Setting version strings for release and development builds

BV

[2014-09-15 Mon 11:26]

1 The problem

It is typical and usually desirable to maintain one or more files in a source repository that holds a version string that is used when building a source or binary distribution package. For python packages this is usually the `version` argument to `setup()` in `setup.py`. For Fermilab CET's UPS-encumbered CMake system it is the `ups/product_deps` file.

This version string is intended to precisely identify the state of the source code that went into making the distributed package. Knowing it, and knowing the invariant procedure for producing the package, should allow someone to go back and exactly reproduce the distributed package or reproduce its source for archaeological purposes.

It is, however, typical that any given version string persists for some period of time and can be found in a number of commits. This is largely a problem for producing "development" packages but can also exist in the production of "release" packages. For example, projects that follow the `git flow` branching model, by construction, introduce multiple commits on the `release` branch that all carry the eventual release version string. This technical degeneracy is mostly forgivable as it is understood that this version string should only apply to the eventual tagged commit on the `master` branch.

However, this `git flow` branching model fails to address what becomes of this version string after a `release` branch is merged back to `develop`. It also does not address modifying this version string a the point of starting or finishing a `feature` branch. If this omission is carried through to implementing the model one ends up carrying the most recent release version into a `feature` branch and persisting it on `develop`. Any packages built from these branches will appear to be releases!

2 Desired Solution Features

What is wanted is a way to produce a package what clearly but succinctly identifies the source code that was used in its production. This method should work regardless of the state of the source repository. Specifically it needs to support the case where a packages is created from a release, feature, "hotfix" or maintenance branch (to use `git flow` terms). It should also work if a package is produced from any commit on these branches and ideally indicate if any uncommitted modifications existed in the source files.

3 Git-based solution

The obvious implications of the above is that the version information must ultimately come from the repository technology as nothing else supplies the needed level of bookkeeping.

3.1 Using `git describe` to set the version string

In the case of `git` part of the needed functionality is provided using:

```
git describe --dirty
```

This will return one of three possible values:

`<tag>` the annotated tag if it exists on the current commit and the working directory is clean

`<tag>-<N>-g<hash>` the <tag> is the most recent annotated tag, `<N>` counts the number of commits made since that tag and an abbreviation of its git hash of the current commit, again assuming the working directory is clean

`<tag>-<N>-g<hash>-dirty` as above but if any tracked files have uncommitted changes.

This is a suitable source of information for a version string. In the case of a release the version is simply `<tag>`. For any other point in the repository the version string will gain an simple method to identify and locate the state from a known release. This identification is made automatically. As a bonus, if a package is prepared from a "dirty" working directory there is some indication of this (although, no way to identify **what** was modified).

3.2 Applying a version string

While this source of version information is ideal, applying it necessarily requires some procedure that works outside of the repository. If one runs `git describe`, saves the output to a file under the control of the repository and commits then that commit is not what the version string claims, while the commit to which it applied remains holding whatever the version string was prior. This sets up a chicken-and-egg problem. The only solution is to apply the version string in some manner not tied to the repository. It is noted that this problem is universal to any mechanism that stores a version string in a file under the control of the repository.

The solution is to incorporate the consumption of this version string into the package build procedures or, at the very least, into the build procedures for non-release packages. Generally speaking the build procedure needs to incorporate these steps:

1. run `git describe --dirty`
2. modify any files which carry the version string
3. build a distribution package
4. restore the source directory to it's prior state

A few notes:

- The use of bumpversion may help with step (2), although it is recommended to that the version string is written generically in all files, eg like `@VERSION@` and replaced only at this step during package build time
- If there is no concern for any modifications that caused the working directory to be dirty, (4) can simply be `git reset`