# Branching

Trunk is always contain live like code and should be only used for Prod bug fixing and considers as Maintenance branch.

Release branch owner (Developer) should merge the code from Trunk frequently to their release branch to incorporate changes went as bug fixing. In case of any merge conflict, developers needs to talk to each other and fix it.

Post Release merges from release branch to Trunk should be done by Developer.

Some Guideline

Release branch should always cut from Trunk

Dev Branch can be cut from anywhere but preferable Trunk and frequent merge should happen from Trunk.

# Main/Prod Branching

This branch is strictly for the use of Production release. This branch should not contain any code but have the correct directory structure of all projects/repositories with their pom file.

Developer should update the artifact version in pom.xml for deployment.

Deployment to QA and Prod would happen from this branch.

# Build Script

Each Project should have top level build script which should call the subsequent script if necessary.

Project directory structure should look as per maven standard. (refer http://maven.apache.org/guides/introduction/introduction-to-the-standard-directorylayout.html for more information)

# Build

For creating new job in Hudson, Developer needs to open ticket to SCM with detail instruction including repo path, maven command, mailing list etc…

Developer needs to keep eye on their build and proactively work on any failed build.

Needs to upload maven artifact to Maven repository.

Mavenization

**Steps to create a new project:**

1. Create a parent directory. This directory should contain just a reactor pom.xml

(See Reactor plugin section).

2. Run ***mvn clean install*** to check whether this is installing successfully to local

repository.

3. If successful you should submit this as the first version of the trunk.

4. Once the trunk has been created, create a project branch from trunk.

For the component artifacts you can start migrating files from your old working

location to your **new** location. This will include all source files, xml files,

test properties and anything else that is required on the classpath.(See

http://maven.apache.org/guides/introduction/introduction-to-the-standard-directorylayout.html for the directory structure standards)

The preferred approach for migration is to integrate file directories across from ant

workspace to maven. This will ensure that we maintain file history in SVN.

You will be required to create a module for each archive type. This means you should

have separate components for each jar, war, ear and sar. You can further split the jars

into logical modules such as having a core component that’s shared across other

components. The application teams should make this decision.

Each artifact will be required to include all the dependencies required for that artifact.

All configurations should be externalised from the binary artifacts. A configuration

bundle should be available for each environment supported e.g. dev, qa1, qa2, qa3,

qa4, qa5, stg, prod3 and prod9 plus qa and prod

Each environment configuration bundle should be created using the maven-assemblyplugin.

This plugin will take the input directories and files to create the specified

Output.

<http://maven.apache.org/plugins/maven-assembly-plugin/>

Here you can see an example implementation of the maven-lmn-config-processorplugin.

You can view the project at **http://code.google.com/p/maven-configprocessor-**

**plugin**/ and the LMN extension is //depot/services/tools/plugins

This plugin declaration will use the *src/assemble/config/env/dev.properties* properties

file to filter the values from *src/assemble/config/project-config.xml* to create a dev

version in *target/assemble/dev/server/default/deploy/project-config.xml*. The

processing follows the rules specified in *src/assemble/config/config-processing.xml*.

You can see more on the processing rules at

**http://code.google.com/p/maven-config-processorplugin/**

**wiki/TransformationConfiguration**

Once the environment specific files have been created you can use the mavenassembly-

plugin to included these files to form the environment config tar



This plugin declaration will execute the specified assembly files in the specified

execution phase.



This assembly execution will take the files generated by the maven-configprocessing-

plugin to construct the environment config jar. You will notice that the

directory structure is maintained in the tar so to support a simple copy when

deploying to a JBOSS container