SSS Build Howto

Introduction

The examples below step you through the process of producing version 2.4.0 of parminator; text in

```
this font
```

that starts with a

```
%
```

are command prompts. The output has been trimmed for brevity. See SSS Build Policy for definitions of MAJOR, MINOR and MAINTENANCE.

Subversion Repository Structure

```
parminator
|-- branches
|   |-- 2.2
|   |-- 2.3
|   `-- 2.4
|-- tags
|   |-- 2.2.0
|   |-- 2.2.1
|   |-- 2.2.2
|   |-- 2.3.0
|   |-- 2.3.1
|   `-- 2.4.0
`-- trunk
```

To the left is the directory structure of a sample subversion repository for **parminator**. The three elements under the directory tree root are trunk, branches and tags.

- branches: this holds MAJOR.MINOR branches for parminator, and is used for creating MAINTENANCE releases of parminator.
- tags: holds MAJOR.MINOR.MAINTENANCE releases of parminator: tags are immutable, this tree is for historical reasons, such as recompiling a release or comparing source of different releases.
- trunk: the development source tree for parminator.

How to Make a New MAJOR.MINOR Version

Follow this procedure to create a new MAJOR.MINOR release of **parminator**, when new features have been introduced in the trunk.

1. Start with a clean copy of the trunk.

```
% svn checkout https://svn.aoc.nrao.edu/repos/EVLA/parminator/trunk parminator
```

2. Change into the directory you created.

```
% cd parminator
```

3. Copy the trunk into branches/2.4; this is one line.

```
% svn copy https://svn.aoc.nrao.edu/repos/EVLA/parminator/trunk    https://svn.aoc.nrao.edu/repos/EVLA/parminator/branches/2.4    -m "Init
```

4. Change trunk/pom.xml, setting version to 2.5-SNAPSHOT, or 3.0-SNAPSHOT, if that seems to be the more likely next version.

```
% emacs pom.xml
```

5. Commit your changes

```
% svn commit -m "Starting 2.5-SNAPSHOT"
```

6. Continue on to the next section to build MAINTENANCE Release zero for parminator 2.4 (aka parminator 2.4.0).

How to Make a New MAINTENANCE Release

Follow this procedure to create a new MAINTENANCE release of **parminator**, either a dot zero release (e.g. 2.4.0) or a subsequent bugfix release of an established MAJOR.MINOR series (2.4.1 and so on).

1. Assume we're in a directory that has a sandbox of branches/2.4. If not, check out or switch to the branches/2.4 tree.

```
% svn switch https://svn.aoc.nrao.edu/repos/EVLA/parminator/branches/2.4
```

2. Edit branches/2.4/pom.xml, set version to 2.4.0, check to make sure any NRAO dependencies are real releases and not SNAPSHOTs.

```
% emacs pom.xml
```

3. Build branches/2.4, test it to make sure you didn't typo or make it depend on a missing dependency. If you are porting a bugfix over from trunk, do that now.

```
% mvn package clean
```

4. Commit your change to branches/2.4/pom.xml

```
% svn commit -m "Preparing for rollout of 2.4.0"
```

5. Copy branches/2.4 to tags/2.4.0

```
% svn copy https://svn.aoc.nrao.edu/repos/EVLA/parminator/branches/2.4    https://svn.aoc.nrao.edu/repos/EVLA/parminator/tags/2.4.0 -m
```

6. Compile, install and clean the sandbox.

```
% mvn install clean
```

Branching and Tagging Scripted

The above process to do a simple branch and tag of a release has been scripted. branchNTag.sh is a bash shell script that will branch and tag all of the SSS software at once. It will branch, tag, and deploy: `sss-xsd`, `util`, `data-models`, `data-access-layers`, `web-util`, `rct`, `sct`, `opt`, and `model2script`. The script assumes that you have all the mentioned projects checked out into directories at the same directory level as the script and that the directories have the same names as their svn project name. For example:

```
- branchNTag.sh
- sss-xsd/
- util/
- data-models/
...
etc.
```

`branchNTag.sh` takes 2 command line arguments: the MAJOR.MINOR version that you are tagging (2.4) and the MAJOR.MINOR version of the next snapshot (2.5). The script will follow the procedure on this page. It will change directories into a project and ask you if you want to continue with branching and tagging that project. You can hit `CTRL-C` to cancel, or Enter to continue. If you continue, it will create a branch, and then open `pom.xml` in your default text editor in a new terminal for you to edit it and update it for the next snapshot version. The editor used is dependent on your `EDITOR` environment variable. If it is not defined, `vim` is used.

After you save and exit, it will commit those changes, `svn switch` to the newly created branch, and open `pom.xml` for you again so you can update the version and dependencies for the release you will be making. When you save and exit, it will do a `mvn clean deploy -Dmaven.test.skip=true` and then give you a prompt that says "Press Enter to continue with tagging". You can hit `CTRL-C` at this point to stop the script. If you hit enter, it will commit the changes to `pom.xml` and then do an `svn copy` to create the tag. Finally it will `svn switch` back to the trunk.

Whenever the script opens a new terminal for you to edit in, it will display a message in the original terminal that should be a helpful reminder as to what you are expected to do in that step. In the event that you have canceled the script, the next time you run the script, it should pick up with the project that you canceled during. It does this by adding empty files to each directory as it processes them. At the end of the script, all such files are removed.

maintTag.sh is a very similar script created to do maintenance releases of the SSS software. Use this script if you have already branched and tagged this MAJOR.MINOR release sometime before and are now just creating a new MAINTENANCE release. This script has the same assumptions as the previous script. It takes a single command line argument: the new version being tagged (i.e. 2.4.1). It starts by changing directories to the project directory and doing an `svn switch` to the appropriate branch. Then it opens an `xterm` so that you can merge changes in and update `pom.xml` as necessary. As soon as you close that terminal, the script will continue and deploy the project, commit the changes and create a tag just as before.

If you have to merge in changes, you should commit those changes BEFORE modifying `pom.xml` and before closing the `xterm`! The commit message should include the range of revision numbers that you have merged. You do NOT have to commit changes to `pom.xml` after that as the script does that for you.

Note that if we follow a development pattern where development for maintenance releases takes place in the branch instead of the trunk (which I think is a good idea) then you will have no merges to do during this step. You can merge changes from the branch into the trunk at some other time (either before tagging or after, it doesn't matter). All you'll need to do is update `pom.xml`.

Both of these scripts are under version control in subversion and can be found at https://svn.aoc.nrao.edu/repos/SSS/projectmanagement/

These scripts are not a replacement for knowing what you are doing!

This topic: EVLA > SSS > SSSBuildHowto
Topic revision: r9 - 17 Apr 2009 - 14:36:41 - BrianTruitt