Creating Magnolia Debian Packages

Debian Packaging submodule for Magnolia projects

The debian packaging git submodule can be found in our git repository. This wiki page is a copy of the README.md file in that repository as at 26 June 2015.

How to use this module

This git repository is intended to be used as a git submodule.

To add it to your repo, follow these five steps.

0. Ensure you have the Debian package tools installed

```
sudo aptitude install build-essential fakeroot devscripts dpkg-dev git-buildpackage
```

1. Graft in this repo as a submodule

From your package's top-level directory, run the following:

```
git submodule add http://git.magnolia-cms.com/build/debian
```

Then commit and push this change to your repo as usual. When cloning, you should then be sure to use the `--recursive` option to fetch the full tree. Doing this will not track updates; to update this submodule, use `git submodule update --remote debian` then commit and push this change as usual.

For more information regarding submodules, please refer to the git submodule chapter of the git book.

2. Create the `settings.mk` file

Create a file called `settings.mk` in the top-level directory like the following example for ce-bundle:

```
WEBAPPNAME := ce
WEBAPPDESC := Community Edition
GIT := ce-bundle.pub
HOMEPAGE := https://wiki.magnolia-cms.com/display/WIKI/Community+Wiki
AUTHORINSTNAME := ce-auth
AUTHORINSTNUM := 0
PUBLICINSTNAME := ce-pub1
PUBLICINSTNUM := 1
WEBAPPWAR := magnolia-bundled-webapp
PACKAGER := Pete Ryland <pete.ryland@magnolia-cms.com>
```

Don't forget to edit this with the right details for your package.

3. Clean the repo

```
fakeroot debian/rules clean
```

This will also initialise the changelog file which every debian package must have.

4. Create your packages

```
fakeroot debian/rules binary
```

The following commands may be helpful when updating the changelog.

Get the version from the pom.xml file:

```
VERSION="$(python -c "import xml.etree.ElementTree as ET; print(ET.parse(open('pom.xml')).
getroot().find('{http://maven.apache.org/POM/4.0.0}version').text.replace('-
SNAPSHOT','~SNAPSHOT'))")"

echo Pom Version: "$VERSION"
```

Get the version from the changelog:

```
DEBVERSION="$(dpkg-parsechangelog -SVersion | sed 's/~/_/g')"

echo Deb Version: "$DEBVERSION"
```

Update the changelog after a pom version bump, assuming you have a tag as below:

```
VERSION="$(python -c "import xml.etree.ElementTree as ET; print(ET.parse(open('pom.xml')).
getroot().find('{http://maven.apache.org/POM/4.0.0}version').text.replace('-
SNAPSHOT','~SNAPSHOT'))")"

DEBVERSION="$(dpkg-parsechangelog -SVersion | sed 's/~/_/g')"

git-dch --new-version="$(VERSION)-1" --since="$DEBVERSION"
```

Update the changelog without bumping the pom version, assuming you have a tag as below:

```
DEBVERSION="$(dpkg-parsechangelog -SVersion | sed 's/~/_/g')"

git-dch --since="$DEBVERSION"
```

Finalise changelog before creating packages:

```
dch -r
```

Tag against new deb version:

```
DEBVERSION="$(dpkg-parsechangelog -SVersion | sed 's/~/_/g')"

git tag "$DEBVERSION"
```

Don't forget to push tags after committing:

```
git push --tags
```

Jenkins

I use the following script to build and push packages from jenkins, which is called with the `main` parameter for the *release* job which triggers only when the changelog is pushed to git.

```
```

```
#!/bin/bash

# This should be called by Jenkins in a post-build action after a successful maven build

repo=${1:-staging}

git checkout changelog

PKGNAME="$(dpkg-parsechangelog -SSource)"

if [ "$repo" = main ]; then
  PKGVER="$(dpkg-parsechangelog -SVersion)"
else
  PKGVER="$(dpkg-parsechangelog -SVersion)"."$BUILD_NUMBER"

  # Update changelog, appending the build number, and marking as unreleased
  dch --no-conf -c changelog -v "$PKGVER" "Jenkins build $BUILD_NUMBER: $BUILD_URL"
fi

# Assume maven has built the war correctly
touch build-stamp

# Assemble the packages
fakeroot debian/rules binary

cd ..

# Copy over to the repo server
scp "$PKGNAME"-{common,author,public}_"$PKGVER"_all.deb reposerver:

# Place the packages in the repo
ssh reposerver ./update_debs "$repo" "$PKGNAME"-{common,author,public}_"$PKGVER"_all.deb

# Remove the temporary copies
rm -f "$PKGNAME"-{common,author,public}_"$PKGVER"_all.deb
```

where update_debs consists of the following little script for use with reprepro and an appropriate sudoers rule:

```
#!/bin/bash

LOCKFILE=/home/jenkins/.update_debs.pid
trap "rm -f '$LOCKFILE'" EXIT
(
flock 3
repo=${1:-staging}
shift

for deb in "$@"; do
  if [ -r "$deb" ]; then
    for distro in jessie; do
      sudo reprepro -C "$repo" -Vb /var/www/debian-repo/ includedeb "$distro" "$deb"
    done
    rm -f "$deb"
  fi
done
) 3> "$LOCKFILE"
```