Installation: Installing OSP/Sakai v2.3

NOTE: for a more up-to-date step-by-step guide on installing a modern Sakai these days, check out the official Development Environment Setup Walkthrough at the Sakai Confluence site. In particular, Sakai version 2.5.x and beyond all use Maven 2 ("mvn"). Sakai 2.4.x and older versions of Sakai use Maven 1 ("maven"). Also, the JDK compatibility package mentioned below is no longer required for recent versions of Sakai.

Still, you may still find the HOWTO here helpful, as it includes newbie-friendly rationale and context not found in the official version (plus, we're told that our writing is a bit different from most, in that it's "written in English, for humans" -- we hear that a lot).

Sakai is an open-source **CLE** (collaborative learning environment) or **LMS** (learning management system) similar to Blackboard, but without the licensing headaches. OSP is an **electronic portfolio** toolkit built into Sakai -- so, once you install Sakai, you've got an electronic portfolio system ready to roll! Here's how you can get it running as a sandbox so you can learn more about what it can do--
Prerequisites:

To get SAKAI/OSP running, you'll need:

- Java Developer Kit, or "JDK" for short (Sakai and OSP are written in Java)
- Maven (this builds the system)
- Tomcat (it serves up the web pages)
- Subversion, or "svn" for short (it pulls the source code from sakaiproject.org)
- Mysql (somewhere to store all the data)
- and, of course, Sakai itself

Note that versions change when patches are applied and bugs are fixed, so we'll try to avoid version-specific info when possible. For the exact versions we're using as we write this (January 2007) see footnotes at the very bottom.

Ready?
Java:
We download the "netbeans" version. Don't be surprised if this part of your experience is confusing -- which link to follow? You're not alone (Java is for programmers, don't forget, so they build the website for a clientele different from normal mortals)...

Here's the most generic path to a healthy JDK that we could find -- instad of giving you a final URL which could change weekly, here's a conceptual path to get to the stuff you need (and the website may change too, of course, but they're likely to keep the general idea the same):

- Visit http://java.sun.com/ -- now, if you think you can 1) where to go and 2) which one is the right download for you, feel free and dive right in! We find that there are enough varieties and versions to select among that it's easy to get lost, so:
- Click "Downloads":

- Click "Java SE":

- Click "Get the JDK download" line:

- Check out the license agreement and click "Accept" -- though it looks like you can skip this part and go straight to your download, that won't work! :)

- Pick the instance appropriate for your platform (we use Debian/GNU Linux) and click it to start your download:Linux
  - Solaris SPARC
  - Solaris x86
  - Windows

  For Linux, we use the non-rpm *.bin style.

Now that we've got it downloaded, we plop it into /usr/local:

```
# ls -lF /usr/local
total 48460
drwxrwsr-x   2 root    staff     4096 Jun 30 18:14 bin/
drwxrwsr-x   2 root    staff     4096 Nov 18  2005 games/
drwxrwsr-x   2 root    staff     4096 Nov 18  2005 include/
-rw-r--r--   1 root    staff 49518537 Oct 25 16:21 jdk-***-linux-***.bin
drwxrwsr-x   5 root    staff     4096 Mar 22  2006 lib/
lrwxrwxrwx   1 root    staff        9 Nov 18  2005 man -› share/man/
drwxrwsr-x   2 root    staff     4096 Nov 18  2005 sbin/
drwxrwsr-x   6 root    staff     4096 Mar 22  2006 share/
drwxrwsr-x   2 root    staff     4096 Nov 18  2005 src/
```

(Instead of a specific version number above we're showing asterisks -- when you get around to doing these steps you're likely to wind up with a more modern instance than what we just downloaded, so we're trying to keep this very generic.) And there, we "run" it (after all it's just a shell script):

```
# sh jdk*bin
```

It'll ask for you to agree to the licencing scheme again, and once you do it'll decompress everything in no time.

Now you'll have a new directory:

```
drwxr-xr-x   9 root    staff     4096 Oct 12 15:01 jdk1.*/
```

This is your java home! (Again, we're generifying the version numbers since they're ephemeral.) For convenience, we make a generic "java" symbolic link to refer to our new java home:

```
# ln -s jdk1.* java
# ls -ldF j*
lrwxrwxrwx  1 root staff       11 Oct 25 16:41 java -› jdk1.*/
-rw-r--r--  1 root staff 49518537 Oct 25 16:21 jdk-***-linux-***.bin
```

```
drwxr-xr-x  9 root staff     4096 Oct 12 15:01 jdk1.*/
```

So now when we refer to **/usr/local/java** we're really referring to the instance we unfolded into the jdk1.*/ directory. And if we ever need to install an update, we do so and then just point the "java" symlink to the new directory. Neat!

Maven:
Here, the specific VERSION is actually important -- Maven 1.0.2 is what Sakai expects, until further notice. If you look at the Apache website you'll see that Maven is already past version 2.0... but the Sakai architects haven't given the "green light" to that one yet [NOTE: Maven 2.0 **is required** for 2.5.x and beyond, but this install-howto is for version 2.3.x] so stick with 1.0.2 for now. So here we go:

- Browse to http://maven.apache.org/maven-1.x/ making sure you're NOT looking at version 2.x!
- Click "Download"

- Choose the **Maven 1.0.2 version** appropriate for your platform:

  Only version 1.0.2, remember! For Debian/GNU Linux, we use ".tar.gz".
- Clicking the link there will take you to a list-of-mirrors, and THAT is where you download the actual file. (If you download the above link and try to unzip it, you won't get very far since it's really just an HTML page.)

Once we have it downloaded we move it to /usr/local:

```
# cd /usr/local
# ls -lF maven*
-rw-r--r--  1 root staff 20092 Oct 25 20:46 maven-1.0.2.tar.gzAnd now we unzip it:# tar zxf maven-1.0.2.tar.gz
# ls -ldF mav*
drwxr-xr-x  5 root root     4096 Dec  7  2004 maven-1.0.2/
-rw-r--r--  1 root staff 6060686 Dec  7  2004 maven-1.0.2.tar.gz
```

Now we do our "symlink" trick again:

```
# ln -s maven-1.0.2 maven
# ls -ldF mav*
lrwxrwxrwx  1 root staff      11 Oct 27 11:11 maven -› maven-1.0.2/
drwxr-xr-x  5 root root     4096 Dec  7  2004 maven-1.0.2/
-rw-r--r--  1 root staff 6060686 Dec  7  2004 maven-1.0.2.tar.gz
```

Now for our web server--
Tomcat:
There are TWO downloads to make here: one is Tomcat itself, the other is the JDK compatibility package.

- Visit http://tomcat.apache.org/
- Click "Tomcat 5.x" under "Download":

- Download the appropriate "Core" instance for your platform -- for Debian/GNU Linux we use ".tar.gz":

- ALSO download the JDK compatibility package as well:

First, move the Tomcat archive to /usr/local and unfold it there -- and make a symlink to it:

```
# tar xfz apache-tomcat*
# ln -s apache-tomcat-*** tomcat
# ls -ldF *tomcat*
drwxr-sr-x  11 root staff    4096 Oct 27 15:33 apache-tomcat-*/
-rw-r--r--   1 root staff 5949295 Sep 28 09:01 apache-tomcat-*.gz
lrwxrwxrwx   1 root staff      20 Oct 27 15:33 tomcat -› apache-tomcat-*/
```

(Note we're leaving out the version-number specifics again -- just make sure you're using the latest *version 5* tomcat.)

Now move the JDK-compat package to /usr/local and unfold it right on top of your new Tomcat setup (it adds bin/jmx.jar, common/endorsed/xercesImpl.jar and common/endorsed/xml-apis.jar):

```
# tar xfz apache-tomcat-***-compat.tar.gz
# ls -ldF *tomcat*
drwxr-sr-x  11 root staff    4096 Oct 27 15:50 apache-tomcat-*/
-rw-r--r--   1 root staff 1624224 Sep 28 09:01 apache-tomcat-*-compat.tar.gz
-rw-r--r--   1 root staff 5949295 Sep 28 09:01 apache-tomcat-*.tar.gz
lrwxrwxrwx   1 root staff      20 Oct 27 15:53 tomcat -› apache-tomcat-*/
```

So far so good!
Now for a little configuration--
Edit **tomcat/conf/server.xml** and look for port="8080" to find this snippet:

```
    ‹Connector port="8080" maxHttpHeaderSize="8192"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" redirectPort="8443" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true" /›
```

We need to add a URIencoding attribute there:

```
    ‹Connector port="8080" maxHttpHeaderSize="8192"
               URIEncoding="UTF-8"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" redirectPort="8443" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true" /›
```

Nothing to it! Now, we'll make life a little easier for casual browsers by giving them a default redirect: they'll be able to browse to http://your.server:8080 and this will redirect them to your OSP portal [which is being deprecated in version 2.5.x and beyond] instead of showing a default Tomcat page:

```
# cat › tomcat/webapps/ROOT/index.html
‹html›
    ‹head›
        ‹meta http-equiv="refresh" content="0;url=/osp-portal"›
        ‹title›Redirecting to /osp-portal‹/title›
```

```
    ‹/head›
    ‹body›serving up ‹a href="/osp-portal"›/osp-portal‹/a› instead...‹/body›
‹/html›
^D
```
End with control-D to close and finish the file. Tomcat should be ready to rumble!

Subversion:
You may have noticed that we use Debian/GNU Linux -- and here's why:

```
# apt-get update
# apt-get install subversion
```

And to keep everything up-to-date, we just do:

```
# apt-get upgrade
```

Big chore, eh? Next!

Unfortunately, Debian's Tomcat is a version behind, and Maven isn't even available as a Debian package. Zut alors!

Mysql:
Again, using Debian, we can get MySql running in a jiffy:

```
# apt-get update
# apt-get install mysql-client-4.1 mysql-server-4.1
```

Nothing to it!

Another nice aspect of dealing with Debian is that "unstable" to the debian community matches what many other folks would consider "rock solid". There are mission-critical servers running Debian "unstable" for years on end without a blip!

Note that we still don't have an actual Sakai database yet -- that will come soon enough...

We also need a java library to hook up your Java to your MySql (strangely enough, it's called the JDBC MySql connector):

- Browse to http://dev.mysql.com/downloads/connector/j/3.1.html
- Download the latest version-3 connector (version 5 isn't fully endorsed)
- Unfold it so you can get to the *.jar file:

```
$ tar xfz mysql*tar.gz
$ ls -ld mysql*
drwxr-xr-x  5 sakai sakai     4096 Oct 18 16:35 mysql-connector-java-3.1.*/
-rw-r--r--  1 sakai sakai 28936239 Oct 31 14:21 mysql-connector-java-3.1.*.tar.gz
$ ls -l mysql*/*jar
-rw-r--r--  1 sakai sakai 459094 Oct 18 16:35
mysql-connector-java-3.1.14/mysql-connector-java-3.1.*-bin.jar
```

- Plop the *.jar file into common/lib:

```
$ cp mysql*/*.jar /usr/local/tomcat/common/lib/
```

Without that, Tomcat would NOT be able to communicate with MySql.

On with the configuring!

Variables, Preparations and other Settings:
Okay. Before we do anything else, we'll create a "sakai" shell user and make sure permissions on /usr/local/tomcat are properly set:

```
# useradd sakai
# chown -R sakai:sakai /usr/local/tomcat/*
# su - sakai
```

The home directory of user "sakai" is where the source code and settings will be for building Sakai/OSP. And when user "sakai" builds and deploys the sakai system, it'll wind up in /usr/local/tomcat.

You should set some environmental variables: do these "live" on your command line so you can test them, but also put them in a shell script that you can source at any time (including from ~sakai/.bashrc) so they'll be "permanent".

First, SAKAI_HOME is ~sakai:

```
$ export SAKAI_HOME=/home/sakai
```

Next, where's java? It's in $JAVA_HOME:

```
$ export JAVA_HOME=/usr/local/java
$ ls -l $JAVA_HOME
lrwxrwxrwx  1 root staff 11 Oct 25 16:41 /usr/local/java -› jdk1.5***
```

So now $JAVA_HOME refers to our /usr/local/java symlink. And Java itself can be found in $JAVA_HOME/bin/java. Easy!

Now for $MAVEN_HOME:

```
$ export MAVEN_HOME=/usr/local/maven
```

Now add the appropriate BIN directories to your $PATH (in your .bashrc so it's permanent) and then test to make sure everything is all set up:

```
$ export PATH="$PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin"
$ java -version
java version "1.5***"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5***)
Java HotSpot(TM) Client VM (build 1.5***, mixed mode, sharing)
$ maven -v
 __  __
|  /  |__ _Apache__ ___
| |/| / _`  V / -_) '   ~ intelligent projects ~
|_|  |___,_|_/___|_||_|  v. 1.0.2
```

Piece of cake!

For Tomcat, the environmental variable is, for historical purposes, $CATALINA_HOME -- we set the variable (in .bashrc, of course) and then test it:

```
$ export CATALINA_HOME=/usr/local/tomcat
$ sh $CATALIA_HOME/bin/version.sh
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:       /usr/local/java
Server version: Apache Tomcat/***
Server built:   ***
Server number:  ***
OS Name:        Linux
OS Version:     ***
Architecture:   ***
JVM Version:    ***
JVM Vendor:     Sun Microsystems Inc.
```

To be specific, we create a script called "~sakai/variables.sh" and it contains:

```
#!/bin/sh
export SAKAI_HOME=/home/sakai
export JAVA_HOME=/usr/local/java
export MAVEN_HOME=/usr/local/maven
export CATALINA_HOME=/usr/local/tomcat
export PATH="$PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin"
export JAVA_OPTS="
```

Your ~/.bashrc should 'source' that file, and so could any other script -- which is useful, as we'll soon see.

Now let's get MAVEN ready: create a file "build.properties" to point to your tomcat directory and indicate the remote sakai repository:

```
$ cat › build.properties
maven.repo.remote=http://source.sakaiproject.org/maven/
maven.tomcat.home=/usr/local/tomcat/
^D
```

End "cat" with control-D (and note that /usr/local/tomcat is the tomcat-specific symlink we established a while back). Now, initialize your MAVEN repository:

```
$ $MAVEN_HOME/bin/install_repo.sh ~/.maven
$ ls -lF ~/.maven
drwxr-xr-x   3 sakai sakai 4096 Oct 27 17:57 ant/
drwxr-xr-x   3 sakai sakai 4096 Oct 27 17:57 commons-beanutils/
drwxr-xr-x   3 sakai sakai 4096 Oct 27 17:57 commons-betwixt/
drwxr-xr-x   3 sakai sakai 4096 Oct 27 17:57 commons-cli/
[yada yada, snip snip]
```

A walk in the park! Now for the important stuff--

Download, Build, Deploy:
Now you're ready to get started with the meat of the project -- and this is the part you can iterate over as you tweak your sakai instance (adding skins, for example):

```
$ svn export https://source.sakaiproject.org/svn/sakai/branches/sakai_2-3-x/
```

This will take quite a while -- lots of stuff to download! -- so go catch up on email if you need to. The 2-3-x instance gets all the latest patches and fixes, so we use that instead of 2-3-0 or 2-3-9.

Subversion ("svn" for short) creates the "sakai_2-3-x" directory and populates it with all kinds of source stuff. Eventually. Now, another symlink:

```
$ ln -s sakai_2-3-x sakai
$ ls -ldF sakai*
lrwxrwxrwx   1 sakai sakai   11 Oct 27 18:42 sakai -› sakai_2-3-x/
drwxr-xr-x  57 sakai sakai 4096 Oct 27 18:41 sakai_2-3-x/
```

Now our sakai source files will be found at **~sakai/sakai/** because of our old friend, the symlink trick. Nice!

Here we go! [Maven 2, which is used for 2.5.x and beyond, has a different set of commands to use here, of course: mvn clean install sakai:build] If you build it, they will come!

```
$ maven sakai | tee maven-sakai-log
Starting the reactor...[snip]
BUILD SUCCESSFUL
Total time: 10 minutes 43 seconds
Finished at: Fri Oct 27 19:23:04 CDT 2006
```

Using "maven sakai" is a nice shortcut for "maven cln bld dpl" which means "clean, build, deploy". It zaps old stuff (for the first instance here, there isn't any to zap, of course) and then rebuilds it, and deploys it into the tomcat directory (according to your build.properties file).

This command will take a LOOOONG time -- ten minutes, maybe more -- and generate copious output. (Thus it's good to use "tee" to keep a copy in case you want to look at it later.)

What BUILD does, is takes all the .src/* stuff, and compiles it to their respective .target/* destinations in preparation for deployment. DEPLOY moves it all to the live Tomcat directory. (And CLEAN zaps all the .target/ directories.)

So -- now that Tomcat has everything it needs to serve up Sakai, we need to configure Sakai to our liking, and we also need a database for Sakai to work with...

Final Configuration:
Two things remain: Sakai needs a database, and some configuration.
Create the Sakai database
As root, connect to MySql and:

```
mysql› create database sakai_2_3_x default character set utf8;
mysql› grant all on sakai_2_3_x.* to sakaiuser@'localhost' identified by 'snarkySekrit';
mysql› grant all on sakai_2_3_x.* to sakaiuser@'127.0.0.1' identified by 'snarkySekrit';
```

Piece of cake! Just make sure you use your own settings for the bold stuff -- particularly a reasonable password, as opposed to cut-and-pasting our example here...

Now we need to make sure sakai.properties is all set up:
Configuring sakai.properties
If we're going to tell Sakai (tomcat) that SAKAI_HOME is ~sakai, that's where it expects to find **sakai.properties**. You'll notice that there isn't any such animal there, yet.

Fortunately there are several available in the source code you just downloaded via svn: the one we want is specifically OSP-savvy:

```
$ mkdir $CATALINA_HOME/sakai
```

```
$ cp
sakai/osp/overlay/component/component-api/component/src/config/org/sakaiproject/config/s
akai.properties $CATALINA_HOME/sakai
$ ln -s $CATALINA_HOME/sakai/sakai.properties ~/properties
$ ln -s $CATALINA_HOME/logs/catalina.out ~/
```

Now we have ~sakai/properties as a symlink to $CATALINA_HOME/sakai/sakai.properties and a shortcut to the catalina.out logfile as well.

The OSP instance is the one with /osp-portal instead of just /portal. (This is assuming your sakai symlink -- the one that points to the sakai-* source code you just downloaded via svn -- is in your current directory, of course.)

Now, edit your "~sakai/sakai.properties" file to ensure:

```
# set the hibernate dialect (for shared datasource), HSQLDB by default, mySql and Oracle
examples
#hibernate.dialect=org.hibernate.dialect.HSQLDialect
hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

That is, disable (comment-out) HSQL, and enable MySQL instead. Also add:

```
vendor@org.sakaiproject.db.api.SqlService=mysql
driverClassName@javax.sql.BaseDataSource=com.mysql.jdbc.Driver
url@javax.sql.BaseDataSource=jdbc:mysql://localhost:3306/sakai_2_2_x
?useUnicode=true&characterEncoding=UTF-8
username@javax.sql.BaseDataSource=sakaiuser
password@javax.sql.BaseDataSource=snarkySekrit
validationQuery@javax.sql.BaseDataSource=select 1 from DUAL
defaultTransactionIsolationString@javax.sql.BaseDataSource=TRANSACTION_READ_COMMITTED
```

Those are the parameters needed to tell Tomcat how to connect to the Sakai database; the bold stuff needs to be replaced by the strings you've used in your setup. (During the first connection, it'll create all the tables as needed. Nice!)

Also tweak the following, accordingly:

```
serverId=serious.name.net
serverUrl=http://www.serious.name.net:8080
serverName=www.serious.name.net
termyear.*
termlistabbr.*
termstarttime.*
termendtime.*
```

Now we can create up.sh/down.sh scripts to launch and terminate Sakai:

```
#!/bin/sh
. ~sakai/variables.sh
$CATALINA_HOME/bin/startup.sh && tail -f $CATALINA_HOME/logs/catalina.out
```

That's "up.sh" and here's "down.sh":

```
#!/bin/sh
. ~sakai/variables.sh
$CATALINA_HOME/bin/shutdown.sh && tail -f $CATALINA_HOME/logs/catalina.out
```

Note that these scripts finish with "tail -f" which will show the log file contuously, as it grows... to get your command-line back, you can press ^C (control-C) any time.

```
$ ~/up.sh
Using CATALINA_BASE:   /usr/local/tomcat
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:       /usr/local/java
[snip]
```

The first four lines should reflect some of your environment variable settings. And we're off to the races!

This will continue for QUITE A WHILE -- ten minutes or more, on the first run. Remember, it has to initialize the database as well!

If all goes well, you'll see LOTS of "INFO:" items fly by (and hopefully no ERROR or SEVERE). If you get to the point where the log shows:

```
INFO: Server startup in 258168 ms
```

Then you're up! To shut it down:

```
$ ~/down.sh
[snip]
INFO: Stopping Coyote HTTP/1.1 on http-8080[snip]When you see "Stopping Coyote" you're all finished!
```

Versions (as of January 2007):Here are the specific versions we were working with above -- by the time you get to using these instructions, many will have been upgraded and the version numbers will have evolved accordingly...

- Java: *jdk-1.5.0_10-linux-i586-rpm.bin*
- Maven: *maven-1.0.2*
- Tomcat: *apache-tomcat-5.5.20*
- Subversion: *1.1.4-2 (Debian Sarge)*
- MySql: *4.1.12 (Ubuntu universe)*
- Sakai/OSP: 2-3-x

Unique solution ID: #1025
Author: will trillich
Last update: 2010-11-16 07:50