Git Tricks

BV

[2014-09-05 Fri 10:16]

1 Development Version Control

Clone a fresh copy and initialize for `git flow`

```
$ git clone git@github.com:brettviren/PACKAGE.git PACKAGE-for-release
$ cd PACKAGE-for-release/
$ git flow init -d
```

It's now on the `develop` branch. What to do next depends. There are two choices:

1. work on the `develop` branch if some limited work is planned and if the clone was not from a central, shared repository
2. start a feature branch for extended development or if unfinished development needs to be pushed back to a central, shared repository

1.1 Work in `develop`

Check version in `setup.py`:

```
$ grep version setup.py
      version = '0.5.0-dev',
```

If it doesn't already end in "`-dev`" edit the file to make it so and commit. Now, hack away and when done:

```
$ git commit -a -m "My little development"
$ git push
```

1.2 Work in a feature branch

Use `git flow` to start a feature branch:

```
$ git flow feature start some-feature-name
```

Edit the `version` in `setup.py` so that it is marked by your feature name

```
$ grep version setup.py
      version = '0.5.0-some-feature-name',
```

Commit this and hack away. Commit and push intermediate work with abandon, it's your branch!

In the process of finishing the branch and merging it back into develop, reset the version string.

```
$ grep version setup.py
      version = '0.5.0-dev',
$ git flow feature finish some-feature-name
$ git push
```

2 Version Control and Release Management

Following `git flow`, releases come from `develop` and give the target release tag:

```
$ git flow release start 0.6.0
```

Now edit `setup.py` to reflect new version (remove `"-dev"`) and commit and change any dependency versions that might be needed (deps may also change as part of development).:

```
$ emacs setup.py requirements.txt
$ grep version setup.py
      version = '0.6.0',
$ git commit -a -m "Set version to 0.6.0."
```

If there is any additional release munging needed, do it now. But, if `develop` was essentially ready to release already then there is nothing more to do. If there are features to merge, they should happen as part of the end game of working in a feature branch and not onto a release branch.

Now, finish up:

```
$ git flow release finish 0.6.0
```

Now is a good time to upload to PyPI or otherwise prepare any release archives. This can be done at any later point by checking out the release tag, but might as well do it now.

```
$ python setup.py sdist upload
```

WAIT one last thing. Move **off** the release version so that any ad-hoc packaging that may be created in the future from the `develop` branch does not pollute the ecosystem with a false release version string.

```
$ grep version setup.py
      version = '0.6.0-dev',
$ git commit -a -m "Bump off release and on to dev version."
```

NOW, you can commit everything:

```
$ git push
$ git push --tags
```

3 Obsolete

This section holds an obsolete workflow but kept around as it has some useful workflow.

I want to automate management of versions and their releases. For that I use git-flow, bumpversion and for Python packages some custom tweak to set the version. Here is me making a release of lbne-build. It starts with me in a clean repository sitting on the `develop` branch.

```
$ git flow release start 0.3.3
Branches 'develop' and 'origin/develop' have diverged.
And local branch 'develop' is ahead of 'origin/develop'.
Switched to a new branch 'release/0.3.3'

Summary of actions:
- A new branch 'release/0.3.3' was created, based on 'develop'
- You are now on branch 'release/0.3.3'

Follow-up actions:
- Bump the version number now!
```

```
- Start committing last-minute fixes in preparing your release
- When done, run:

     git flow release finish '0.3.3'
```

To "bump the version" the very nice `bumpversion` script is used. To start out it's `./.bumpversion.cfg` script is primed (or as-is from last use) like:

```
[bumpversion]
current_version = 0.3.2
tag_name = {new_version}
commit = False

[bumpversion:file:RELEASE-VERSION]
```

To "bump the version" all one has to do is run the `bumpversion` script with the version "level" of the bump (`major`, `minor` or `patch`).

```
$ cat RELEASE-VERSION
0.3.2
$ bumpversion patch
$ cat RELEASE-VERSION
0.3.3
```

It's a long walk to take to change one character but once set up it's much easier than editing files by hand and this scales well to cases that require the version to change in multiple places.

To finish the release I do:

```
$ git commit -a -m "Ready for release 0.3.3"
$ git flow release finish 0.3.3
(accept default commit messages but do add an annotation to the tag)
...
Summary of actions:
- Release branch 'release/0.3.3' has been merged into 'master'
- The release was tagged '0.3.3'
- Release tag '0.3.3' has been back-merged into 'develop'
- Release branch 'release/0.3.3' has been locally deleted
- You are now on branch 'develop'
```

```
$ git push --tags
...
To git@github.com:LBNE/lbne-build.git
 * [new tag]         0.3.3 -> 0.3.3
```

3.1 Development Version Lies

One thing I really detest is leaving a release version as-is outside of the release branch. I would like code in the `develop` branch or any feature branches to always represent itself as some version **other** than whatever release happen to have been made in the recent past. I'd settle for the version to be simply a rather anonymous "`dev`" tag or better one that represents how "close" the development is to some established release.

Thankfully `git` comes to the rescue with:

```
$ git describe --dirty
0.3.2-2-g24c4c90-dirty
```

If sitting on a commit with an annotated tag it will give that tag. Otherwise it will (by default) give the number of commits away from that tag and the start of the hash of the commit it sits on. The `--dirty` option tells it to additionally tack on that "-dirty" flag if the git working directory has modifications.

Perfect.

Unfortunately there is not yet support in `bumpversion` to somehow integrate this (which I can find). And if there were, it would likely lead to an arms race between the desire to mark a version and the fact that marking it in a file controlled by git will dirty up the repository.

So, to use this in a development context it is best to use it at the point of version string **consumption** rather than version string **production**.

3.1.1 Python packages

For Python packages this is added to `setup.py` with these few lines:

```
def git(cmd):
    import subprocess
    return subprocess.check_output("git %s"%cmd, shell=True).strip()

def current_version():
    branch = [x[2:] for x in git("branch").split('\n') if x.startswith('* ')][0]
```

```
    if branch.startswith("release/"):
return open("RELEASE-VERSION").read().strip()
    return git("describe --dirty")

setup(name = ...,
      version = current_version(),
      ...)
```

Test with:

```
$ git branch
develop
master
* release/0.3.3
$ git describe --dirty
0.3.2-2-g24c4c90-dirty
$ python setup.py --version
0.3.3
```

I can now produce a test package with what will become the release version. Of course, I have to be careful not to distribute this test package until I'm ready and the release branch is closed. The "real" release package can be produced at any later time by making a checkout using the release tag:

```
$ git checkout -b v0.3.3 0.3.3
Switched to a new branch 'v0.3.3'
$ python setup.py --version
0.3.3
$ git describe
0.3.3
$ git describe --dirty
0.3.3
```