this, get the file at http://python-distribute.org/distribute_setup.py and save it somewhere. Then run it with python:

```
C:\temp>python distribute_setup.py
Downloading http://pypi.python.org/packages/source/d/distribute/distribute-0.6.49.tar.gz
Extracting in c:\docume~1\stormt~1\locals~1\temp\1\tmp3nwu44
Now working in c:\docume~1\stormt~1\locals~1\temp\1\tmp3nwu44\distribute-0.6.49
Installing Distribute
Before install bootstrap.
Scanning installed packages
No setuptools distribution found
running install
running bdist_egg
running egg_info
writing distribute.egg-info\PKG-INFO
writing top-level names to distribute.egg-info\top_level.txt
writing dependency_links to distribute.egg-info\dependency_links.txt
writing entry points to distribute.egg-info\entry_points.txt
reading manifest file 'distribute.egg-info\SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'distribute.egg-info\SOURCES.txt'
installing library code to build\bdist.win32\egg
…
byte-compiling build\bdist.win32\egg\_markerlib\__init__.py to __init__.pyc
creating build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\PKG-INFO -> build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\SOURCES.txt -> build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\dependency_links.txt -> build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\entry_points.txt -> build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\top_level.txt -> build\bdist.win32\egg\EGG-INFO
copying distribute.egg-info\zip-safe -> build\bdist.win32\egg\EGG-INFO
creating dist
creating 'dist\distribute-0.6.49-py2.7.egg' and adding 'build\bdist.win32\egg' to it
removing 'build\bdist.win32\egg' (and everything under it)
Processing distribute-0.6.49-py2.7.egg
creating c:\python27\lib\site-packages\distribute-0.6.49-py2.7.egg
Extracting distribute-0.6.49-py2.7.egg to c:\python27\lib\site-packages
Adding distribute 0.6.49 to easy-install.pth file
Installing easy_install-script.py script to C:\Python27\Scripts
```

```
Installing easy_install.exe script to C:\Python27\Scripts
Installing easy_install.exe.manifest script to C:\Python27\Scripts
Installing easy_install-2.7-script.py script to C:\Python27\Scripts
Installing easy_install-2.7.exe script to C:\Python27\Scripts
Installing easy_install-2.7.exe.manifest script to C:\Python27\Scripts

Installed c:\python27\lib\site-packages\distribute-0.6.49-py2.7.egg
Processing dependencies for distribute==0.6.49
Finished processing dependencies for distribute==0.6.49
After install bootstrap.
Creating C:\Python27\Lib\site-packages\setuptools-0.6c11-py2.7.egg-info
Creating C:\Python27\Lib\site-packages\setuptools.pth

C:\temp>
```

Now, install junit-xml using the easy_install command.

```
C:\temp>\Python27\Scripts\easy_install junit-xml
Searching for junit-xml
Reading http://pypi.python.org/simple/junit-xml/
Best match: junit-xml 1.3
Downloading https://pypi.python.org/packages/source/j/junit-xml/junit-xml-
1.3.tar.gz#md5=2b95e9a2a7ebf240678271f33fcd8d3e
Processing junit-xml-1.3.tar.gz
Writing c:\docume~1\stormt~1\locals~1\temp\1\easy_install-drr8i4\junit-xml-
1.3\setup.cfg
Running junit-xml-1.3\setup.py -q bdist_egg --dist-dir
c:\docume~1\stormt~1\locals~1\temp\1\easy_install-drr8i4\junit-xml-1.3\egg-
dist-tmp-ttivgz
zip_safe flag not set; analyzing archive contents...
Adding junit-xml 1.3 to easy-install.pth file

Installed c:\python27\lib\site-packages\junit_xml-1.3-py2.7.egg
Processing dependencies for junit-xml
Finished processing dependencies for junit-xml
```

5.1.3.2 Script Updates

When reporting results, you must split your tests into 'testsuites' and 'testcases'. How these relate to your actual scripts is up to you.

For example, each script could be a 'testsuite' and the steps in the test could each be a 'testcase'. Since this is closely matched to how StormTest scripts are often structured, we will describe using the module in this way. However, it is up to the specific project to decide what best reflects the way they wish to track results.

At beginning of test, import the module and create a list to collect the test steps:

```
from junit_xml import TestSuite, TestCase
test_steps = []
```

After each step, create a TestCase:

```
tc = TestCase('Step number '+str(i), filename, end-start)
```

where the first argument is the name of the step, the second is the 'class name' (in this case we've used the name of the script file) and the third is the amount of time the step took to execute. This is optional, but can easily be calculated by putting `start=time.time()` at the start of the step and `end=time.time()` at the end.

If the step failed, add a failure message:

```
tc.add_failure_info('step failed for some reason')
```

then append the step to the list:

```
test_steps.append(tc)
```

At the completion of the test, create a TestSuite and write the contents to an xml file:

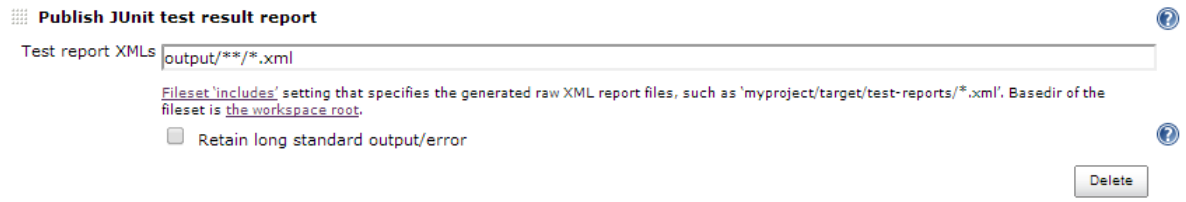
```
if __name__ == '__main__':
    ret = main(sys.argv[1:])

    ts = [TestSuite("my test suite", test_steps)]
    f = open('output/'+filename+'.xml','w')
    TestSuite.to_file(f, ts, prettyprint=False)
    f.close

    ST.ReturnTestResult(ret)
```

Here we are again using the name of the test script as the name of the output xml file.

Those are the only changes needed to the tests.

5.1.3.3 Jenkins job updates

In **Post-build Actions**, add a 'publish junit test result report' step and tell it where to get the xml files.

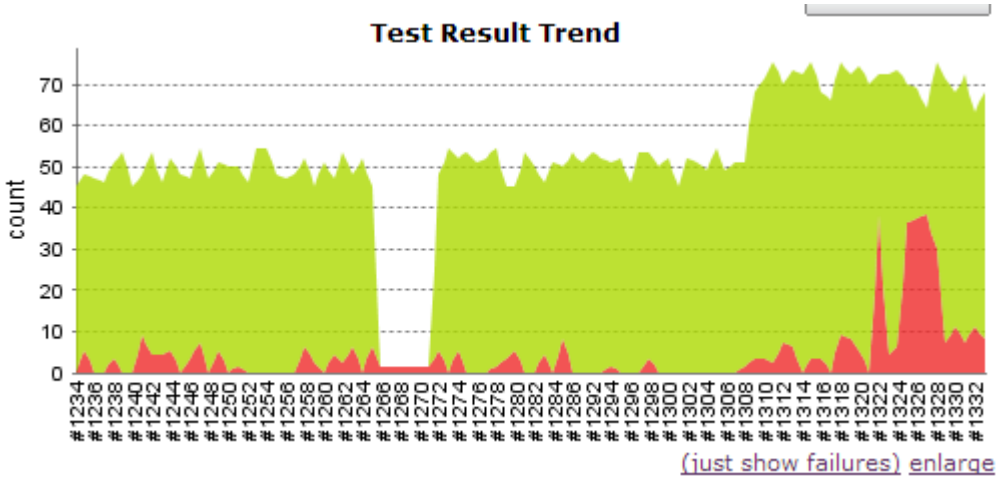


5.1.3.4 Run multiple test scripts

Now its possible to run multiple test scripts in a single job. Just add as many 'windows batch command' steps as you like, one for each test script.

Each test will generate its own xml file and at the end of the job, Jenkins will find all the xml files, aggregate them together and give you a nice report.

In this example, one job is running two script, each with a number of steps:



The trend shows you in aggregate how many steps were run and how many failed.

Then for a single run, you can see the spread of failures, the difference versus the last run (for example 2 more failures than last run) and the specific steps that failed.

Test Result

8 failures (-3)

68 tests (+5)
Took 30 sec.
add description

All Failed Tests

Test Name	Duration	Age
report_junit_one.py.Step number 3	0.45 sec	1
report_junit_one.py.Step number 7	0.36 sec	1
report_junit_one.py.Step number 9	0.28 sec	1
report_junit_two.py.Loop number 27	0.48 sec	1
report_junit_two.py.Loop number 35	0.36 sec	1
report_junit_two.py.Loop number 43	0.37 sec	1
report_junit_two.py.Loop number 45	0.5 sec	1
report_junit_two.py.Loop number 17	0.54 sec	4

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
report_junit_one	7 sec	3	-2	0		12	+1	15	-1
report_junit_two	23 sec	5	-1	0		48	+7	53	+6

Page generated: Jan 23, 2014 5:51:09 PM REST API Jenkins ver. 1.547

It can be seen from this screen also how long a specific test step has been failing (i.e. the 'age' of the failure).

5.2 How do I decide what slots/DUTs my test runs on?

This is the big drawback of Jenkins relative to the StormTest scheduler. Jenkins knows nothing of slots/DUTs, so you need to build this logic into your tests.

There are a number of options:

1. Hardcode the slot to be used into the test script (not recommended)
2. Pass the slot to the test on the command line, which means you have to hardcode the slot into the job definition.
3. Store slot allocations into an external file, e.g. an excel file, which is read by the test script. Slots could be allocated to a test run based on the job name. Since the job name is available as an environment variable, the test could lookup its slot in the file based on this variable. This could be quite unwieldy to manage however.
4. Some other custom resource manager that would allocate the test resources to the scripts. This is beyond the scope of this document

It is important to note that using Jenkins for your test execution is most likely going to require you to make much more static allocation of your test resources (slot) compared to using the built-in StormTest scheduler.

5.3 Can I have my Jenkins master on a linux server?

Yes, there is no problem with this as long as there is at least one Windows slave, configured as detailed earlier in this document.

5.4 Can I install a Jenkins slave on the StormTest config server?

It is certainly possible to do this. However since the config server is critical to the operation of the StormTest facility, it is not recommended to run a slave on the config server. If you must do this, then we recommend that the number of executors for that slave be set to a small number (2-3).

5.5 Can I trigger test runs when a file is checked into svn?

Yes, this is just another type of build trigger. You need to select the 'Poll SCM' option under **Build Triggers**. Then tell Jenkins how often to check for subversion changes. This follows a 'cron-like' format.

For example:

H/5 * * * *

Will poll the subversion server every 5 minutes.

You can also trigger build periodically in a similar way. So to build every day at 8pm, click 'Build periodically' and enter:

H 20 * * *

Alternatively, you can use the shortcuts @yearly, @annually, @monthly, @weekly, @daily, @midnight, and @hourly

5.6 Can I use my own test repository?

The instructions given in this document will work with the built-in StormTest SVN repository. However, Jenkins has plugins for many other varieties of source code management tool. For more information on using them, please refer to the Jenkins wiki:

https://wiki.jenkins-ci.org/display/JENKINS/Plugins#Plugins-Sourcecodemanagement

5.7 How do I get an email notification from Jenkins?

The built-in email notification feature in Jenkins will notify designated email addresses when there is a failure.

The email will contain a copy of the console output from the job and a link to the job.

Another email will then be sent when the job starts passing again.

There are more configurable email plugins available from the Jenkins website, such as email-ext, which allows the body of the email and the triggers to be configured to a much higher degree.

5.8 I installed the slave as a windows service. How do I uninstall it?

When you chose to install the slave as a service, a number of files will have been created in the location that you configured as the filesystem root for that Jenkins slave. In this location, run the following to uninstall the service:

```
D:\Jenkins>jenkins-slave.exe uninstall
```

Then delete the workspace.