hexrd Documentation

Release 0.3.2

Joel Bernier, Darren Dale, et. al.

November 20, 2015

Contents

Contents:

CHAPTER 1

The HEXRD Users' Guide

Contents:

1.1 Installation

CHAPTER 2

The HEXRD API

CHAPTER 3

The HEXRD Developers' Guide

Contents:

3.1 Getting Started

3.1.1 Revision Control

Hexrd uses the git revision control system and GitHub. If you are not familiar with git, github provides an interactive online tutorial. The free, online Pro Git book is another good reference.

To install git, Linux users should use their package manager. OS X users may want to install GitHub for Mac. Windows users may want to install GitHub for Windows.

The next step is to clone the repository. At the command line, run:

```
git clone -o hexrd https://github.com/praxes/hexrd.git
```

That creates the project directory. Move into that directory:

```
cd hexrd
```

3.1.2 Develop Mode

The following command will allow python to run hexrd from the project directory (assuming numpy and setuptools are already installed):

```
python setup.py develop
```

That will build the extension modules, create and install the executable, but you can still edit python files in the project directory and the changes will be immediately available without having to reinstall. However, any time the extension module sources are changed, you must run the above command again to recompile the extensions.

To stop using develop mode:

```
python setup.py develop --uninstall
```

We provide a *hexrd.debug* function that lets you interact with the code in IPython. Be advised, this should always be called using *hexrd.debug()*, rather than *from hexrd import debug* followed by *debug()*, especially when debugging while running the GUI.

3.1.3 Git Branches

Before you make any changes to the project, make sure your clone is up to date:

```
git checkout master
git pull
```

The master branch should be treated as an integration branch, changes aren't made directly to master. The next step is to make a branch in which to make modifications:

```
git checkout -b my-new-branch
```

Now, hack away, make commits.

3.1.4 Filing a Pull Request

When you think things are ready to be merged, push the branch to your personal fork of the hexrd repository at GitHub. If you don't already have a fork, visit the hexrd project page, and click the "fork" button in the upper right hand corner of the page (requires a free GitHub account). When the fork is finished, run the following at the command line:

```
git remote add your_github_id https://github.com/your_github_id/hexrd.git
```

Now you are ready to push your branch to GitHub:

```
git push your_github_id my-new-branch:my-new-branch
```

The last step is to file a pull request. Browse to your fork at GitHub, select the *branch: master* button on the left side of the screen, and select *my-new-branch*. Then select the green "compare, review, create a pull request" button, add a little information about the changes in the branch, and submit.

3.1.5 Additional Comments

If you forgot to create a new branch, and have been making changes directly in your master branch, there is potential for a more difficult merge. You generally want the master branch in your local repository to be either in sync with, or behind, the upstream master branch. You don't want commits in your local master branch that don't exist in the upstream master. If you find yourself in such a situation, its easy to resolve. If you have uncommitted changes in your working directory, first run:

```
git stash
```

Next, create a new branch, which will be a copy of master:

```
git checkout -b my-next-branch
```

Now reset the master branch so it is identical to the upstream master branch:

```
git branch -D master
git checkout -b master -t hexrd/master
```

Finally, if you stashed any uncommitted changes to create the new branch:

```
git checkout my-next-branch
git stash pop
```

Now your *my-next-branch* looks just like it would have if you had create the branch before making any changes to *master*, and *master* is in sync with https://github.com/praxes/hexrd/tree/master.

3.2 Releases

Steps for creating an official release, for example version 1.0.0.

3.2.1 Preliminary Checks

First, check that your working directory and environment is clean:

```
git checkout head
python setup.py develop --uninstall
rm -rf build
git clean -n
```

Next, test the conda package (see below), which will only succeed if the project can be built and installed. Building the conda package also runs the test suite.

3.2.2 Update Version Information

First, tag the commit that corresponds to version 1.0.0:

```
git tag v1.0.0 [head]
git push --tags
```

The *head*, or *commit hash* is not necessary if you are tagging the most recent commit in your active branch as v1.0.0. The hexrd library will report its version based on the git tag.

3.2.3 Create Conda Packages

In the *hexrd/conda.recipe* project directory, temporarily modify *git_tag* in *meta.yaml* to point to the version you want to build (don't commit this change to git):

```
#git_tag: master
git_tag: v1.0.0
```

and then run:

```
conda build conda.recipe
```

The resulting conda package can be uploaded to binstar:

```
binstar upload -u praxes /path/to/hexrd-1.0.0-np19py27_0.tar.bz2
```

Finally, change *git_tag* back to the default *master*.

3.2.4 Update the Documentation

Pushing the branch and tags to github will trigger the documentation to build automatically. Visit the hexrd documentation dashboard, the new version may need to be activated.