Building Jenkins

Build Jenkins following the contribution directions.

Configure work environment

To check out Jenkins, follow the steps below. Then run Maven build at least once to have it generate additional source files:

```
// check out the workspace
$ git clone https://github.com/jenkinsci/jenkins.git
// Download dependencies and build with Maven
// Note: mvn install may be a prerequisite for other mvn targets
$ cd jenkins
$ mvn -Plight-test install
```

The git repository mentioned above is where the core of Jenkins is located, i.e. not the plugins. In the root of the repository is a file called **changelog.html**, which you should update if/when you become a committer.

The first build will take a while, because it has to download all needed dependencies and plugins from Maven central.
If it succeeds with `BUILD SUCCESSFUL` at the end, you are good to go.
At the end you should have war/target/jenkins.war, which is the Jenkins you just built.

> Building and running tests can cause permgem errors. If that is the case, try to increase the JVM limit for Maven targets, by setting the following environment variable:
> $ export MAVEN_OPTS="-XX:MaxPermSize=128m"

Eclipse

You can *either* use M2Eclipse/m2e to import the Maven projects directly *or* create Eclipse project files from the command line and then import the projects into Eclipse. There's no consensus in the community as to which approach is better. Experienced Maven users seem to prefer m2eclipse, but people new to Maven seem to find the latter easier and more predictable. For the latter do:

```
// generate .project and .classpath for Eclipse
$ mvn -DdownloadSources=true eclipse:eclipse
```

There is a step-by-step guide to working with Eclipse once you've run the above commands.

NetBeans/Intellij IDEA

NetBeans (6.7+) users can just open the POM directly, with File -> Open Project and navigate to the `jenkins` folder. IntelliJ IDEA users can directly open a Maven POM.

> If your IDE complains that 'Messages' class is not found, they are missing because they are supposed to be generated. Run a Maven build once and you should see them all. If that doesn't fix the problem, make sure your IDE added `target/generated-sources` to the compile source roots.

> If IntelliJ IDEA fails to compile with error messages "XXX clashes with package of same name", this is caused by issue IDEA-80331. JetBrains suggest to add `resources:*.groovy` to Settings | Compiler | Resource Patterns as a workaround.

> If your are using IntelliJ IDEA and Microsoft Windows then you should run Intellij IDEA as administrator. Otherwise some tests which are creating Symlink will fail.;

Debugging Jenkins

Maven Jetty plugin offers a convenient debugging environment, but for a few reasons we maintain a modified version of the plugin under a different name, so the plugin name is different but the configuration parameters are the same.

First you need to add org.jenkins-ci.tools as a plugin group to your maven settings.xml file, as described in Plugin tutorial#SettingUpEnvironment.

Then do the following to run Jenkins under the debugger:

```
# To change the port run mvnDebug jenkins-dev:run -Dport=9090
$ cd war
$ mvnDebug jenkins-dev:run
```

This will launch maven with the debugger port for 8000. You can connect to this port from your IDE by using the remote debug feature. After you connect the debugger, open http://localhost:8080/ in your browser. Once this starts running, keep it running. Jetty will pick up all the changes automatically.

1. When you make changes to view files in core/src/main/resources, just hit F5 in your browser to see the changes.
2. When you make changes to static resources in war/resources, just hit F5 in your browser to see the changes.
3. When you change Java source files. Compile them in your IDE and Jetty should pick that up. You don't need to run mvn at all for this.
4. NetBeans users can just hit Debug with main/war open.

If you are developing plugins and want to test them see Plugin Structure#Testing With Jenkins

Other Tips

1. Consider running Maven like mvn -o ... to avoid hitting repositories every time. This will make various operations considerably faster.
2. When Maven complains about something, try "cd $JENKINS; mvn clean install".
3. The above mentioned profile (-Plight-test) does only run the fairly quick unit tests and leaves out the integration test in the test-harness module. If you want to skip all tests (not recommended!), run Maven with "-Dskip-test-harness".
4. If you have trouble that isn't addressed here, please send email to the Jenkins dev list. The developers all read that list frequently. **You might not get an answer for a long time - or not at all - if you post your questions as comments to this page.**
5. Setting up a local Maven repository manager can considerably speed up your Jenkins builds, and make life with Maven more pleasant in general. Please click here for Artifactory or here for Nexus instructions.
6. Users can also use Artifactory for easy and rapid setup for hackatons and meetups.
7. In development & test modes, the setup wizard is skipped. To force the setup wizard to run, use this system property: `jenkins.install.runSetupWizard=true`

Credits

The sponsor statement from YourKit Java Profiler, which gave us a free license for the Hudson project.