# Migrating Build Jobs

There are times when you need to move or copy Jenkins build jobs from one Jenkins instance to another, without copying the entire Jenkins configuration. For example, you might be migrating your build jobs to a Jenkins instance on a brand new box, with system configuration details that vary from the original machine. Or you might be restoring an old build job that you have archived.

As we have seen, Jenkins stores all of the data it needs for a project in a subdirectory of the jobs directory in your Jenkins home directory. This subdirectory is easy to identify—it has the same name as your project. Incidentally, this is one reason why your project names really shouldn’t contain spaces, particularly if Jenkins is running under Unix or Linux—it makes maintenance and admin tasks a lot easier if the project names are also well-behaved Unix filenames.

You can copy or move build jobs between instances of projects simply enough by copying or moving the build job directories to the new Jenkins instance. The project job directory is self-contained—it contains both the full project configuration and all the build history. It is even safe enough to copy build job directories to a running Jenkins instance, though if you are also deleting them from the original server, you should shut this one down first. You don’t even need to restart the new Jenkins instance to see the results of your import—just go to the Manage Jenkins screen and click on Reload Configuration From Disk. ...

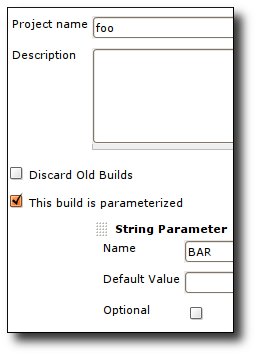
**Jenkins Parameterized Builds**

Sometimes, it is useful/necessary to have your builds take several "parameters." Consider the following use case:

* You set up a test job on Jenkins, and it accepts a distribution bundle as a parameter and perform tests against it. You want to have developers do local builds and let them submit builds for test execution on Jenkins. In such a case, your parameter is a zip file that contains a distribution.
* Your test suite takes so much time to run that in normal execution you can't afford to run the entire test cycle. So you want to control the portion of the test to be executed. In such a case, your parameter is perhaps a string token that indicates that test suite to be run.

The parameter are available as environment parameters. So e.g. a shell (*$FOO*, *%FOO%*) or Ant ( ${env.FOO} ) can access these values.

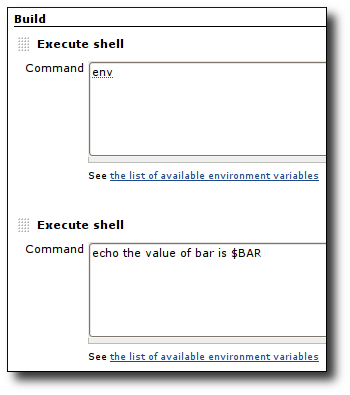
**Defining Parameters**

First, you need to define parameters for your job by selecting "This build is parameterized", then using the drop-down button to add as many parameters as you need.  


There are different parameter types available, and it is extensible, too. The way parameters take effect is also different depending on the parameter type you choose.

**String parameter**

String parameters are exposed as environment variables of the same name. Therefore, a builder, like Ant and Shell, can use the parameters. Continuing the above example, the following is a simple example:

1. Reference parameter by name in builder. I'm using the "env" command to show the variable, followed by an echo statement to demonstrate referencing the value:  
   
2. Run build and observe output toward the bottom of the log (some vars removed for security and brevity):
3. started
4. [workspace] $ /bin/sh -xe /opt/apache-tomcat-6.0.14/temp/jenkins20854.sh
5. + env
6. BAR=bat
7. ...
8. [workspace] $ /bin/sh -xe /opt/apache-tomcat-6.0.14/temp/jenkins20855.sh
9. + echo the value of bar is bat
10. the value of bar is bat
11. finished: SUCCESS

Ant works equally well. In the Properties section of the Ant builder, define a build property like:

my.prop=${env.BAR}

Note that because of the case sensitivity difference of environment variables in Windows and Unix, all the environment variables added by parameters are in upper case.

**File parameter**

File parameter allows a build to accept a file, to be submitted by the user when scheduling a new build. The file will be placed inside the workspace at the known location after the check-out/update is done, so that your build scripts can use this file.

**Define Custom Parameter Types**

A plugin can define custom parameter types. See ParameterDefinition for the starting point.

**Launching a build with parameters**

|  |  |
| --- | --- |
| https://wiki.jenkins-ci.org/images/icons/emoticons/warning.gif | **Parameters are Case Sensitive!** When passing parameters through the URL, casing is important! For example *token=TOKEN&MESSAGE=yo* will not work if the job defines the parameter as *Message*. |

* A build can be started just by POSTing to

http://server/job/myjob/buildWithParameters?PARAMETER=Value

* All parameters need to be properly URL-escaped. To use with wget, quote the URL on the command line too.
* The parameter *delay=0sec* can be added to start the build immediately.
* To use a Run Parameter, the value should be in the format *jobname#buildNumber* (eg. "&MyRunParam=foo-job%2399" for foo-job #99)
* If you are using an authorization token to trigger the builds (Job -> Configure -> 'Build Triggers' -> 'Trigger builds remotely (e.g., from scripts)'), you can access:

http://server/job/myjob/buildWithParameters?token=TOKEN&PARAMETER=Value

(Note that the & will need to escaped, or the whole URL quoted, if using a Unix shell script.)

**Limitations**

Currently the following are the known problems:

* When build triggers are used to start a build, there's no way to pass parameters. This includes SCM polling, downstream builds, and periodic builds. Instead, the specified default values will be used for string, boolean and choice parameters.

**Open issues**

How can i pass the configured parameter to the maven execution of the build?  
E.g. I've got a maven build and a specific profile should be given as a String parameter in the Jenkins build.