Building a software project

Table of Contents

Jenkins can be used to perform the typical build server work, such as doing continuous/official/nightly builds, run tests, or perform some repetitive batch tasks. This is called "free-style software project" in Jenkins.

Setting up the project

Go to Jenkins top page, select "New Job", then choose "Build a free-style software project". This job type consists of the following elements:

- optional SCM, such as CVS or Subversion where your source code resides.
- optional triggers to control when Jenkins will perform builds.
- some sort of build script that performs the build (ant, maven, shell script, batch file, etc.) where the real work happens
- optional steps to collect information out of the build, such as archiving the artifacts and/or recording javadoc and test results.
- optional steps to notify other people/systems with the build result, such as sending e-mails, IMs, updating issue tracker, etc.

For more details, click the ? icons in the configuration page.

Jenkins Set Environment Variable

Jenkins sets some environment variables that are available to shell scripts, Windows batch files, Ant and Maven^#1^ files that are executed by Jenkins. A list of environment variables and how they are used are shown #below.

Builds for Non-Source Control Projects

There is sometimes a need to build a project simply for demonstration purposes or access to a SVN/CVS repository is unavailable. By choosing to configure the project as "None" under "Source Code Management" you will have to:

1. Build the Project at least once, (it will fail), but Jenkins will create the structure jenkins/workspace/PROJECTNAME/
2. Copy the project files to jenkins/workspace/PROJECTNAME/
3. Build again and configure appropriately

Jenkins Set Environment Variables

When a Jenkins job executes, it sets some environment variables that you may use in your shell script, batch command, Ant script or Maven POM [#1]. The following table contains a list of all of these environment variables.

Environment Variable	Description
BUILD_NUMBER	The current build number, such as "153"
BUILD_ID	The current build id, such as "2005-08-22_23-59-59" (YYYY-MM-DD_hh-mm-ss, defunct since version 1.597)
BUILD_URL	The URL where the results of this build can be found (e.g. http://buildserver/jenkins/job/MyJobName/666/)
NODE_NAME	The name of the node the current build is running on. Equals 'master' for master node.
JOB_NAME	Name of the project of this build. This is the name you gave your job when you first set it up. It's the third column of the Jenkins Dashboard main page.
BUILD_TAG	String of `jenkins-${JOB_NAME}-${BUILD_NUMBER}`. Convenient to put into a resource file, a jar file, etc for easier identification.
JENKINS_URL	Set to the URL of the Jenkins master that's running the build. This value is used by Jenkins CLI for example

EXECUTOR_NUMBER	The unique number that identifies the current executor (among executors of the same machine) that's carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1.
JAVA_HOME	If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to have $JAVA_HOME/bin.
WORKSPACE	The absolute path of the workspace.
SVN_REVISION	For Subversion-based projects, this variable contains the revision number of the module. If you have more than one module specified, this won't be set.
CVS_BRANCH	For CVS-based projects, this variable contains the branch of the module. If CVS is configured to check out the trunk, this environment variable will not be set.
GIT_COMMIT	For Git-based projects, this variable contains the Git hash of the commit checked out for the build (like ce9a3c1404e8c91be604088670e93434c4253f03) (all the GIT_* variables require git plugin)
GIT_URL	For Git-based projects, this variable contains the Git url (like git@github.com:user/repo.git or [https://github.com/user/repo.git])
GIT_BRANCH	For Git-based projects, this variable contains the Git branch that was checked out for the build (normally origin/master)

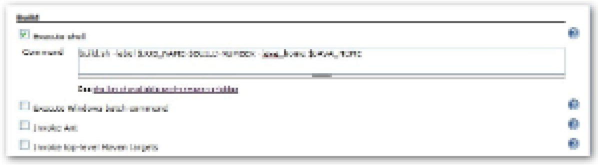
Promoted Build Plugin Environment Variables

If you are using the Promoted Build Plugin, you will have access to the following environment variables. This allows you to access information about your Jenkins build since certain environment variables stated above (such as `BUILD_TAG` now refer to the Promoted Build Plugin's job.

Environment Variable	Replaces	Description
PROMOTED_URL	BUILD_URL	The URL of the **original** Jenkins job that is involved with the promotion. BUILD_URL now refers to the Promotion's URL
PROMOTED_JOB_NAME	JOB_NAME	The name of the **original** Jenkins job that is involved with the promotion. JOB_NAME now refers to the Promotion's job's name
PROMOTED_NUMBER	BUILD_NUMBER	The Build Number of the job being promoted. BUILD_NUMBER now refer's the the Promotion Number
PROMOTED_ID	BUILD_ID	The Build ID (ex. "2005-08-22_23-59-59" (YYYY-MM-DD_hh-mm-ss)) of the **original** Jenkins job. BUILD_ID now refer's to the Promotion's build ID.

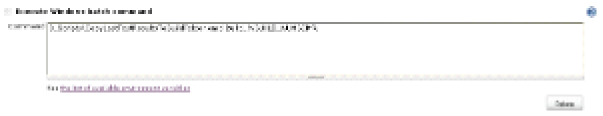
Shell Scripts and Windows Batch Commands

If you're using a shell script to do your build, you can either put these environment variables directly into your shell scripts, or call them as parameters in your shell script. Below is an example how this can be done:



```
[illegible]
```

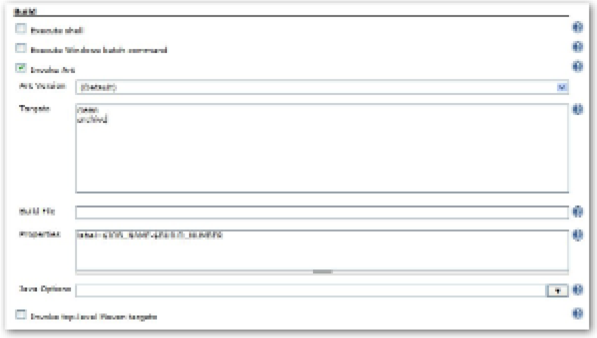
If you are executing a Windows Batch Command, the variables should be referenced using the %VARIABLE_NAME% pattern. For example:



```
[illegible]
```

Ant Scripts

If you're using an Ant script to do your build, you may include environment variables in property settings. Click on the ***Advanced...*** button just below where you put the Ant targets you want to build. This will display the *Properties* box. Below is an example how to use the Properties box to set Ant properties with Jenkins Environment variables:



Invoke Ant ✓

Ant Version: (Default)

Targets:
clean
archive

Build File:

Properties: label=${JOB_NAME}-${BUILD_NUMBER}

Java Options:

As an alternative, you can use the Environmental prefix to pull in all environmental variables as properties right inside your `build.xml` file. Below is an example how to set the property "label" to include the Project Name and the Build Number:

```
<property environment="env"/>
<property name="label" value="${env.JOB_NAME}-${env.BUILD_NUMBER}"/>
```

Configuring automatic builds

Builds in Jenkins can be triggered periodically (on a schedule, specified in configuration), or when source changes in the project have been detected, or they can be automatically triggered by requesting the URL:

```
http://YOURHOST/jenkins/job/PROJECTNAME/build
```

This allows you to hook Jenkins builds into a variety of setups. For more information (in particular doing this with security enabled), see Remote access API.

Builds by source changes

You can have Jenkins poll your Revision Control System for changes. You can specify how often Jenkins polls your revision control system using the same syntax as crontab on Unix/Linux. However, if your polling period is shorter than it takes to poll your revision control system, you may end up with multiple builds for each change. You should either adjust your polling period to be longer than the amount of time it takes to poll your revision control system, or use a post-commit trigger. You can examine the *Polling Log* for each build to see how long it took to poll your system.

Alternatively, instead of polling on a fixed interval, you can use a URL trigger (described above), but with `/polling` instead of `/build` at the end of the URL. This makes Jenkins poll the SCM for changes rather than building immediately. This prevents Jenkins from running a build with no relevant changes for commits affecting modules or branches that are unrelated to the job. When using /polling the job must be configured for polling, but the schedule can be empty.

Using a post-commit trigger in CVS

With some revision control systems, like Subversion, polling is very quick. Subversion can poll your project in a few seconds to see if there are any changes. In some revision control systems like CVS, polling can take quite a long time.

In this case, you should probably use a post-commit hook to trigger the build. In CVS, you can add a post commit trigger to the `$CVSROOT/loginfo` file. To edit this file, check out the CVSROOT project, edit the file, and then do a commit. Don't edit the file directly.

The `loginfo` file consists of two entries. The first is the repository, and the second is the post-commit hook to run. If you name your Jenkins projects as *<project>-<branch>*, you can use the following shell script trigger:

```
#! /bin/bash
/usr/bin/sed -n '/^  *Tag:/s/.*: *//p' | while read branch
do
    #
    #  You need to set these
    #
    wgetCmd=/usr/bin/wget           #Location of wget command
    logName=/usr/home/cvs/log.txt   #Logfile name
    projectBase=jenkins             # First part of the Jenkins project name
    hudsonUrl="http://hudson:8080"  #URL to trigger Jenkins
    triggerString="BUILD"           #String to trigger builds

    hudsonJob="$cvsProject-$branch"

    #
    # Possible exceptions to Jenkins Name Rule
    #
    if [ "$branch" == "REL_1_0_2" ]
    then
        hudsonJob="$projectBase-DEV"
    fi

    $wgetCmd -q $hudsonUrl/job/$hudsonJob/build?token=$triggerString
    echo "$wgetCmd -q $hudsonUrl/job/$hudsonJob/build?token=$triggerString" >> $logName
    echo "----------------------------------------------------" >> $logName
done
```

Builds by e-mail (sendmail)

If you have the root account of your system and you are using sendmail, I found it the easiest to tweak `/etc/aliases` and add the following entry:

```
jenkins-foo: "|/bin/wget -o /dev/null http://YOURHOST/jenkins/job/PROJECTNAME/build"
```

and then run "newaliases" command to let sendmail know of the change. Whenever someone sends an e-mail to "jenkins-foo@yoursystem", this will trigger a new build. See this for more details about configuring sendmail.

Builds by e-mail (qmail)

With qmail, you can write `/var/qmail/alias/.qmail-jenkins` as follows:

```
|/bin/wget -o /dev/null http://YOURHOST/jenkins/job/PROJECTNAME/build"
```

[1] Maven requires that you include the parameter as part of the build goals.
Example Jenkins configuration for the Maven "Goals" field: clean install -DBUILD_NUMBER=${BUILD_NUMBER}