Committers' Guide

Table of Contents

Chapter 1. Committers' Guide

This committers' guide is for committers of Apache Isis itself who want guidance on release process, publishing documents and other related procedures.

1.1. Other Guides

Apache Isis documentation is broken out into a number of user, reference and "supporting procedures" guides.

The user guides available are:

- Fundamentals
- Wicket viewer
- Restful Objects viewer
- DataNucleus object store
- Security
- Testing
- Beyond the Basics

The reference guides are:

- Annotations
- Domain Services
- Configuration Properties
- Classes, Methods and Schema
- Apache Isis Maven plugin
- Framework Internal Services

The remaining guides are:

- Developers' Guide (how to set up a development environment for Apache Isis and contribute back to the project)
- Committers' Guide (this guide)

This guide provides guidance for Apache Isis' own committers.

Chapter 2. Applying Patches

If a patch is received on a JIRA ticket, then it should be reviewed and applied. The commands are slightly different for diff files vs patch files.

2.1. Diff files

If a diff file has been provided, then it can easily be applied using eGit's wizards (Eclipse's Git integration)...

FIXME - this stuff needs fleshing out ...

2.2. Patch files

If a patch file has been provided, then it can be applied using command line tools.

2.2.1. Inspect the patch

First, take a look at what changes are in the patch. You can do this easily with `git apply`

```
git apply --stat ISIS-xxx.patch
```

Note that this command does not apply the patch, but only shows you the stats about what itâ€™ll do. After peeking into the patch file with your favorite editor, you can see what the actual changes are.

Next, you're interested in how troublesome the patch is going to be. Git allows you to test the patch before you actually apply it.

```
git apply --check ISIS-xxx.patch
```

If you don't get any errors, the patch has no conflicts. Otherwise you may see what trouble youâ€™ll run into.

2.2.2. Apply a (clean) patch

To apply a clean patch (adding the items and commit/signoff in a single step), use `git am`:

```
git am --signoff < ISIS-xxx.patch
```

This preserves the original author's commit message.

However, this can fail if the patch file does not contain the original committers email address. In this case you will need to abort the `am` session:

```
git am abort
```

and continue on by applying a non-clean patch, as described next.

2.2.3. Apply a (non-clean) patch

If the patch does not apply cleanly, then the original authors commit message cannot be preserved. This sequence in this case is:

```
git apply ISIS-xxx.patch
```

Fix up any issues. The add and commit as usual

```
git add .
git commit -am "<original authors' commit message>" --signoff
```

The `--signoff` simply adds a line to the commit message indicating you have signed off the commit.

Information adapted from this blog post and this wiki page.

Chapter 3. Merging a Pull Request

The process for merging in github pull requests (so that they can be tested locally before committing) has been scripted in the `github-pr.sh` script.

The script will merge the fork into a temporary branch, and then run a build. Once you are happy, you can commit.

3.1. Process and Usage

The overall process is as follows:

- locate/raise corresponding JIRA ticket, eg ISIS-1162
- checkout branch from which PR was forked (usually just 'master')
- merge PR into temporary branch using the `github-pr.sh` script
- test the change locally (run the app, rebuild, manual regression tests etc)
- if required, tidy up/refactor code as required
- merge temporary branch into mainline, and commit

This screencast also shows the process.

3.2. Prerequisites

The script uses 'jq' to parse JSON. To install:

- on Linux:

```
aptitude install jq
```

- on MacOS:

```
brew install jq
```

- on Windows:

  Download exe from website

3.3. Syntax

The syntax is:

```
github-pr.sh -j 1162 -g 31 [-s] [-p ISIS]
```

where:

- `-j 1162`

  is the JIRA ticket number

- `-g 31`

  is the github PR issue number

- `-s`

  will optionally skip the build and auto-merge

- `-p ISIS`

  optionally overrids the JIRA project (defaults to 'ISIS')

3.4. Example transcript

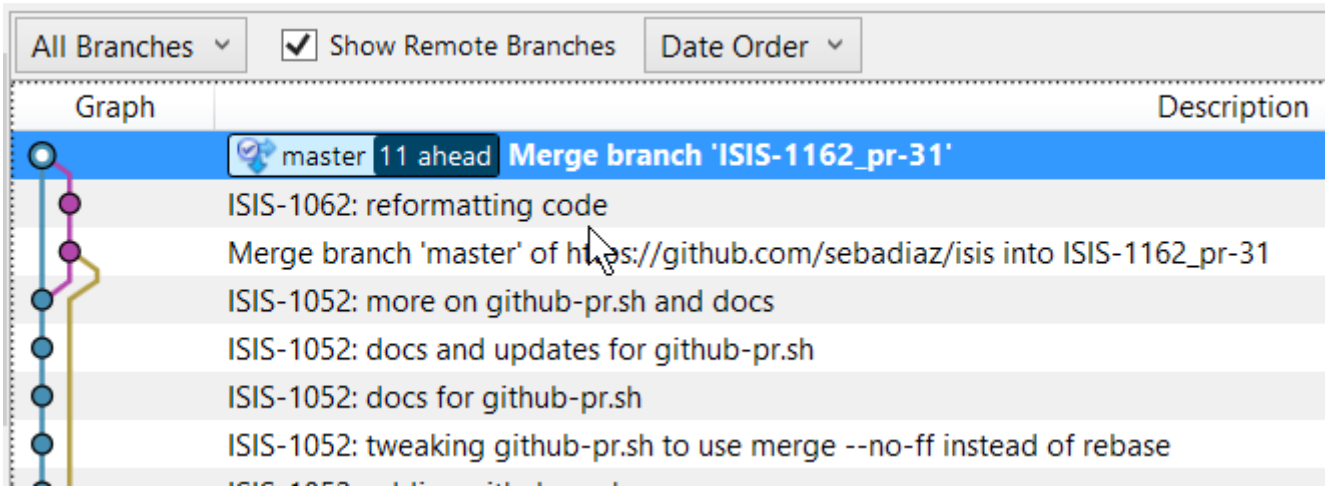
The listing below shows the steps taken by the script:

```
$ sh github-pr.sh isis 1162 31

Found JIRA ticket
Found github PR
branch_name_local: master
username         : sebadiaz
repo_full_name   : sebadiaz/isis
repo_clone_url   : https://github.com/sebadiaz/isis.git
branch_name_fork : master

merging into: ISIS-1162_pr-31

Deleting branch 'ISIS-1162_pr-31'
Deleted branch ISIS-1162_pr-31 (was bd2e3c2).
Creating the branch ISIS-1162_pr-31
Switched to a new branch 'ISIS-1162_pr-31'
Pulling the changes from https://github.com/sebadiaz/isis.git master
From https://github.com/sebadiaz/isis
 * branch            master     -> FETCH_HEAD
Auto-merging core/pom.xml
Merge made by the 'recursive' strategy.
 core/pom.xml                                       |   3 +-
 .../apache/isis/security/shiro/IsisLdapRealm.java  | 198 ++++++++++++++++++--
 2 files changed, 186 insertions(+), 15 deletions(-)

Merged the PR; hit enter to build
```

The build now commences. Once done, the script continues:

```
If build successful and happy to merge, execute:

git checkout master && git merge --no-ff ISIS-1162_pr-31 && git branch -d ISIS-
1162_pr-31
```

The screenshot belows shows the history we end up with:



This shows the fork being merged into the temporary branch ("ISIS-1162_pr-31"), then some further tidy-up, and finally the merging of the temporary branch into mainline.

Note that there is no rebasing in this model. This is intentional: when the merged branch is pushed, github will automatically close the original pull request.

Chapter 4. Cutting a Release

The release process consists of:

- the release manager cutting the release (documented below)
- Members of the Apache Isis PMC verifying and voting on the release
- the release manager performing post-release tasks, for either a successful or an unsuccessful vote.

Apache Isis itself consists of three separately releasable modules; relative to the source code root there are:

- `core`
- `component/example/archetypes/helloworld`
- `component/example/archetypes/simpleapp`

This section details the process for formally releasing Isis modules. It describes the process for both `core` and then the archetypes. The subsequent sections describe how other committers can verify a release and how the release manager can then perform post-release activities and set up for the next development iteration.

If you've not performed a release before, then note that there are some configuration prerequisites that must be configured first. In particular, you'll need signed public/private keys, and the ASF Nexus staging repo inlocal `~/.m2/settings.xml` file.

These release notes using bash command line tools. They should work on Linux and MacOS; for Windows, use mSysGit.

4.1. Obtain Consensus

Before releasing `core`, ensure there is consensus on the dev mailing list that this is the right time for a release. The discussion should include confirming the version number to be used, and to confirm content.

These discussions should also confirm the version number of the module being released. This should be in line with our semantic versioning policy.

Make sure you have a JIRA ticket open against which to perform all commits. In most cases a JIRA ticket will have been created at the beginning of the previous release cycle.

4.2. Set environment variables

We use environment variables to parameterize as many of the steps as possible. For example:

```
cd core
export ISISTMP=/c/tmp                                   ①
export ISISDEV=1.16.0-SNAPSHOT
export ISISREL=1.15.1
export ISISRC=RC1
export ISISBRANCH=release-$ISISREL-$ISISRC
export ISISJIRA=ISIS-9999                               ②
export CATALINA_HOME=/c/java/apache-tomcat-8.0.30       ③

env | grep ISIS | sort
```

① adjust by platform

② set to an "umbrella" ticket for all release activities. (One should exist already, created at the beginning of the development cycle now completing).

③ adjust as required (Tomcat is used to smoke test the simpleapp archetype)

Obviously, alter `$ISISDEV` and `$ISISREL` as required, and bump `$ISISRC` for re-releasing following an unsuccessful releases.

Note that the branch name is **not** the same any of the eventual tag names (eg `isis-1.16.0` or `simpleapp-archetype-1.16.0`).

If they did have the same name, then what would happen is that the `maven-release-plugin` would checkout the (HEAD of the) branch and thus upload a SNAPSHOT to the snapshot repository. What it should of course do is checkout the tag and then upload that to the release staging repository.

4.3. Pull down code to release

Set the HEAD of your local git repo to the commit to be released. This will usually be the tip of the origin's `master` branch. Then, create a release branch for the version number being released; eg:

```
git checkout master
git pull --ff-only
git checkout -b $ISISBRANCH
```

All release preparation is done locally; if we are successful, this branch will be merged back into master.

Double check that the version number of the parent pom should reflect the branch name that you are now on (with a `-SNAPSHOT` suffix). his will normally have been done already during earlier development; but confirm that it has been updated. If it has not, make the change.

Double check that the version number of the core POM (`core/pom.xml`) should reflect the branch name that you are now on. For example, if releasing version `1.16.0`, the POM should read:

```
<groupId>org.apache.isis.core</groupId>
<artifactId>isis</artifactId>
<version>1.16.0-SNAPSHOT</version>
```

Also, check that there are no snapshot dependencies:

```
grep SNAPSHOT `/bin/find . -name pom.xml | grep -v target | grep -v mothball | sort`
```

The only mention of `SNAPSHOT` should be for the Isis modules about to be released.

Obviously, don't update Apache Isis' `SNAPSHOT` references; these get updated by the `mvn release:prepare` command we run later.

4.4. Releasing Core

First, we release `core`. Switch to the appropriate directory:

```
cd core
```

4.4.1. Set environment variables

Set additional environment variables for the core "artifact":

```
export ISISART=isis
export ISISCOR="Y"

env | grep ISIS | sort
```

4.4.2. License headers

The Apache Release Audit Tool `RAT` (from the Apache Creadur project) checks for missing license header files. The parent `pom.xml` of each releasable module specifies the RAT Maven plugin, with a number of custom exclusions.

To run the RAT tool, use:

```
mvn org.apache.rat:apache-rat-plugin:check -D rat.numUnapprovedLicenses=50 -o && \
for a in `/bin/find . -name rat.txt -print`; do grep '!???' $a; done || \
for a in `/bin/find . -name rat.txt -print`; do grep '!AL' $a; done
```

where `rat.numUnapprovedLicenses` property is set to a high figure, temporarily overriding the default value of 0. This will allow the command to run over all submodules, rather than failing after the first one. The command writes out a `target\rat.txt` for each submodule. missing license notes are

indicated using the key `!???`. The `for` command collates all the errors.

Investigate and fix any reported violations, typically by either:

- adding genuinely missing license headers from Java (or other) source files, or
- updating the `<excludes>` element for the `apache-rat-plugin` plugin to ignore test files, log files and any other non-source code files
- also look to remove any stale `<exclude>` entries

To add missing headers, use the groovy script `addmissinglicenses.groovy` (in the `scripts` directory) to automatically insert missing headers for certain file types. The actual files checked are those with extensions specified in the line `def fileEndings = [".java", ".htm"]`:

```
groovy ../scripts/addmissinglicenses.groovy -x
```

(If the `-x` is omitted then the script is run in "dry run" mode). Once you've fixed all issues, confirm once more that `apache-rat-plugin` no longer reports any license violations, this time leaving the `rat.numUnapprovedLicenses` property to its default, 0:

```
mvn org.apache.rat:apache-rat-plugin:check -D rat.numUnapprovedLicenses=0 -o && \
for a in `find . -name rat.txt -print`; do grep '!???' $a; done
```

4.4.3. Missing License Check

Although Apache Isis has no dependencies on artifacts with incompatible licenses, the POMs for some of these dependencies (in the Maven central repo) do not necessarily contain the required license information. Without appropriate additional configuration, this would result in the generated `DEPENDENCIES` file and generated Maven site indicating dependencies as having "unknown" licenses.

Fortunately, Maven allows the missing information to be provided by configuring the `maven-remote-resources-plugin`. This is stored in the `src/main/appended-resources/supplemental-models.xml` file, relative to the root of each releasable module.

To capture the missing license information, use:

```
mvn license:download-licenses && \
groovy ../scripts/checkmissinglicenses.groovy
```

The Maven plugin creates a `license.xml` file in the `target/generated-resources` directory of each module. The script then searches for these `licenses.xml` files, and compares them against the contents of the `supplemental-models.xml` file.

For example, the output could be something like:

```
licenses to add to supplemental-models.xml:

[org.slf4j, slf4j-api, 1.5.7]
[org.codehaus.groovy, groovy-all, 1.7.2]

licenses to remove from supplemental-models.xml (are spurious):

[org.slf4j, slf4j-api, 1.5.2]
```

If any missing entries are listed or are spurious, then update `supplemental-models.xml` and try again.

4.4.4. Commit changes

Commit any changes from the preceding steps:

```
git commit -am "$ISISJIRA: updates to pom.xml etc for release"
```

4.4.5. Sanity check

Perform one last sanity check on the codebase. Delete all Isis artifacts from your local Maven repo, then build using the `-o` offline flag:

```
rm -rf ~/.m2/repository/org/apache/isis
mvn clean install -o
```

4.4.6. Release prepare "dry run"

Most of the work is done using the `mvn release:prepare` goal. Since this makes a lot of changes, we run it first in "dry run" mode; only if that works do we run the goal for real.

Run the dry-run as follows:

```
mvn release:prepare -P apache-release -D dryRun=true \
    -DreleaseVersion=$ISISREL \
    -Dtag=$ISISART-$ISISREL \
    -DdevelopmentVersion=$ISISDEV
```

You may be prompted for the gpg passphrase.

Experiments in using `--batch-mode -Dgpg.passphrase="…"` to fully automate this didn't work; for more info, see here (maven release plugin docs) and here (maven gpg plugin docs).

4.4.7. Release prepare "proper"

Assuming this completes successfully, re-run the command, but without the `dryRun` flag and specifying `resume=false` (to ignore the generated `release.properties` file that gets generated as a side-effect of using `git`). You can also set the `skipTests` flag since they would have been run during the previous dry run:

```
mvn release:prepare -P apache-release -D resume=false -DskipTests=true \
        -DreleaseVersion=$ISISREL \
        -Dtag=$ISISART-$ISISREL \
        -DdevelopmentVersion=$ISISDEV
```

If there are any snags at this stage, then explicitly delete the generated `release.properties` file first before trying again.

4.4.8. Post-prepare sanity check

You should end up with artifacts in your local repo with the new version (eg `1.16.0`). This is a good time to do some quick sanity checks; nothing has yet been uploaded:

- unzip the source-release ZIP and check it builds.
- Inspect the `DEPENDENCIES` file, and check it looks correct.

These steps can be performed using the following script:

```
rm -rf $ISISTMP/$ISISART-$ISISREL
mkdir $ISISTMP/$ISISART-$ISISREL

if [ "$ISISCOR" == "Y" ]; then
    ZIPDIR="$M2_REPO/repository/org/apache/isis/core/$ISISART/$ISISREL"
else
    ZIPDIR="$M2_REPO/repository/org/apache/isis/$ISISCPT/$ISISART/$ISISREL"
fi
echo "cp \"$ZIPDIR/$ISISART-$ISISREL-source-release.zip\" $ISISTMP/$ISISART-
$ISISREL/."
cp "$ZIPDIR/$ISISART-$ISISREL-source-release.zip" $ISISTMP/$ISISART-$ISISREL/.

pushd $ISISTMP/$ISISART-$ISISREL
unzip $ISISART-$ISISREL-source-release.zip

cd $ISISART-$ISISREL
mvn clean install

cat DEPENDENCIES

popd
```

4.4.9. Release perform (Upload)

Once the release has been built locally, it should be uploaded for voting. This is done by deploying the Maven artifacts to a staging directory (this includes the source release ZIP file which will be voted upon).

The Apache staging repository runs on Nexus server, hosted at repository.apache.org. The process of uploading will create a staging repository that is associated with the host (IP address) performing the release. Once the repository is staged, the newly created staging repository is "closed" in order to make it available to others.

Use:

```
mvn release:perform -P apache-release \
    -DworkingDirectory=$ISISTMP/$ISISART-$ISISREL/checkout
```

The custom `workingDirectory` prevents file path issues if releasing on Windows. The command checks out the codebase from the tag, then builds the artifacts, then uploads them to the Apache staging repository:

```
...
[INFO] --- maven-release-plugin:2.3.2:perform (default-cli) @ isis ---
[INFO] Performing a LOCAL checkout from scm:git:file:///C:\APACHE\isis-git-rw\co
re
[INFO] Checking out the project to perform the release ...
[INFO] Executing: cmd.exe /X /C "git clone --branch release-1.16.0
file:///C:\APACHE\isis-git-rw\core C:\APACHE\isis-git-rw\core\target\checkout"
[INFO] Working directory: C:\APACHE\isis-git-rw\core\target
[INFO] Performing a LOCAL checkout from scm:git:file:///C:\APACHE\isis-git-rw
[INFO] Checking out the project to perform the release ...
[INFO] Executing: cmd.exe /X /C "git clone --branch release-1.16.0
file:///C:\APACHE\isis-git-rw C:\APACHE\isis-git-rw\core\target\checkout"
[INFO] Working directory: C:\APACHE\isis-git-rw\core\target
[INFO] Executing: cmd.exe /X /C "git ls-remote file:///C:\APACHE\isis-git-rw"
[INFO] Working directory: C:\Users\ADMINI~1\AppData\Local\Temp
[INFO] Executing: cmd.exe /X /C "git fetch file:///C:\APACHE\isis-git-rw"
[INFO] Working directory: C:\APACHE\isis-git-rw\core\target\checkout
[INFO] Executing: cmd.exe /X /C "git checkout release-1.16.0"
[INFO] Working directory: C:\APACHE\isis-git-rw\core\target\checkout
[INFO] Executing: cmd.exe /X /C "git ls-files"
[INFO] Working directory: C:\APACHE\isis-git-rw\core\target\checkout
[INFO] Invoking perform goals in directory C:\APACHE\isis-git-
rw\core\target\checkout\core
[INFO] Executing goals 'deploy'...
...
```

You may (again) be prompted for gpg passphrase. All being well this command will complete successfully. Given that it is uploading code artifacts, it could take a while to complete.

4.5. Releasing the Archetypes

Apache Isis archetypes are reverse engineered from example applications. Once reverse engineered, the source is checked into git (replacing any earlier version of the archetype) and released.

There are currently two archetypes, `simpleapp` and `helloworld`.

If releasing using Windows and Maven >= 3.3.3, then there is an issue that requires a small workaround.

In Maven 3.3.3 the `mvn.bat` file was removed, replaced instead with `mvn.cmd`. However, `maven-archetype-plugin:2.4` only uses `mvn.bat`; this causes the `archetype:create-from-project` goal to fail. The fix is to simple: just copy `mvn.cmd` to `mvn.bat`.

4.5.1. Releasing `simpleapp` archetype

Switch to the directory containing the `simpleapp` example:

```
cd ../example/application/simpleapp
```

Setup environment variables

Set additional environment variables for the `simpleapp-archetype` artifact:

```
export ISISART=simpleapp-archetype
export ISISPAR=$ISISREL                 ①

export ISISCPT=$(echo $ISISART | cut -d- -f2)
export ISISCPN=$(echo $ISISART | cut -d- -f1)

env | grep ISIS | sort
```

① `$ISISPAR` is the version of the Apache Isis core that will act as the archetype's parent. Usually this is the same as `$ISISREL`.

Check the example app

Update the parent `pom.xml` to reference the *released* version of Apache Isis core, eg:

```
<properties>
    <isis.version>1.16.0</isis.version>
    ...
</properties>
```

Previously at this point we used to check for and fix any missing license header notices; however it doesn't make sense for the archetype to include the Apache rat-plugin, so this has been removed.

Finally, double check that the app

- builds:

```
mvn clean install
```

- can be run from an IDE
  - mainClass=`org.apache.isis.WebServer`
  - args=`-m domainapp.application.manifest.DomainAppAppManifestWithFixtures`
  - run before: `mvn -f pom-jdo-enhance-all.xml datanucleus:enhance -o` in the root module
- can be run using the mvn jetty plugin:

```
mvn -pl webapp jetty:run
```

- can be packaged and run using the mvn jetty-console plugin:

```
mvn install -Dmavenmixin-jettyconsole
mvn antrun:run -Dmavenmixin-jettyconsole
```

- can be deployed as a WAR

```
cp webapp/target/simpleapp.war $CATALINA_HOME/webapps/ROOT.war
pushd $CATALINA_HOME/bin
sh startup.sh
tail -f ../logs/catalina.out
```

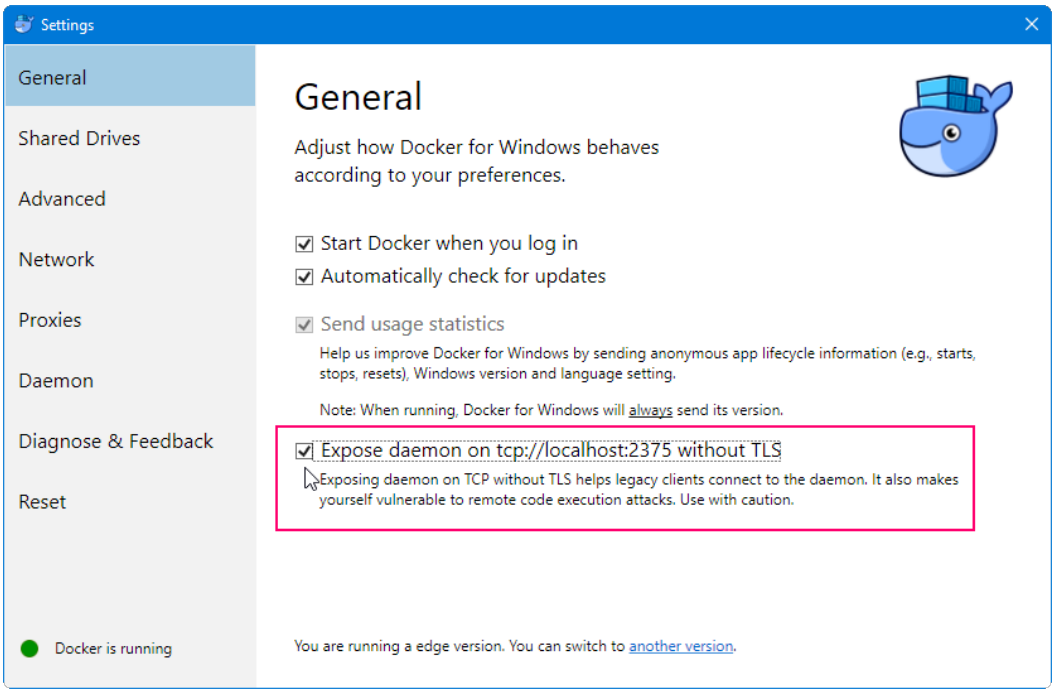
  quit using:

```
sh shutdown.sh
popd
```

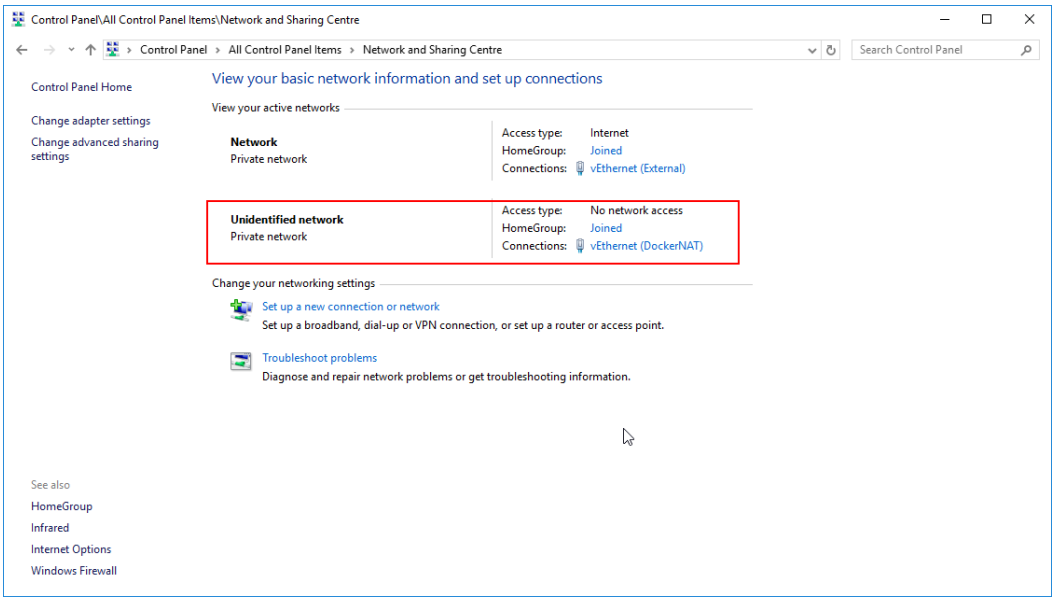
- can be packaged and run using Docker:

```
mvn install -Dmavenmixin-docker -D docker-plugin.imageName=test/simpleapp
docker container run -p 8080:8080 -d test/simpleapp
```

On Windows, you may need to enable the Docker daemon first, otherwise the "mvn install" command above will fail:



You might also need to ensure that the "Unidentified networks" are configured to be private:



This can be done using the "Local Security Policy" application.

This too should be accessed at localhost:8080.

The Kitematic UI tool is handy for viewing and interacting with running containers.

In each case, check the about page and confirm has been built against non-SNAPSHOT versions of the Apache Isis jars.

Create the archetype

Make sure you are in the correct directory and environment variables are correct.

To recreate the **simpleapp** archetype:

```
cd example/application/simpleapp
env | grep ISIS | sort
```

Then, run the script:

```
sh ../../../scripts/recreate-archetype.sh $ISISJIRA
```

The script automatically commits changes; if you wish use `git log` and `git diff` (or a tool such as SourceTree) to review changes made.

4.5.2. Release prepare

Switch to the **archetype** directory and execute the `release:prepare`:

```
cd ../../../example/archetype/$ISISCPN
rm -rf $ISISTMP/checkout
mvn release:prepare -P apache-release \
                -DreleaseVersion=$ISISREL \
                -DdevelopmentVersion=$ISISDEV \
                -Dtag=$ISISART-$ISISREL
```

Post-prepare sanity check

This is a good point to test the archetype; nothing has yet been uploaded.

In a different session, create a new app from the archetype. First set up environment variables:

```
export ISISTMP=/c/tmp    # or as required
export ISISCPN=simpleapp
env | grep ISIS | sort
```

Then generate a new app from the archetype:

```
rm -rf $ISISTMP/test-$ISISCPN

mkdir $ISISTMP/test-$ISISCPN
cd $ISISTMP/test-$ISISCPN
mvn archetype:generate  \
    -D archetypeCatalog=local \
    -D groupId=com.mycompany \
    -D artifactId=myapp \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=$ISISCPN-archetype
```

Build the newly generated app and test:

```
cd myapp
mvn clean install -o
mvn -pl webapp jetty:run              # runs as mvn jetty plugin
```

Release Perform (upload)

Back in the original session (in the **archetype** directory, `example/archetype/$ISISCPN`), execute `release:perform`:

```
mvn release:perform -P apache-release \
                -DworkingDirectory=$ISISTMP/checkout
```

This will upload the artifacts to the ASF Nexus repository.

4.5.3. Releasing `helloworld` archetype

We now repeat the procedure for the `helloworld` example app.

Start by switching to the directory containing the `helloworld` example:

```
cd ../../../example/application/helloworld
```

Setup environment variables

Update additional environment variables for the `helloworld-archetype` artifact:

```
export ISISART=helloworld-archetype
export ISISPAR=$ISISREL

export ISISCPT=$(echo $ISISART | cut -d- -f2)
export ISISCPN=$(echo $ISISART | cut -d- -f1)

env | grep ISIS | sort
```

Check the example app

Update the parent `pom.xml` to reference the *released* version of Apache Isis core, eg:

```
<properties>
    <isis.version>1.16.0</isis.version>
    ...
</properties>
```

Finally, double check that the app

- builds:

  ```
  mvn clean install
  ```

- can be run from an IDE
  - mainClass=`org.apache.isis.WebServer`
  - args=`-m domainapp.application.HelloWorldAppManifest`
  - run before: `mvn datanucleus:enhance -o` in the root module
- can be run using the mvn jetty plugin:

  ```
  mvn jetty:run
  ```

- can be deployed as a WAR

  ```
  cp target/helloworld.war $CATALINA_HOME/webapps/ROOT.war
  pushd $CATALINA_HOME/bin
  sh startup.sh
  tail -f ../logs/catalina.out
  ```

  quit using:

```
sh shutdown.sh
popd
```

This too should be accessed at localhost:8080.

In each case, check the about page and confirm has been built against non-SNAPSHOT versions of the Apache Isis jars.

Create the archetype

Make sure you are in the correct directory and environment variables are correct.

To recreate the **helloworld** archetype:

```
cd example/application/helloworld
env | grep ISIS | sort
```

Then, run the script:

```
sh ../../../scripts/recreate-archetype.sh $ISISJIRA
```

The script automatically commits changes; if you wish use `git log` and `git diff` (or a tool such as SourceTree) to review changes made.

4.5.4. Release prepare

Switch to the **archetype** directory and execute the `release:prepare`:

```
cd ../../../example/archetype/$ISISCPN
rm -rf $ISISTMP/checkout
mvn release:prepare -P apache-release \
                -DreleaseVersion=$ISISREL \
                -DdevelopmentVersion=$ISISDEV \
                -Dtag=$ISISART-$ISISREL
```

Post-prepare sanity check

This is a good point to test the archetype; nothing has yet been uploaded.

In a different session, create a new app from the archetype. First set up environment variables:

```
export ISISTMP=/c/tmp    # or as required
export ISISCPN=helloworld
env | grep ISIS | sort
```

Then generate a new app from the archetype:

```
rm -rf $ISISTMP/test-$ISISCPN

mkdir $ISISTMP/test-$ISISCPN
cd $ISISTMP/test-$ISISCPN
mvn archetype:generate  \
    -D archetypeCatalog=local \
    -D groupId=com.mycompany \
    -D artifactId=myapp \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=$ISISCPN-archetype
```

Build the newly generated app and test:

```
cd myapp
mvn clean install -o
mvn jetty:run
```

Release Perform (upload)

Back in the original session (in the **archetype** directory, `example/archetype/$ISISCPN`), execute `release:perform`:

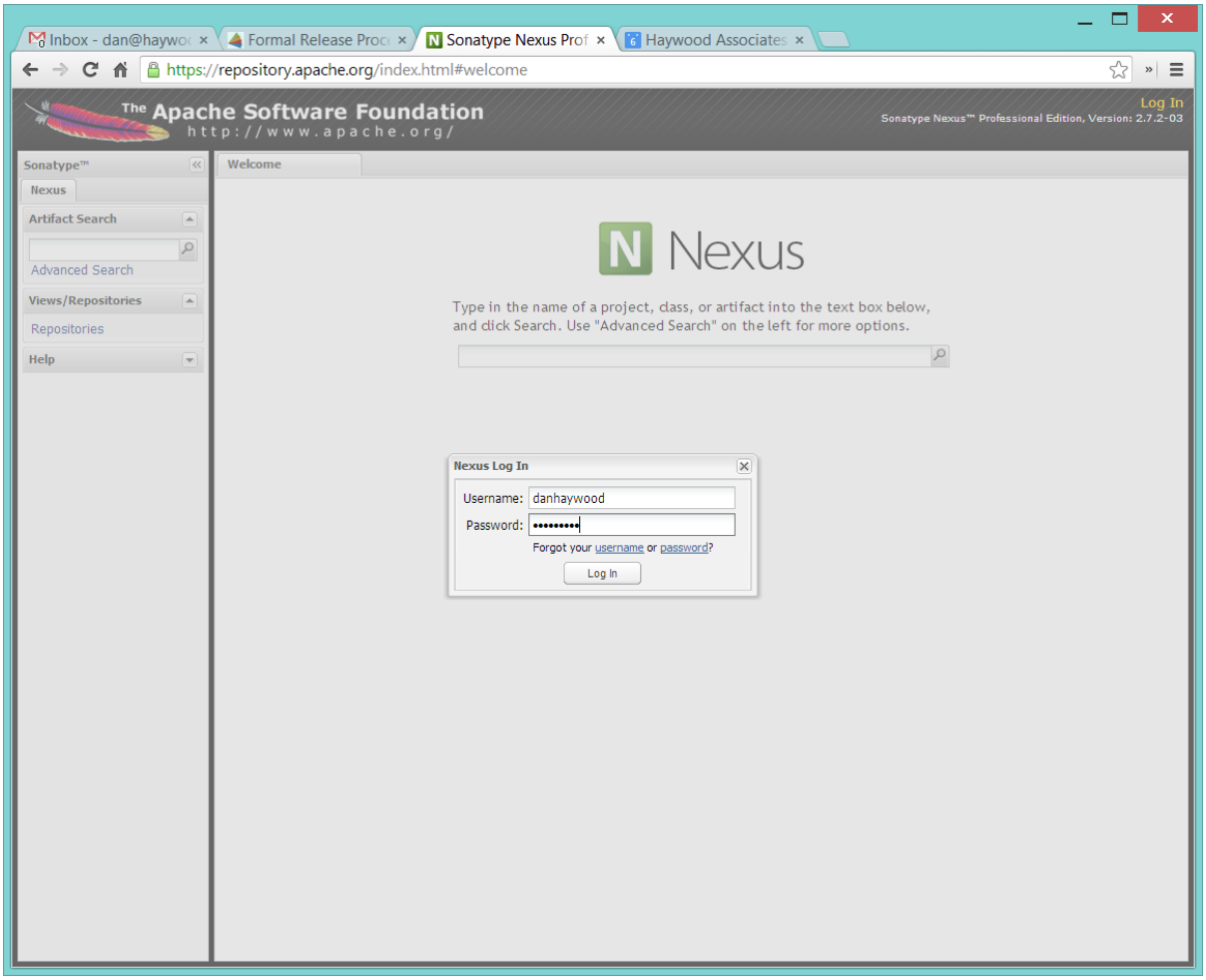
```
mvn release:perform -P apache-release \
                -DworkingDirectory=$ISISTMP/checkout
```

This will upload the artifacts to the ASF Nexus repository.

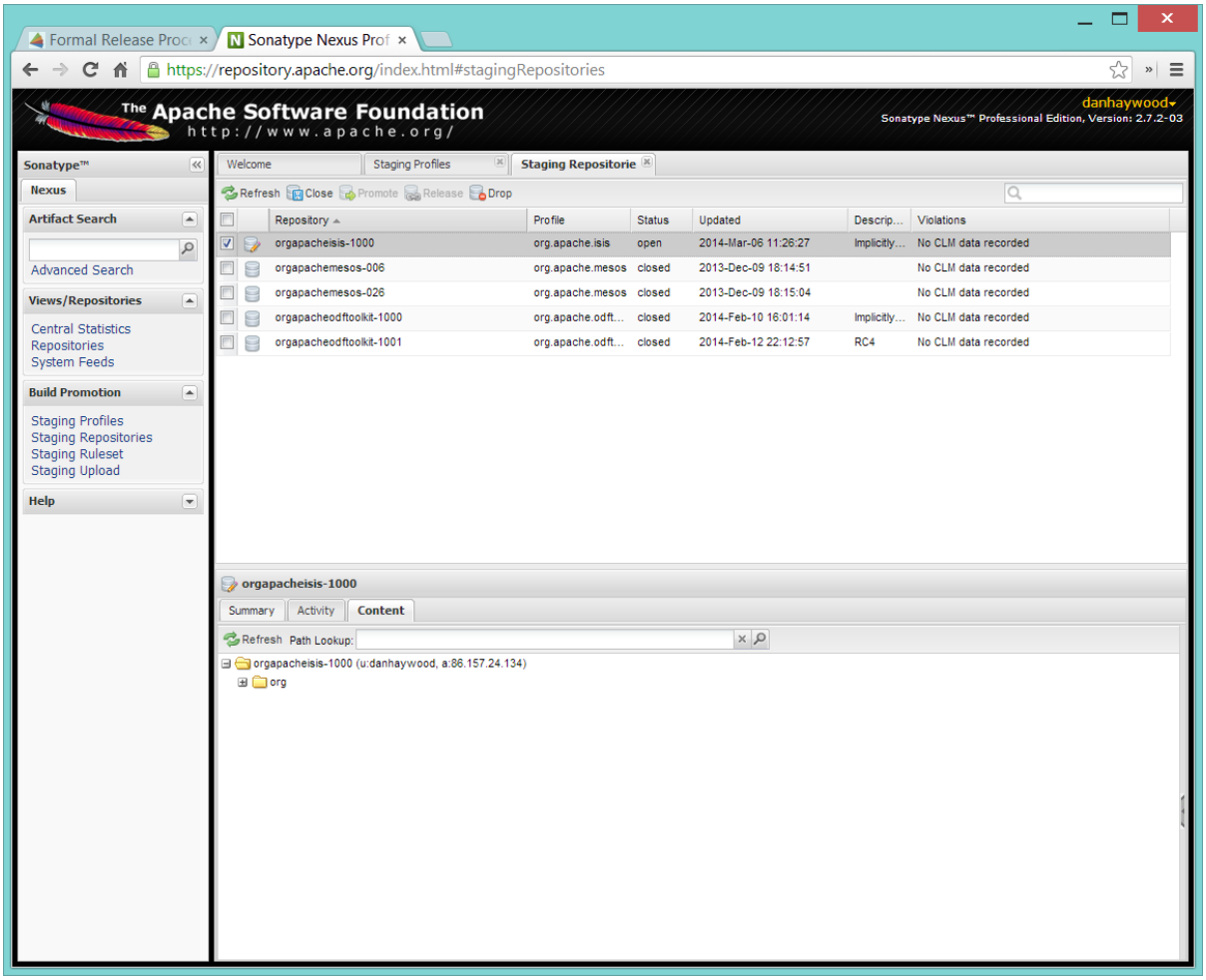
4.6. Check/Close Staging Repo

The `mvn release:perform` commands will have put release artifacts for both `core` and the `simpleapp` archetype into a newly created staging repository on the ASF Nexus repository server.

Log onto repository.apache.org (using your ASF LDAP account):



And then check that the release has been staged (select `staging repositories` from left-hand side):

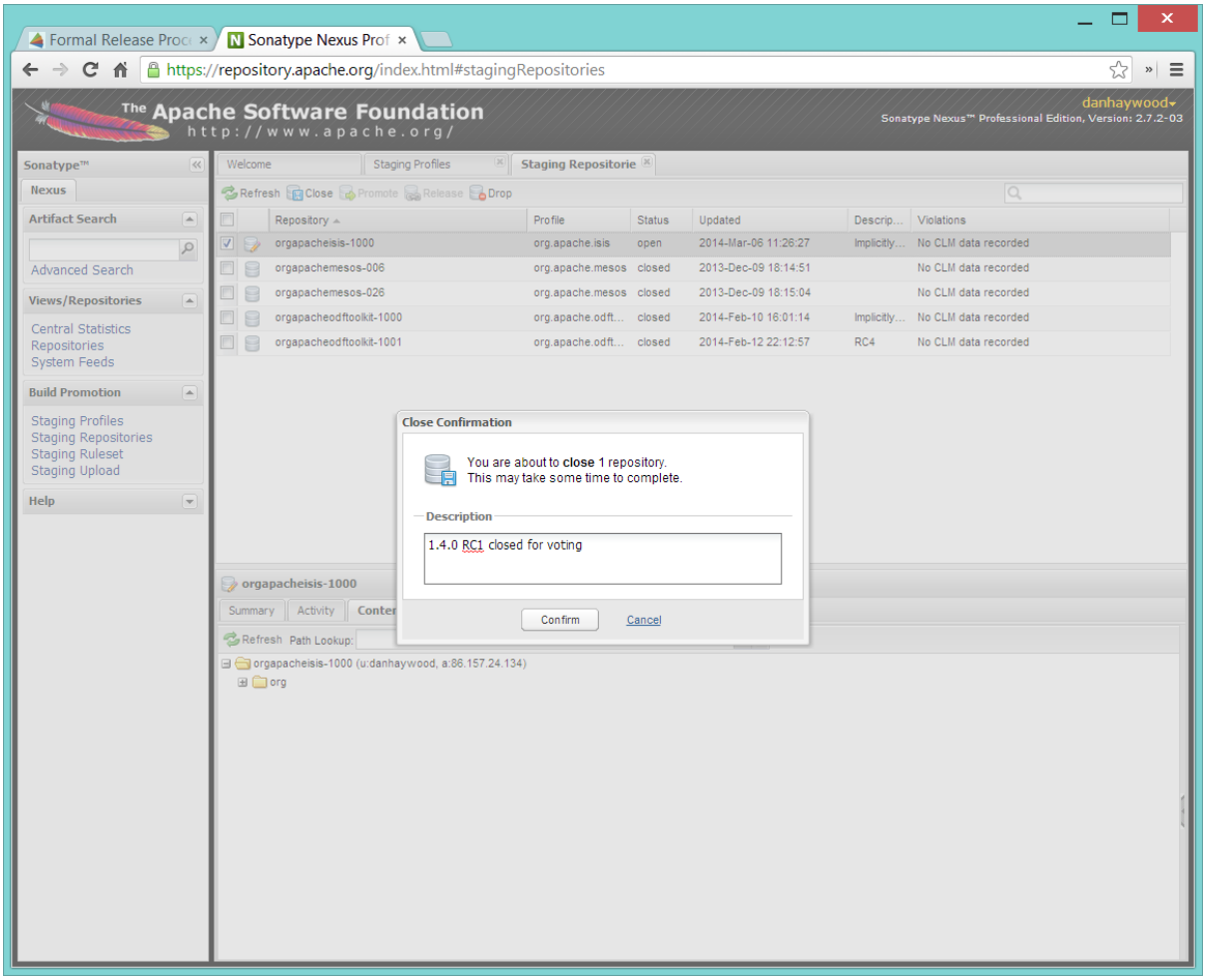


If nothing appears in a staging repo you should stop here and work out why.

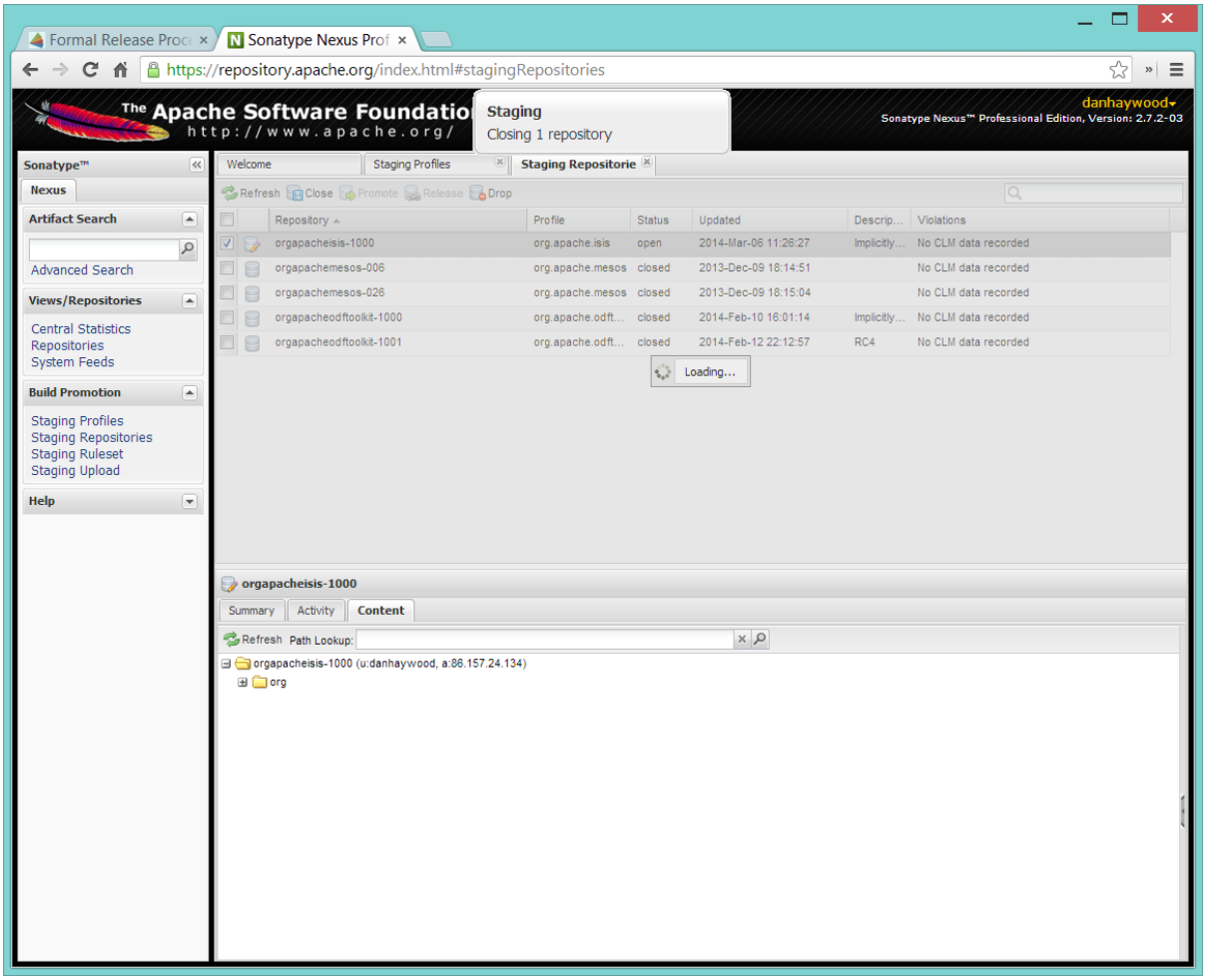
Assuming that the repo has been populated, make a note of its repo id; this is needed for the voting thread. In the screenshot above the id is `org.apache.isis-008`.

After checking that the staging repository contains the artifacts that you expect you should close the staging repository. This will make it available so that people can check the release.

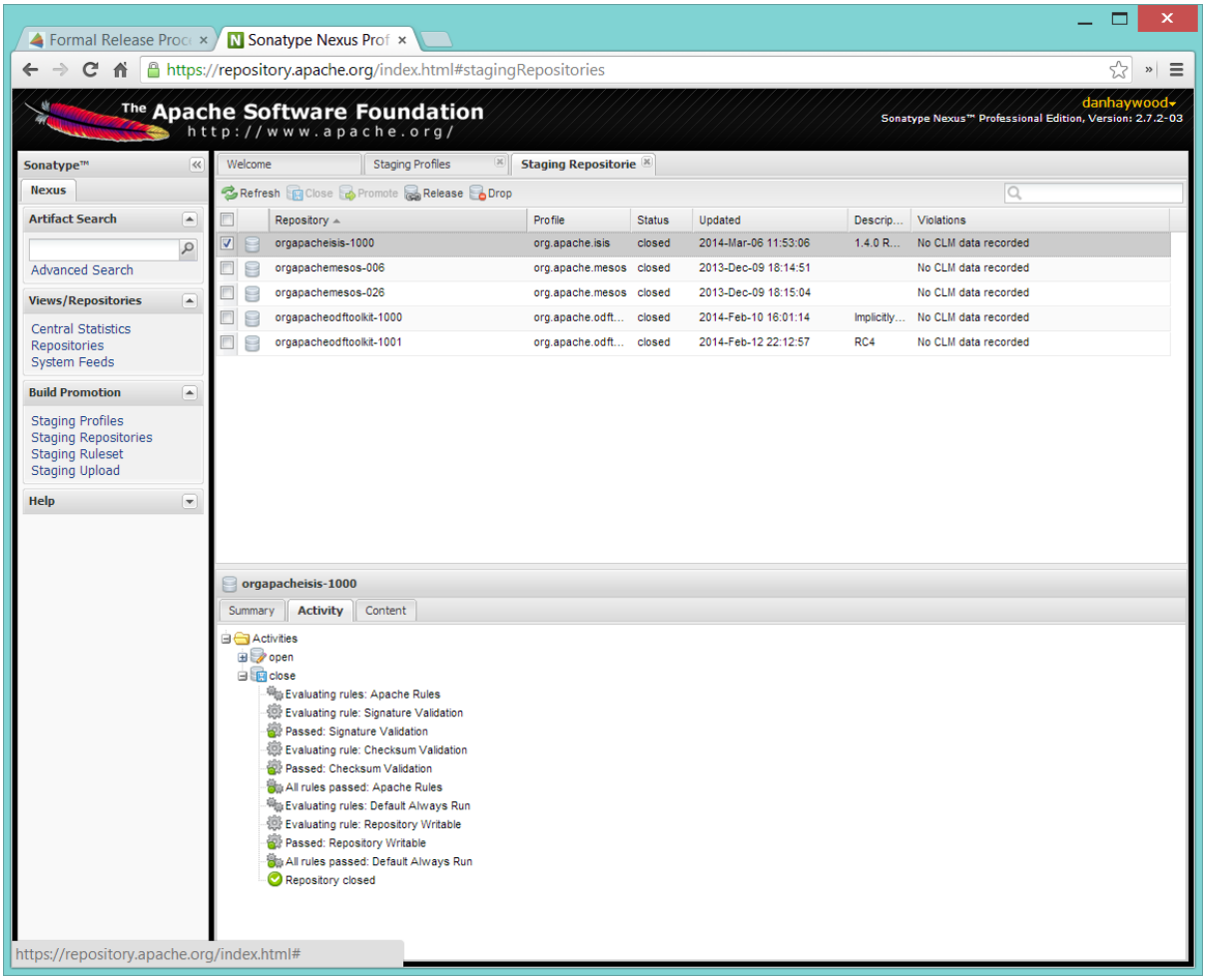
Press the Close button and complete the dialog:



Nexus should start the process of closing the repository.

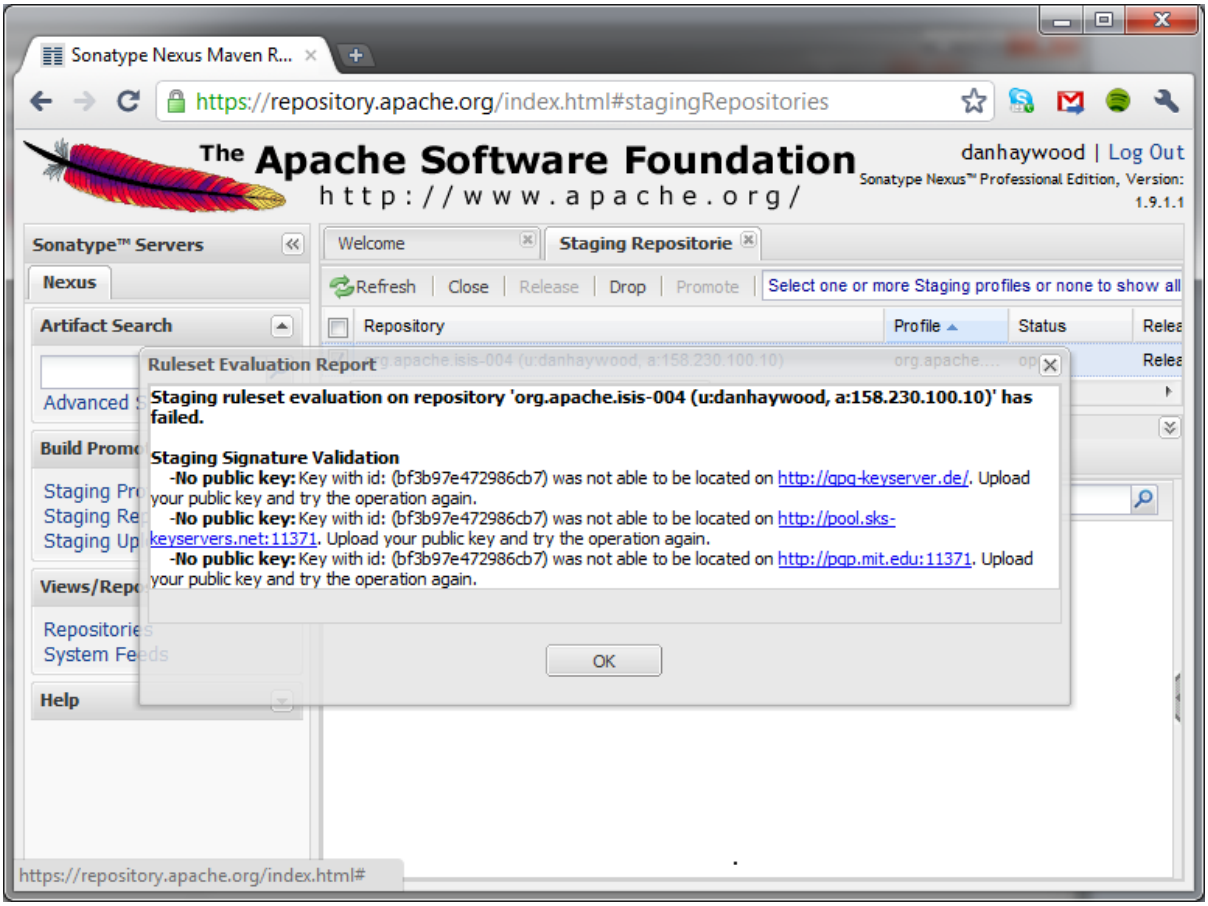


All being well, the close should (eventually) complete successfully (keep hitting refresh):



The Nexus repository manager will also email you with confirmation of a successful close.

If Nexus has problems with the key signature, however, then the close will be aborted:



Use `gpg --keyserver hkp://pgp.mit.edu --recv-keys nnnnnnnn` to confirm that the key is available.

Unfortunately, Nexus does not seem to allow subkeys to be used for signing. See Key Generation for more details.

4.7. Push branches

Push the release branch to origin:

```
git push -u origin $ISISBRANCH
```

and also push tags for both core and the archetype:

```
git push origin refs/tags/isis-$ISISREL:refs/tags/isis-$ISISREL-$ISISRC
git push origin refs/tags/simpleapp-archetype-$ISISREL:refs/tags/simpleapp-archetype-
$ISISREL-$ISISRC
git push origin refs/tags/helloworld-archetype-$ISISREL:refs/tags/helloworld-
archetype-$ISISREL-$ISISRC
git fetch
```

The remote tags aren't visible locally but can be seen online.

4.8. Voting

Once the artifacts have been uploaded, you can call a vote.

In all cases, votes last for 72 hours and require a +3 (binding) vote from members.

4.8.1. Start voting thread on dev@isis.apache.org

The following boilerplate is for a release of the Apache Isis Core. Adapt as required:

Use the following subject, eg:

```
[VOTE] Apache Isis Core release 1.16.0 RC1
```

And use the following body:

```
I've cut a release for Apache Isis Core and the two archetypes:

* Core 1.16.0
* HelloWorld Archetype 1.16.0
* SimpleApp Archetype 1.16.0

The source code artifacts have been uploaded to staging repositories on
repository.apache.org:

* http://repository.apache.org/content/repositories/orgapacheisis-
10xx/org/apache/isis/core/isis/1.16.0/isis-1.16.0-source-release.zip
* http://repository.apache.org/content/repositories/orgapacheisis-
10xx/org/apache/isis/archetype/helloworld-archetype/1.16.0/helloworld-archetype-
1.16.0-source-release.zip
* http://repository.apache.org/content/repositories/orgapacheisis-
10xx/org/apache/isis/archetype/simpleapp-archetype/1.16.0/simpleapp-archetype-1.16.0-
source-release.zip

For each zip there is a corresponding signature file (append .asc to the zip's url).

In the source code repo the code has been tagged as isis-1.16.0-RC1, helloworld-
archetype-1.16.0-RC1 and simpleapp-archetype-1.16.0-RC1; see https://git-wip-
us.apache.org/repos/asf?p=isis.git

For instructions on how to verify the release (build from binaries and/or use in Maven
directly), see https://isis.apache.org/guides/cgcom/cgcom.html#_cgcom_verifying-
releases

Please verify the release and cast your vote.  The vote will be open for a minimum of
72 hours.

[ ] +1
[ ]  0
[ ] -1
```

Remember to update:

- the version number (1.16.0 or whatever)
- the release candidate number (`RC1` or whatever)
- the repository id, as provided by Nexus earlier (`orgapacheisis-10xx` or whatever)

Note that the email also references the procedure for other committers to verify the release.

Chapter 5. Verifying a Release

The release process consists of:

- the release manager cutting the release
- members of the Apache Isis PMC verifying and voting on the release (documented below)
- the release manager performing post-release tasks, for either a successful or an unsuccessful vote.

This section describes some guidance on what a voter (members of the Apache Isis PMC and anyone else who wishes) is expected to do before casting their vote in order to verify a release.

5.1. Background

Whenever a release manager announces a vote on a release (as per the release process) on the dev mailing list, it is the responsibility of the project's PMC to cast their vote on the release. Anyone else can also vote, but only members of the Apache Isis PMC's vote are binding.

Per this ASF documentation, the legal requirements for an ASF release are:

- a source zip file + corresponding signature (signed by the release manager, which is in the ASF web of trust and in our KEYS file)
- all source files have the Apache license (this is ensured by the running of the rat plugin prior to release; you could run it on the unzipped source)
- all dependencies are appropriately licensed; see the `DEPENDENCIES` file which is automatically generated from the POMs plus the supplemental-models.xml file

Note that the binaries are *not* an ASF release, they merely exist on the Maven central repo as a convenience. That said, you might also want to verify the release by pulling the binaries from the Maven staging repository. Details of how to do this are also documented below.

5.2. Prerequisites

To verify the source ZIP files, you will need to have imported the public keys used for signing Apache Isis releases. These can be downloaded from the root of the Apache Isis source tree.

Since the Apache Isis source is mirrored on github.com, you can just use:

```
curl http://www.apache.org/dist/isis/KEYS > /tmp/KEYS
gpg --import /tmp/KEYS
```

Also, we will be rebuilding Isis from source. Therefore delete all Isis artifacts from your local Maven repo:

```
rm -rf ~/.m2/repository/org/apache/isis
```

5.3. Verifying source artifacts

You can either verify the source artifacts manually, or use a script that automates the steps.

5.3.1. Manual procedure

The following section describes the steps to perform to manually verify a release.

Download the artifacts

Download both the ZIP and .ASC files from the location specified in the voting email. To verify that the signature is correct, use:

```
gpg --verify isis-x.y.z.zip.asc isis-x.y.z.zip
```

Building source artifacts

Assuming the ZIP file verifies, it should be unpacked, and then the artifact built from source.

To build Apache Isis core, first download any dependencies:

```
mvn dependency:go-offline
```

Check that no Isis artifacts have yet been downloaded, ie there is no `~/.m2/org/repository/org/apache/isis` directory. If there are, it could indicate that the release being verified incorrectly references previous versions of Apache Isis

Assuming all is ok, build using the `-o` offline flag:

```
mvn clean install -o
```

Confirm that the versions of the Isis artifacts now cached in your local repository are correct.

Verifying binary artifacts

You can verify the binary releases by configuring your local Maven install to point to the Maven staging repository (or repositories) and then using them, eg to run the HelloWorld archetype or the SimpleApp archetype and running the resultant app.

Configuring your local Maven install amounts to updating the `~/.m2/settings.xml` file:

```
<profiles>
    <profile>
        <id>verify-isis</id>
        <repositories>
            <repository>
                <id>isis-core-staging</id>
                <name>Isis Core Staging</name>
                <releases>
                    <enabled>true</enabled>
                    <updatePolicy>always</updatePolicy>
                    <checksumPolicy>warn</checksumPolicy>
                </releases>
                <url>http://repository.apache.org/content/repositories/orgapacheisis-
10xx</url>
                <layout>default</layout>
            </repository>
            ...
        </repositories>
    </profile>
    ...
</profiles>
<activeProfiles>
    <activeProfile>verify-isis</activeProfile>
    ...
</activeProfiles>
```

where the repository URL is as provided in the VOTE email. If there is more than one repository (as is sometimes the case if multiple components have been released), then repeat the <repository> section for each.

Once the vote has completed, the staging repositories will be removed and so you should deactive the profile (comment out the `<activeProfile>` element). If you forget to deactive the profile, there should be no adverse effects; Maven will just spend unnecessary cycles attempting to hit a non-existent repo.

5.3.2. Automated procedure

To save some time in verifying an Apache Isis release we've assembled a script to automate the process. The script is tested on Mac OSX and Linux. Windows users can use Cygwin or msysgit.

It's *recommended* that you start this process in an empty directory:

```
mkdir ~/verify-isis-release
cd ~/verify-isis-release
```

Copy script to local machine

Download this gist, save to `verify-isis-release.sh`.

Create an input file

The input file is a plain `urls.txt` file containing all urls to the packages to be verified. Here's a sample of the release from Apache Isis 1.12.0:

```
http://repository.apache.org/content/repositories/orgapacheisis-
1058/org/apache/isis/core/isis/1.12.0/isis-1.12.0-source-release.zip
http://repository.apache.org/content/repositories/orgapacheisis-
1059/org/apache/isis/archetype/helloworld-archetype/1.12.0/helloworld-archetype-
1.12.0-source-release.zip
http://repository.apache.org/content/repositories/orgapacheisis-
1059/org/apache/isis/archetype/simpleapp-archetype/1.12.0/simpleapp-archetype-1.12.0-
source-release.zip
```

You will find the actual list of URLs to be verified in the vote mail on the dev mailing list.

Execute the script

Execute...

```
sh verify-isis-release.sh
```

... and get yourself a cup of coffee.

5.4. (Optional) Creadur Tools

The Apache Creadur project exists to provide a set of tools to ensure compliance with Apache's licensing standards.

The main release auditing tool, Apache RAT is used in the release process.

Creadur's remaining tools - Tentacles and Whisker - are to support the verification process.

For example, Tentacles generates a report called `archives.html`. This lists all of the top-level binaires, their `LICENSE` and `NOTICE` files and any `LICENSE` and `NOTICE` files of any binaries they may contain.

Validation of the output at this point is all still manual. Things to check include:

- any binaries that contain no LICENSE and NOTICE files
- any binaries that contain more than one LICENSE or NOTICE file

In this report, each binary will have three links listed after its name '(licenses, notices, contents)'

5.5. Test the archetypes

Assuming that everything builds ok, then test the archetypes (adjust version as necessary):

- First the `helloworld` archetype:

```
mvn archetype:generate  \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=helloworld-archetype \
    -D archetypeVersion=1.16.0 \
    -D groupId=com.mycompany \
    -D artifactId=myapp \
    -D version=1.0-SNAPSHOT \
    -B \
    -o

cd myapp
mvn clean install jetty:run -o
```

  Adjust the version as necessary.

- Next, the `simpleapp` archetype:

```
mvn archetype:generate  \
    -D archetypeGroupId=org.apache.isis.archetype \
    -D archetypeArtifactId=simpleapp-archetype \
    -D archetypeVersion=1.16.0 \
    -D groupId=com.mycompany \
    -D artifactId=myapp \
    -D version=1.0-SNAPSHOT \
    -B \
    -o

cd myapp
mvn clean install -o && mvn -pl jetty:run -o
```

  Adjust the version as necessary.

If the archetypes run up ok, then it's time to vote!

5.6. Casting a Vote

When you have made the above checks (and any other checks you think may be relevant), cast your vote by replying to the email thread on the mailing list.

If you are casting `-1`, please provide details of the problem(s) you have found.

Chapter 6. Post Release (Successful)

The release process consists of:

- the release manager cutting the release
- members of the Apache Isis PMC verifying and voting on the release
- the release manager performing post-release tasks, for either a successful or an unsuccessful vote (former documented below)

For a vote to succeed, there must be +3 votes from PMC members, and the vote must have been open at least 72 hours. If there are not +3 votes after this time then it is perfectly permissible to keep the vote open longer.

This section describes the steps to perform if the vote has been successful.

6.1. Inform dev ML

Post the results to the `dev@isis.a.o` mailing list:

```
[RESULT] [VOTE] Apache Isis Core release 1.16.0
```

using the body (alter last line as appropriate):

```
The vote has completed with the following result :

  +1 (binding): ... list of names ...
  +1 (non binding): ... list of names ...

  -1 (binding): ... list of names ...
  -1 (non binding): ... list of names ...

The vote is SUCCESSFUL.

I'll now go ahead and complete the post-release activities.
```

6.2. Update tags

Replace the `-RCn` tag with another without the qualifier.

You can do this using the `scripts/promoterctag.sh` script; for example:

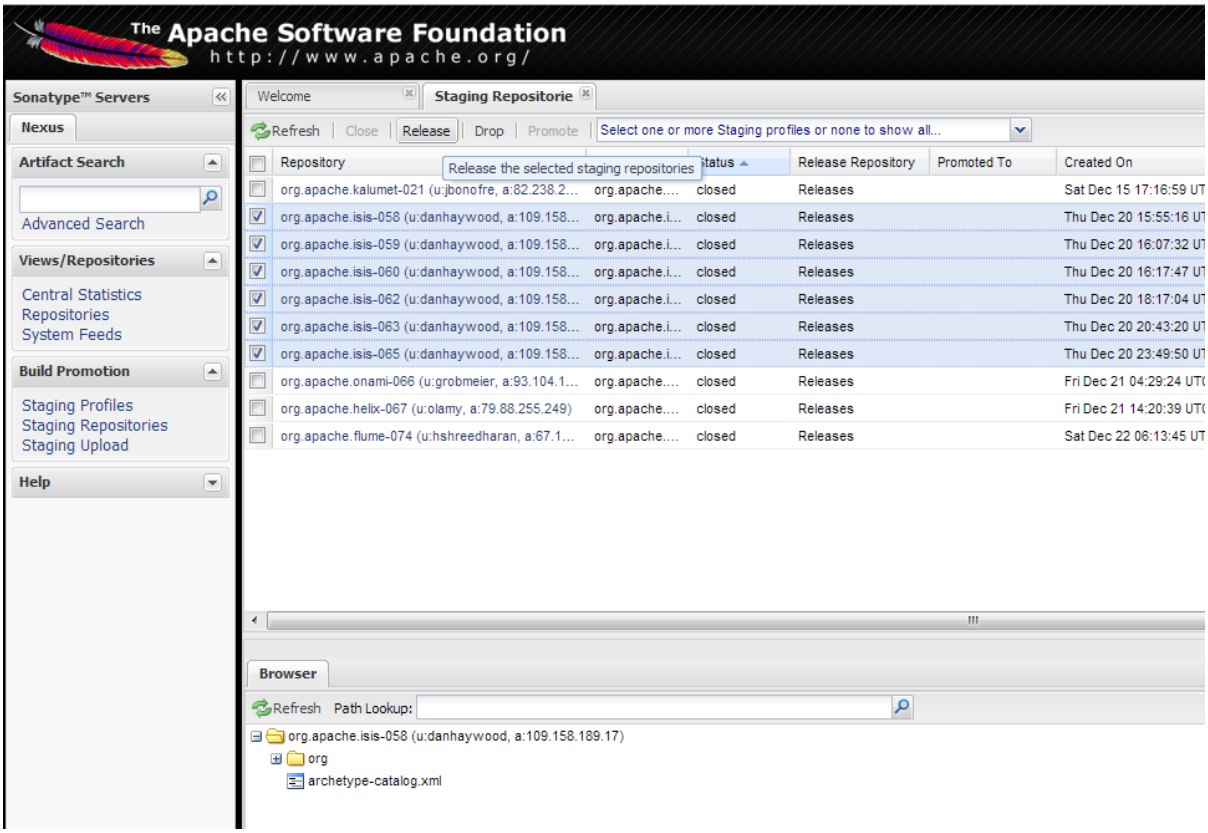
```
sh scripts/promoterctag.sh isis-1.16.0 RC1
sh scripts/promoterctag.sh helloworld-archetype-1.16.0 RC1
sh scripts/promoterctag.sh simpleapp-archetype-1.16.0 RC1
```

This script pushes the tag under `refs/tags/rel`. As per Apache policy (communicated on 10th Jan 2016 to Apache PMCs), this path is 'protected' and is unmodifiable (guaranteeing the provenance that the ASF needs for releases).

Then, continue onto the next section for the steps to promote and announce the release.

6.3. Release to Maven Central

From the ASF Nexus repository, select the staging repository and select 'release' from the top menu.



This moves the release artifacts into an Apache releases repository; from there they will be automatically moved to the Maven repository.

6.4. Release Source Zip

As described in the Apache documentation, each Apache TLP has a `release/TLP-name` directory in the distribution Subversion repository at https://dist.apache.org/repos/dist. Once a release vote passes, the release manager should `svn add` the artifacts (plus signature and hash files) into this location. The release is then automatically pushed to http://www.apache.org/dist/ by `svnpubsub`. Only the most recent release of each supported release line should be contained here, old versions should be deleted.

Each project is responsible for the structure of its directory. The directory structure of Apache Isis reflects the directory structure in our git source code repo:

```
isis/
  core/
  example/
    archetype/
      simpleapp/
```

If necessary, checkout this directory structure:

```
svn co https://dist.apache.org/repos/dist/release/isis isis-dist
```

Next, add the new release into the appropriate directory, and delete any previous release. The `upd.sh` script (also downloadable from this gist) can be used to automate this:

```
old_ver=$1
new_ver=$2


# constants
repo_root=https://repository.apache.org/content/repositories/releases/org/apache/isis

zip="source-release.zip"
asc="$zip.asc"
md5="$zip.md5"


#
# isis-core
#
type="core"
fullname="isis"
pushd isis-core

curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$asc
svn add $fullname-$new_ver-$asc
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$md5
svn add $fullname-$new_ver-$md5
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$zip
svn add $fullname-$new_ver-$zip

svn delete $fullname-$old_ver-$asc
svn delete $fullname-$old_ver-$md5
svn delete $fullname-$old_ver-$zip

popd


#
# helloworld-archetype
```

```
#
type="archetype"
fullname="helloworld-archetype"
pushd $type/$fullname

curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$md5
svn add $fullname-$new_ver-$md5
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$asc
svn add $fullname-$new_ver-$asc
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$zip
svn add $fullname-$new_ver-$zip

svn delete $fullname-$old_ver-$md5
svn delete $fullname-$old_ver-$asc
svn delete $fullname-$old_ver-$zip

popd


#
# simpleapp-archetype
#
type="archetype"
fullname="simpleapp-archetype"
pushd $type/$fullname

curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$md5
svn add $fullname-$new_ver-$md5
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$asc
svn add $fullname-$new_ver-$asc
curl -O $repo_root/$type/$fullname/$new_ver/$fullname-$new_ver-$zip
svn add $fullname-$new_ver-$zip

svn delete $fullname-$old_ver-$md5
svn delete $fullname-$old_ver-$asc
svn delete $fullname-$old_ver-$zip

popd
```

```
sh upd.sh 1.15.1 1.16.0
```

The script downloads the artifacts from the Nexus release repository, adds the artifacts to subversion and deletes the previous version.

Double check that the files are correct; there is sometimes a small delay in the files becoming available in the release repository. It should be sufficient to check just the `md5` or `.asc` files that these look valid (aren't HTML 404 error pages):

```
vi `find . -name *.md5`
```

Assuming all is good, commit the changes:

```
svn commit -m "publishing isis source releases to dist.apache.org"
```

If the files are invalid, then revert using `svn revert . --recursive` and try again in a little while.

6.5. Update JIRA

6.5.1. Generate Release Notes

From the root directory, generate the release notes for the current release, in Asciidoc format; eg:

```
sh scripts/jira-release-notes.sh ISIS 1.16.0 > /tmp/1
```

6.5.2. Close tickets

Close all JIRA tickets for the release, or moved to future releases if not yet addressed. Any tickets that were partially implemented should be closed, and new tickets created for the functionality on the ticket not yet implemented.

6.5.3. Mark the version as released

In JIRA, go to the administration section for the Apache Isis project and update the version as being released.

In the Kanban view this will have the effect of marking all tickets as released (clearing the "done" column).

6.5.4. Create new JIRA

Create a new JIRA ticket as a catch-all for the *next* release.

6.5.5. Update the ASF Reporter website

Log the new release in the ASF Reporter website.

6.6. Update website

Update the Apache Isis (asciidoc) website:

- Paste in the JIRA-generated release notes generated above, adding to top of `adocs/documentation/src/main/asciidoc/release-notes.adoc`. Also add a summary line for the release.

- Search for any `-SNAPSHOT` suffices, and remove
- Search these release procedures, and update any hard-coded reference to the release to the next release (so when they are followed next time the text will be correct).
- Update the downloads page with a link to the source release zip file (under https://dist.apache.org/repos/dist/release/isis)
- Update any pages (`.adoc`, `.md`, `.html` etc) that describe how to run the archetype, and ensure they reference the correct version.

  A search for `archetypeGroupId=org.apache.isis.archetype` should find these pages.

- update the DOAP RDF file (which provides a machine-parseable description of the project) should also be updated with details of the new release. Validate using the W3C RDF Validator service.

  For more information on DOAP files, see these Apache policy docs.

- Update the STATUS file (in root of Apache Isis' source) should be updated with details of the new release.

Don't forget to commit the `.adoc` changes and publish to the isis-site repo.

6.7. Announce the release

Announce the release to users mailing list.

For example, for a release of Apache Isis Core, use the following subject:

```
[ANN] Apache Isis version 1.16.0 Released
```

And use the following body (summarizing the main points as required):

```
The Apache Isis team is pleased to announce the release of Apache Isis v1.16.0.

New features in this release include:
* ...

Full release notes are available on the Apache Isis website at [1].
Please also read the migration notes [2].

You can access this release directly from the Maven central repo [3].
Alternatively, download the release and build it from source [4].

Enjoy!

--The Apache Isis team

[1] http://isis.apache.org/release-notes/release-notes.html#_release-notes_1.16.0
[2] http://isis.apache.org/migration-notes/migration-notes.html#_migration-
notes_1.15.0-to-1.16.0
[3] http://search.maven.org
[4] http://isis.apache.org/downloads.html
```

6.8. Blog post

Log onto the Apache blog and create a new post. Copy-n-paste the above mailing list announcement should suffice.

6.9. Merge in release branch

Because we release from a branch, the changes made in the branch (changes to `pom.xml` made by the `maven-release-plugin`, or any manual edits) should be merged back from the release branch back into the `master` branch:

```
git checkout master                           # update master with latest
git pull
git merge release-1.16.0-RC1                  # merge branch onto master
git push origin --delete release-1.16.0-RC1   # remote branch no longer needed
git branch -d release-1.16.0-RC1              # branch no longer needed
```

Finally, update helloworld's `pom.xml` and simpleapp's root `pom.xml` to reference the next SNAPSHOT release (`1.17.0-SNAPSHOT`)

6.10. Update dependencies

With the release complete, now is a good time to bump versions of dependencies (so that there is a full release cycle to identify any possible issues).

You will probably want to create a new JIRA ticket for these updates (or if minor then use the "catch-all" JIRA ticket raised earlier for the next release).

6.10.1. Update parent of Core

Check (via search.maven.org) whether there is a newer version of the Apache parent `org.apache:apache`.

If there is, update the `<version>` in the `<parent>` element in the parent POM to match the newer version:

```
<parent>
    <groupId>org.apache</groupId>
    <artifactId>apache</artifactId>
    <version>NN</version>
    <relativePath />
</parent>
```

where `NN` is the updated version number.

6.10.2. Update plugin versions

The `maven-versions-plugin` should be used to determine if there are newer versions of any of the plugins used to build Apache Isis. Since this goes off to the internet, it may take a minute or two to run:

```
mvn versions:display-plugin-updates > /tmp/foo
grep "\->" /tmp/foo | /bin/sort -u
```

Review the generated output and make updates as you see fit. (However, if updating, please check by searching for known issues with newer versions).

6.10.3. Update dependency versions

The `maven-versions-plugin` should be used to determine if there are newer versions of any of Isis' dependencies. Since this goes off to the internet, it may take a minute or two to run:

```
mvn versions:display-dependency-updates > /tmp/foo
grep "\->" /tmp/foo | /bin/sort -u
```

Update any of the dependencies that are out-of-date. That said, do note that some dependencies may show up with a new dependency, when in fact the dependency is for an old, badly named version. Also, there may be new dependencies that you do not wish to move to, eg release candidates or milestones.

For example, here is a report showing both of these cases:

```
[INFO]   asm:asm ..................................... 3.3.1 -> 20041228.180559
[INFO]   commons-httpclient:commons-httpclient ........... 3.1 -> 3.1-jbossorg-1
[INFO]   commons-logging:commons-logging ......... 1.1.1 -> 99.0-does-not-exist
[INFO]   dom4j:dom4j ................................. 1.6.1 -> 20040902.021138
[INFO]   org.datanucleus:datanucleus-api-jdo ................ 3.1.2 -> 3.2.0-m1
[INFO]   org.datanucleus:datanucleus-core .................. 3.1.2 -> 3.2.0-m1
[INFO]   org.datanucleus:datanucleus-jodatime ............... 3.1.1 -> 3.2.0-m1
[INFO]   org.datanucleus:datanucleus-rdbms ................. 3.1.2 -> 3.2.0-m1
[INFO]   org.easymock:easymock ................................ 2.5.2 -> 3.1
[INFO]   org.jboss.resteasy:resteasy-jaxrs ............. 2.3.1.GA -> 3.0-beta-1
```

For these artifacts you will need to search Maven central repo directly yourself to confirm there are no newer dependencies not shown in this list.

6.11. Code formatting

This is also a good time to make source code has been cleaned up and formatted according to the Apache Isis and ASF conventions. Use this Eclipse template and this import order.

6.12. Push changes

Finally, push the changes up to origin:

```
git fetch    # check no new commits on origin/master
git push
```

6.13. Release (non-ASF) Modules

The (non-ASF) Incode Platform should also be released, as per their release guide.

Chapter 7. Post Release (Unsuccessful)

The release process consists of:

- the release manager cutting the release
- members of the Apache Isis PMC verifying and voting on the release
- the release manager performing post-release tasks, for either a successful or an unsuccessful vote (latter documented below).

If the vote did not succeed (did not achieve +3 votes after 72 hours and/or is unlikely to do so), then the vote should be closed and the following steps performed.

Note that a release manager may also decide to cancel a vote before 72 hours has elapsed (for example if an error is quickly discovered).

7.1. Inform dev ML

Post the results to the `dev@isis.a.o` mailing list.

For example, use the following subject for a vote on Apache Isis Core:

```
[RESULT] [VOTE] Apache Isis Core release 1.16.0
```

using the body (alter last line as appropriate):

```
The vote has completed with the following result :

  +1 (binding): _list of names_
  +1 (non binding): _list of names_

  -1 (binding): _list of names_
  -1 (non binding): _list of names_

The vote is UNSUCCESSFUL.
```

7.2. Tidy up branches

Tidy up remote branches in the git repo:

- delete the remote branch, for example:

```
git push --delete origin release-1.16.0-RC1
```

- delete the remote origin server's tags, for example:

```
git push --delete origin isis-1.16.0-RC1
git push --delete origin helloworld-archetype-1.16.0-RC1
git push --delete origin simpleapp-archetype-1.16.0-RC1
```

- delete the tags that were created locally, for example:

```
git tag -d isis-1.16.0
git tag -d helloworld-archetype-1.16.0
git tag -d simpleapp-archetype-1.16.0
```

7.3. Tidy up the Nexus repo

Drop staging repositories:

- drop the staging repository in Nexus

7.4. Reset

Finally, rewind the release branch to prior to the previous release candidate, and continue from there.

Chapter 8. Snapshot Releases

Snapshot releases allows the current `-SNAPSHOT` version of the `core` modules of the framework to be released to the Maven snapshot repository maintained by Apache Software Foundation.

Unless otherwise stated, you should assume that the steps described here are performed in the base directory of the module being released.

8.1. Prerequisites

Before you start, make sure you've defined the snapshots repo in your local `~/.m2/settings.xml` file:

```
<settings>
  <servers>
    <!-- To publish a snapshot of some part of Maven -->
    <server>
      <id>apache.snapshots.https</id>
      <username>xxxxxxx</username>
      <password>yyyyyyy</password>
    </server>
    ...
  </servers>
  ...
</settings>
```

where `xxxxxxx` and `yyyyyyy` are your Apache LDAP username and password. For more information, see these ASF docs.

{note It is also possible to configure to use `.ssh` secure keys, and thereby avoid hardcoding your Apache LDAP password into your `.m2/settings.xml` file. A description of how to do this can be found, for example, here. }

8.2. Sanity Check

Before deploying the snapshot, perform a quick sanity check.

First, delete all Isis artifacts from your local Maven repo:

```
rm -rf ~/.m2/repository/org/apache/isis
```

Next, check that the framework builds ok:

```
cd core
mvn clean install -o
```

Confirm that the versions of the Isis artifacts now cached in your local repository are correct (both those pulled down from Maven central repo, as well as those of the component built locally).

8.3. Deploy

Deploy the framework using:

```
cd core
mvn -D deploy=snapshot deploy
```

This will deploy all the modules that make up a release.

Expect this to take about 10 minutes, give or take.

To confirm that they are present, browse to Apache's Nexus repository manager and search for "isis".

Chapter 9. Interim Releases

The intent of an "interim" release is to allow a developer team to release their application based on a `-SNAPSHOT` version of the framework. Since `-SNAPSHOT` changes on a day-to-day basis, the idea is to tag a particular revision and to release this, thereby providing stability/traceability for the application being developed.

Whereas formal releases and snapshot releases are public (released through the Maven repository maintained by Apache Software Foundation), interim releases are non-public and rely on infrastructure provided by a developer team. The tagged release resides _not in the official Apache Isis git repository, but instead in a fork/clone maintained by the developer team.

The procedure developed here was put together for the team working on the Estatio app, which maintains its own fork on github and uses CloudBees as a Jenkins build server/private Maven repo.

9.1. Prerequisites

Create a remote fork/clone of the Apache Isis repository (eg using github or bitbucket or similar), and ensure that your local fork specifies this remote.

Also, set up a CI server against your fork/clone, to build against any branches called `origin/interim/*`. To build it should use the command:

```
mvn clean install -Dskip.app -Dskip.arch
```

that is, skipping the example simpleapp and archetype; only `core` framework is built

The CI server should then also publish the resultant artifacts to a local Maven repository. For example, Jenkins provides post-build plugins to perform such a task.

9.2. Sanity Check

Ensure that the framework builds ok using the same command that your CI server is set up to execute (see section above).

9.3. Release

Deploy the framework using:

```
sh interim-release.sh xxx yyyyy
```

where xxx is the most recent release of Isis (to act as the base), and yyyyy is the name of the remote.

For example,

```
sh interim-release.sh 1.13.0 incodehq
```

This will result in a new branch and tagged being pushed to the remote, with an appropriate version (details below).

As noted in the prereqs (above), the CI server configured should then detect the new branch (or tag, if you prefer), build the framework (skipping application and architecture, as in the prerequisites) and then publish the resultant artifacts to a private Maven repo.

9.3.1. Implementation details

The script itself:

- removes any local branches called `interim/*`
- creates a new branch called `interim/YYYYmmDD-HHMM`
  - eg `interim/1.13.0.20160909-0758`
- updates the version of the pom.xml to the baseline plus the date
  - eg `<version>1.13.0.20160909-0758</version>`
- commits the changes and tags the new commit
  - eg `isis-1.13.0.20160909-0758`
- removes any remote branches called `interim/*`
- pushes both the branch and the tag to the remote.

Chapter 10. Publishing the Docs

Apache Isis' documentation (meaning the website and the users' guide, the reference guide and this contributors' guide) is written using Asciidoc, specifically the Asciidoctor implementation.

The website and guides are created by running build tools (documented below) which create the HTML version of the site and guides. You can therefore easily check the documentation before raising a pull request (as a contributor) or publishing the site (if a committer).

For details of authoring/updating the documents and website, see the developers' guide. To help write the Asciidoc text itself, we provide some Asciidoc templates.

Publishing is performed by copying the generated HTML to a different git repository (isis-site). This is synced by ASF infrastructure over to isis.apache.org. This can only be done by Apache Isis committers, and is discussed below.

10.1. One-time setup

The generated site is published by copying into the `content/` directory of the isis-site git repo. You therefore need to check this out this repo.

The procedure assumes that two git repos (for isis itself and for isis-site) are checked out into the same parent directory, eg:

```
/APACHE/
    isis/                       # checkout of isis.git
        adocs/
            documentation/
                README.adoc     # this file you are reading right now
                ...
    isis-site/                  # checkout of isis-site.git
        content/                # the published website
```

If this isn't the case, then it is possible to override the relative directory by passing in a system property to the mvn goal; see below.

You also need to know that ASF's publishing script work from the 'asf-site' branch, NOT from the 'master' branch. Therefore, in the `isis.git` repo site:

```
git checkout asf-site
```

10.2. Publishing (full build)

Back in the `adocs/documentation` directory of the main `isis-git.repo`, to copy the generated documents to the `isis-site.git` repo, run:

```
mvn clean package
```

This deletes the entire content of `contents`, and replaces with the content under `target/site`. It also fixes up line endings, standardizing on unix-style LFs.

If you have checked out the `isis-site.git` repo into some other directory (relative to `isis.site.git), then this can be overridden by specifying `-Disis-site.dir=⋯` when calling `mvn`.

To copy and to also commit the generated documents to the `isis-site.git` repo , run:

```
sh publish.sh "ISIS-nnnn: a custom commit message"
```

Behind the scenes this just calls `mvn clean install -Dmessage=⋯`.

It's also possible to omit the message, in which case `publish.sh` will reuse the most recent commit message from the main `isis.git` repo.

Pushing the commits (in the `isis-site.git` directory, of course) will publishing the changes:

```
git push
```

Double check at isis.apache.org.

10.3. Publishing (partial build)

If none of the guides have been changed, and if you have run the full rebuild recently, then you can skip the generation of PDFs using:

```
mvn install -Dskip.pdf -D"ISIS-nnnn: a custom commit message"
```

The `clean` goal **must not** be included though (else all the guides will disappear from the site content).

Chapter 11. Key Generation

In order that a contributor can make a release it is necessary for them to have generated a key and had that key recognized by other members of the Apache Software Foundation.

For further background information on this topic, see the release signing page and the openpgp page on the Apache wiki.

11.1. Install and Configure gpg

Download and install GnuPG (gpg), version 1.4.10 or higher.

Then, edit `~/.gnupg/gpg.conf` (on Windows, the file to edit is `C:\Users\xxx\AppData\Roaming\gnupg\gpg.conf`) so that the default is to generate a strong key:

```
personal-digest-preferences SHA512
cert-digest-algo SHA512
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2
ZIP Uncompressed
```

11.2. Key Generation

The Apache Software Foundation requires that keys are signed with a key (or subkey) based on RSA 4096 bits. To do this:

```
$ gpg --gen-key
gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
```

Specify RSA key:

```
Your selection? 1

RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

Specify key length as 4096 bits:

```
What keysize do you want? (2048) 4096
Requested keysize is 4096 bits

Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
```

Specify key as non-expiring:

```
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name:
```

Enter your name, email and comment:

- use your apache.org email
- the comment should be "CODE SIGNING KEY"

Real name: Xxx Xxxxxxxxx Email address: xxx@apache.org Comment: CODE SIGNING KEY You selected this USER-ID: "Xxx Xxxxxxxxx (CODE SIGNING KEY) xxx@apache.org"

Change (N)ame, ©omment, (E)mail or (O)kay/(Q)uit? O

You need a Passphrase to protect your secret key. Enter passphrase:

Provide a passphrase to secure your key.

```
Enter passphrase:
Repeat passphrase:
```

GPG will goes on to generate your key:

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
...+++++
.........................+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
....+++++
...+++++
gpg: key nnnnnnnn marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   4096R/nnnnnnnn yyyy-mm-dd
      Key fingerprint = xxxx xxxx xxxx xxxx xxxx  xxxx xxxx xxxx xxxx xxxx
uid                  Xxx Xxxxxx <xxx@apache.org>
sub   4096R/kkkkkkkk yyyy-mm-dd
```

The public key with id nnnnnnnn should now be stored in `~/.gnupg/pubring.pgp` (on Windows 7, this is in `c:/Users/xxx/AppData/Roaming/gnupg/pubring.pgp`).

To confirm the key has been generated, use:

```
$ gpg --list-keys --fingerprint
```

The key Id is the one true way to identify the key, and is also the last 8 digits of the fingerprint. The corresponding secret key for id `nnnnnnnn` is stored in `~/.gnupg/secring.pgp` (on Windows 7, this is in `c:/Users/xxx/AppData/Roaming/gnupg/secring.pgp`).

It's also worth confirming the key has the correct preference of algorithms (reflecting the initial configuration we did earlier). For this, enter the gpg shell for your new key:

```
$ gpg --edit-key nnnnnnnnn
>gpg
```

where `nnnnnnnn` is your key id. Now, use the 'showpref' subcommand to list details:

```
gpg> showpref
[ultimate] (1). Xxx Xxxxxxxx (CODE SIGNING KEY) <xxx@apache.org>
     Cipher: AES256, AES192, AES, CAST5, 3DES
     Digest: SHA512, SHA384, SHA256, SHA224, SHA1
     Compression: ZLIB, BZIP2, ZIP, Uncompressed
     Features: MDC, Keyserver no-modify

gpg>
```

The Digest line should list SHA-512 first and SHA-1 last. If it doesn't, use the "setpref" command:

```
setpref SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP
Uncompressed
```

Finally, remember to take a backup of your key and the keyring (ie, backup the `.gnupg` directory and its contents).

11.3. Subkey Generation

It's recommended to use a subkey with an expiry date to sign releases, rather than your main, non-expiring key. If a subkey is present, then gpg will use it for signing in preference to the main key.

After (binary) release artifacts are created, they are deployed to the ASF's Nexus staging repository. However, Nexus seems unable to retrieve a subkey from the public key server. Until we find a fix/workaround for this, all releases should be signed just with a regular non-expiring main key.

To create a subkey Enter the gpg shell using (the identifier of) your main key:

```
gpg --edit-key xxxxxxxx
gpg>
```

Type 'addkey' to create a subkey, and enter your passphrase for the main key:

```
gpg> addkey
Key is protected.
[enter your secret passphrase]

You need a passphrase to unlock the secret key for
user: "Dan Haywood (CODE SIGNING KEY) <danhaywood@apache.org>"
4096-bit RSA key, ID xxxxxxxx, created 2011-02-01

Please select what kind of key you want:
   (3) DSA (sign only)
   (4) RSA (sign only)
   (5) Elgamal (encrypt only)
   (6) RSA (encrypt only)
Your selection?
```

Select (6) to choose an RSA key for encryption:

It would seem that Nexus repository manager does not recognize RSA subkeys with an 'S'ign usage; see this discussion on a mailing list and this issue on Sonatype's JIRA

```
Your selection? 6

RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096

Requested keysize is 4096 bits

Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for?
```

Specify that the key is valid for 1 year:

```
Key is valid for? (0) 1y

Key expires at yy/MM/dd hh:mm:ss
Is this correct? (y/N) y
Really create? (y/N) y
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
...+++++
.+++++

pub  4096R/xxxxxxxx  created: yyyy-mm-dd  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
sub  4096R/xxxxxxxx  created: yyyy-mm-dd  expires: yyYY-mm-dd  usage: E
[ultimate] (1). Dan Haywood (CODE SIGNING KEY) <danhaywood@apache.org>

gpg>
```

Quit the gpg shell; you now have a subkey.

11.4. Generate a Revocation Certificate

It's good practice to generate a number of revocation certificates so that the key can be revoked if it happens to be compromised. See the gpg page on the Apache wiki for more background on this topic.

First, generate a "no reason specified" key:

```
$ gpg --output revoke-nnnnnnnn-0.asc --armor --gen-revoke nnnnnnnn

sec  4096R/nnnnnnnn yyyy-mm-dd Xxx Xxxxxxx (CODE SIGNING KEY) <xx@apache.org>
Create a revocation certificate for this key? (y/N) Y

Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision?
```

Select 0.

```
Your decision? 0

Enter an optional description; end it with an empty line:
```

Provide a description:

```
> Generic certificate to revoke key, generated at time of key creation.
>
Reason for revocation: No reason specified
Generic certificate to revoke key, generated at time of key creation.
Is this okay? (y/N)
```

Confirm this is ok.

```
Is this okay? y

You need a passphrase to unlock the secret key for
user: "Xxx Xxxxxxx (CODE SIGNING KEY) <xxx@apache.org>"
4096-bit RSA key, ID nnnnnnnn, created yyyy-mm-dd

Enter passphrase:
</pre>

Enter a passphrase:

<pre>
Enter passphrase:
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

The file `revoke-nnnnnnnn-0.asc` should be created: Then, backup this file.

Now repeat the process to create two further revocation certificates:

```
gpg --output revoke-nnnnnnnn-1.asc --armor --gen-revoke nnnnnnnn
```

Specify reason as "1 = Key has been compromised"

and:

```
gpg --output revoke-nnnnnnnn-3.asc --armor --gen-revoke nnnnnnnn
```

Specify reason as "3 = Key is no longer used"

Backup these files also.

11.5. Publish Key

It is also necessary to publish your key. There are several places where this should be done. In most cases, you'll need the "armored" " (ie ASCII) representation of your key. This can be generated using:

```
$ gpg --armor --export nnnnnnnn > nnnnnnnn.asc
```

where nnnnnnnn is the id of your public key.

You'll also need the fingerprint of your key. This can be generated using:

```
$ gpg --fingerprint nnnnnnnn
```

The output from this command includes a line beginning "Key fingerprint", followed by a (space delimited) 40 character hexadecimal fingerprint. The last 8 characters should be the same as the key id (nnnnnnnn).

11.5.1. Publish to a public key server

To a publish your key to a public key server (eg the MIT key server hosted at http://pgp.mit.edu), use the procedure below. Public key servers synchronize with each other, so publishing to one key server should be sufficient. For background reading on this, see the release signing page on the Apache wiki, and the gpg key page on the Maven wiki.

To send the key up to the key server:

```
$ gpg --send-keys --keyserver pgp.mit.edu nnnnnnnn
```

where nnnnnnnn is the key Id.

Alternatively, you can browse to the MIT key server and paste in the armored representation of your key.

Confirm the key has been added by browsing to submitting the following URL:

http://pgp.mit.edu:11371/pks/lookup?search=0xnnnnnnnnn&op=vindex

again, where nnnnnnnn is the key Id.

11.5.2. Publish to your Apache home directory

The armored representation of your public key should be uploaded to your home directory on `people.apache.org`, and renamed as `.pgpkey`. Make sure this is readable by all.

11.5.3. Publish to your Apache HTML home directory

The armored representation of your public key should be uploaded to your `public_html` home directory on `people.apache.org`, named `nnnnnnnn.asc`. Make sure this is readable by all.

Check the file is accessible by browsing to:

`http://people.apache.org/~xxxxxxxx/nnnnnnnn.asc`

where

- `xxxxxxxx` is your apache LDAP user name
- `nnnnnnnn` is your public key id.

11.5.4. FOAF

First, check out the committers/info directory:

Go to Apache FOAF-a-matic web page to generate the FOAF file text (we copy this text out in a minute):

- enter ASF LDAP user name
- enter First name, Last name
- for PGP key fingerprints, add Key
- paste in the key id
- paste in the fingerprint
- press "Create"

In the box below, you should have a FOAF file, something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
      xmlns:pm="http://www.web-semantics.org/ns/pm#"
      xmlns:wot="http://xmlns.com/wot/0.1/"
      xmlns:rss="http://purl.org/rss/1.0/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:ical="http://www.w3.org/2002/12/cal/ical#"
      xmlns:doap="http://usefulinc.com/ns/doap#">
  <foaf:Person rdf:ID="danhaywood">
    <foaf:name>Xxx Xxxxxxxx</foaf:name>
    <foaf:givenname>Xxx</foaf:givenname>
    <foaf:family_name>Xxxxxxxx</foaf:family_name>
    <wot:hasKey>
      <wot:PubKey>
        <wot:fingerprint>nnnn nnnn nnnn nnnn nnnn  nnnn nnnn nnnn nnnn
nnnn</wot:fingerprint>
        <wot:hex_id>nnnnnnnn</wot:hex_id>
      </wot:PubKey>
    </wot:hasKey>
  </foaf:Person>
</rdf:RDF>
```

(If you are creating the FOAF file for the first time, you may want to add additional details).

From this, copy out the `wot:key`, and paste into your FDF file in `committers/info`:

```
<wot:hasKey>
  <wot:PubKey>
    <wot:fingerprint>nnnn nnnn nnnn nnnn nnnn  nnnn nnnn nnnn nnnn
nnnn</wot:fingerprint>
    <wot:hex_id>nnnnnnnn</wot:hex_id>
  </wot:PubKey>
</wot:hasKey>
```

Then, manually add in a `<wot:pubkeyAddress>` element within `<wot:PubKey>`:

```
<wot:hasKey>
  <wot:PubKey>
    <wot:fingerprint>nnnn nnnn nnnn nnnn nnnn  nnnn nnnn nnnn nnnn
nnnn</wot:fingerprint>
    <wot:hex_id>nnnnnnnn</wot:hex_id>
    <wot:pubkeyAddress rdf:resource="http://people.apache.org/~username/nnnnnnnn.asc/
">
  </wot:PubKey>
</wot:hasKey>
```

ie, referencing your publically exported public key

Finally, commit your changes.

11.5.5. Save to KEYS

The armored representation of the public key should be saved to Apache Isis' KEYS file, http://www.apache.org/dist/isis/KEYS (that is, in the ASF distribution directory for Apache Isis).

First, in a new directory, checkout this file:

```
svn -N co https://svn.apache.org/repos/asf/isis/ .
```

This should bring down the KEYS file.

Then, export your signature and armored representation.

```
gpg --list-sigs nnnnnnnn >>KEYS
gpg --armor --export nnnnnnnn >>KEYS
```

Then commit.

11.5.6. id.apache.org

Log onto id.apache.org and ensure that the finger print of your public key is correct.

11.6. Attend Key Signing Party (Apache web of trust)

It is strongly advised that the contributor attend a key signing party at an Apache event, in order that other Apache committers/members can in person verify their identity against the key. The process for this is described here and here.

11.7. Update Maven Settings file (`~/.m2/settings.xml`)

The Maven release plugin will automatically sign the release, however it is necessary to update the `~/.m2/settings.xml` file with your GPG acronym passphrase in order that it can use your secret key. This is defined under a profile so that it is activated only when we perform a release (as defined by `[org,apache:apache]` parent POM.

Therefore, make the following edits:

```
<settings>
  ...
  <profiles>
    <profile>
      <id>apache-release</id>
      <properties>
        <gpg.passphrase>xxx xxx xxx xxx xxx xxx xxx</gpg.passphrase>
      </properties>
    </profile>
  </profiles>
</settings>
```

In addition, to allow the release plugin to tag SVN changes, you must either add in your LDAP username/password or configure `.ssh`:

```
<settings>
  ...
  <servers>
    ...
    <server>
      <id>apache.releases.https</id>
      <username>xxxx</username>
      <password>xxxx</password>
    </server>
  </servers>
</settings>
```

Chapter 12. Appendix: Release Prereqs

This section (appendix) describes the prerequisites for the release process.

12.1. Configure toolchains plugin

Apache Isis releases are built using Java 7, enforced using the maven toolchains plugin. Ensure that Java 7 is installed and the toolchains plugin is configured, as described in the contributors' guide.

12.2. Public/private key

The most important configuration you require is to set up public/private key pair. This is used by the `maven-release-plugin` to sign the code artifacts. See the page on key generation for more details.

In order to prepare the release, you'll (need to) have a `~/.gnupg` directory with the relevant files (`gpg.conf`, `pubring.gpg`, `secring.gpg` etc), and have `gpg` on your operating system PATH.

If on Windows, the equivalent directory is `c:\users\xxx\appdata\roaming\gnupg`. For `gpg`, use either cygwin.com or gpg4win.org.

Note also that the mSysGit version of `gpg.exe` (as provided by GitHub's bash client) is not compatible with that provided by cygwin; move it to one side and check that `gpg.exe` being used is that from gpg4win.

If you use Atlassian's SourceTree, this also bundles a version of `gpg.exe` that is not compatible (in `C:\Users\xxx\AppData\Local\Atlassian\SourceTree\git_local\usr\bin`); again, move it to one side and instead use the one provided by gpg4win.

12.3. Maven `settings.xml`

During the release process the `maven-deploy-plugin` uploads the generated artifacts to a staging repo on the Apache repository manager. This requires your Apache LDAP credentials to be specified in your `~/.m2/settings.xml` file:

```
<settings>
  <servers>
    <server>
      <id>apache.releases.https</id>
      <username>xxxxxxx</username>
      <password>yyyyyyy</password>
    </server>
    ...
  </servers>
  ...
</settings>
```

where xxxxxxx and yyyyyyy are your Apache LDAP username and password. For more information, see these ASF docs.

It is also possible to configure to use .ssh secure keys, and thereby avoid hardcoding your Apache LDAP password into your .m2/settings.xml file. A description of how to do this can be found, for example, here.

Also, set up keyphrase for gpg; this avoids being prompted during release:

```
<profiles>
  <profile>
    <id>gpg</id>
    <properties>
      <gpg.executable>gpg2</gpg.executable>
      <gpg.passphrase>this is not really my passphrase</gpg.passphrase>
    </properties>
  </profile>
  ...
</profiles>

<activeProfiles>
  <activeProfile>gpg</activeProfile>
  ...
</activeProfiles>
```

Chapter 13. Policies

This chapter pulls together various policy documents relating to the development of Apache Isis'.

13.1. Versioning Policy

13.1.1. Semantic Versioning

Starting from v1.0.0, Apache Isis has adopted semantic versioning for its versioning policy.

Version numbers are in the form `x.y.z`:

- x is bumped up whenever there a breaking API change
- y is bumped up whenever there is a new feature that does not break API
- z is bumped up for minor bug fixes.

This scheme would be adopted for both core and components.

13.1.2. Version ranges

Version ranges may not be used. If necessary, end-users can use `<dependencyManagement` elements to have combine components built against different versions of core.

That said, this can introduce instability and so generally we recommend that end-users configure the `maven-enforcer-plugin` and its DependencyConvergence rule. This will help avoid "jar hell" (components having conflicting dependencies of core).

If there is a conflict, we would ask that end-users engage with Apache Isis committers to have an updated version of the component(s) pushed out.

13.2. Git Policy

These notes recommend how contributors should work with git. To understand these notes, the only real concepts that you need to grok are:

- git commits form an acyclic graph, with each commit pointing to its parent commit (or commits, if a merge)
- a branch is merely a pointer to one of these commits; git calls the main branch `master`
- git commits happen in two steps: first they are added to the index (also called the staging area), then they are committed.

For more background reading, see:

- Pro Git book (free in electronic form)
- Git community book
- git reset demystified - differentiating the working directory vs index/staging area

And, of course, there is loads of good advice on stackoverflow.com

13.2.1. Workflow

There are many ways of using Git, but the Apache Isis committers have adopted the following workflow:

- create a topic branch for a feature

```
git checkout -b ISIS-999
```

- periodically, push the branch to origin (for safekeeping):

```
git push origin ISIS-999
```

- `rebase` the topic branch periodically on master.

  How often you do this will depend on whether you are collaborating with others on the feature. You need to ensure that your co-worker has no outstanding work before you do this; otherwise it'll create merge conflict hell for them:

```
git checkout master
git pull
git checkout ISIS-999
git rebase master
git push origin ISIS-999 --force
```

- when feature is complete, rebase once more (as above), then switch to master and perform a `merge --no-ff`:

```
git checkout master
git merge --no-ff ISIS-999
```

- finally, remove the branch

```
git branch -d ISIS-999
git push origin --delete ISIS-999
```

This way of working gives us the full history on the branch as to what the thought processes were for the feature, but only a single commit on to `master` to see the ultimate impact of the changes (acting a bit like a summary).

13.2.2. Commit message

The minimum we expect in a commit messages is:

```
ISIS-nnn: brief summary here

- optionally, longer details
- should be written here
- in bullet points
```

where `ISIS-nnn` is a ticket raised in our JIRA issue tracker.

For non-committers we typically expect more detail again; see the contributing page for the longer format recommended for contributors to use.

Chapter 14. Appendix: PMC

Every ASF project has a Project Management Committee, or PMC. This committee is ultimately responsible for the long-term management of the framework. More information about PMCs can be found here

In Apache Isis, every committer is a member of the PMC.

This page contains some general notes on maintenance activities required by PMC members.

14.1. Prereqs

To complete the procedures documented here, you'll need `ssh` access to `minotaur.apache.org`. For example:

```
ssh danhaywood@minotaur.apache.org
```

When prompted, provide passphrase for private key.

To set up ssh in the first place, take a look at this section in the ASF new committers' guide looks relevant.

14.2. New Committer/PMC member

Currently we don't distinguish between committers and PMC members: every committer is automatically invited to also be a member of the PMC.

Further notes on the steps for new PMC members can be found here.

14.2.1. Start a vote thread on private@

First, send a [VOTE] email to `private@isis.apache.org`, making the case for the new committer to join.

For example:

```
[VOTE] Joe Bloggs as a committer and PMC Member
①
```

① update as required

with body:

```
I'd like to nominate Joe Bloggs as a committer and also a PMC member for Apache Isis.
①

Joe this ...
①

Joe that ...
①

Overall, I think Joe would be a great addition to Isis' committers and PMC; ...

Voting ends in one week's time, i.e. midnight UTC on YYYY-MM-DD
①
http://www.timeanddate.com/countdown/to?year=YYYY&month=MM&day=DD
```

① update as required

By convention, we run these votes for 7 days (a minimum of 3 days is required by ASF).

14.2.2. Close the vote, announce results

After the 7 days has expired, close the vote thread.

```
I'm now closing this vote.  Result will be posted on a new thread.
```

In a separate thread, announce the results. For example:

```
[RESULT] [VOTE] Joe Bloggs as a committer and PMC Member
①
```

① update as required

with body:

```
The vote has now closed. The results are:

+3 (PMC)
①

consisting of:
* Mary (PMC)
①
* Mungo (PMC)
* Midge (PMC)

There were no other votes cast.

The vote is ***successful***

I'll announce to users@ and dev@ by separate thread.
```

① update as required

14.2.3. ICLA, obtain new account

If required (that is, if the committer is not already a committer for a different ASF project), then ask them to complete an ICLA. As a result of this, they should also get an @apache.org user name.

More info can be found in the ASF new committers guide.

14.2.4. Update the `isis` unix group

All committers must be added to the `isis` UNIX group. This will give them commit access to the Apache Isis git repo.

To do this, log onto `minotaur.apache.org`, then eg:

```
list_unix_group.pl isis
```

and

```
modify_unix_group.pl isis --add joebloggs
①
```

① update as required

14.2.5. Update the LDAP committee (if a PMC member)

(Assuming that the new committer is a PMC member), also add them as to the PMC committee. This takes two steps:

- first, log onto `minotaur.apache.org` and update LDAP committee:

```
list_committee.pl isis
```

and

```
modify_committee.pl isis --add joebloggs
①
```

① update as required

- second, update `committee-info.txt`

  This is held in SVN, under https://svn.apache.org/repos/private; the file resides at `committers/board/committee-info.txt`

The new committer does not officially become a member of the PMC until the ASF records have been updated.

14.2.6. Notify the ASF board and private mailing list

Send a `[NOTICE]` email to `board@apache.org` and also to `private@isis.apache.org`

For example:

```
[NOTICE] Joe Bloggs to join Apache Isis as committer and member of the PMC
①
```

① update as required

with body:

```
The Apache Isis PMC has voted Joe Bloggs as a committer and also member of the PMC.
①

Voting thread:
https://mail-search.apache.org/members/private-arch/isis-private/xxx
①

Results announcement:
https://mail-search.apache.org/members/private-arch/isis-private/xxx
①

The committee-info.txt file has been updated, as have the LDAP groups
(modify_unix_group.pl,  modify_committee.pl).
```

① update as required

update the private threads above

14.2.7. Update project metadata

Update the `STATUS` file (in the root directory of the Apache Isis git repo) with the new committer details.

14.2.8. Announce to the world

Send an `[ANNOUNCE]` email TO `users@isis.apache.org` and to `dev@isis.apache.org`.

For example:

```
[ANNOUNCE] New committer - Joe Bloggs
①
```

① update as required

with body:

```
I'm delighted to announce that Joe Bloggs has been voted in as a committer on Isis,
and also as a member of the Isis PMC.  The first gives Joe the right to commit changes
directly to Isis' codebase, the second gives him the right to be involved in future
votes.

Joe this ...
①

Joe that ...

I'm looking forward to working with Joe in the future; another great addition to Isis'
committers.  So please join me in welcoming him to our happy band!

Dan Haywood
Apache Isis PMC Chair
```

① update as required

Also, write a similar blog post at blogs.apache.org/isis

14.3. Removing a committer

these notes are only draft, will need fleshing out.

- remove from `isis` UNIX group:

```
modify_unix_group.pl isis --remove joebloggs
```

- remove from `isis` committee:

```
modify_committee.pl isis --add joebloggs
modify_committee.pl isis --remove joebloggs
```

- remove from `committee-info.txt`
- send a [NOTICE] email to board@ and private@