The Python Build Overlay Managed Bundle System

Wrangling the headaches of complexity in the software radio build process

Tim O'Shea, Research Faculty
Virginia Polytechnic Institute and University
Arlington, VA
1 Oct 2013

Motivation

1. Provide a system to help users build GNU Radio on Linux
2. Provide parallel/prefixed builds of different GNU Radio versions (test upgrades before breaking your current install, benchmark side by side, ensure OOT modules are rebuilt, etc)
3. Use native system package managers when available to provide library dependencies
4. Fall back to source builds when we don't have root, are on old systems, or otherwise don't have package repositories available
5. Provide an easy way to index and build out of tree modules on top of GNU Radio
6. Make installing end-user applications based on a number of OOT modules simple (ideally 1 click)

First Investigation

1. Hesitant to write something new
2. Looked at a lot of options
3. Bit-bake / oe-core, really made for cross compiling - bringing up a whole image
4. Gentoo Portage / e-build system - hard to de-couple from the Gentoo and use as an overlay
5. Puppet, Gems, pip - all targeted for fairly homogeneous software, i.e. only ruby packages or only python setup-tools based packages
6. Ultimately, gave up and built something that was light weight, and built for overlays, and very flexible

Quick Start / Usage:
git clone https://github.com/pybombs/pybombs.git
cd pybombs

- pybombs install gnuradio gr-ieee-80211 (installs both)
- pybombs remove gnuradio (remove gnuradio and any packages that depend on it)
- pybombs list (show current install state of all packages)
- pybombs search foo (search for a package)
- pybombs –help (show help)

PyBOMBS' rough structure:

- pybombs - top level command line executable
- app_store.py - minimal graphical front end
- recipes/*.lwr - recipe files for all known packages
- templates/*.lwt - templates for similar package styles (cmake, autoconf, python setuptools, etc)
- mod_pybombs/*.py - python implementation files

The package installation process:

1. First we make sure dependencies are satisfied
2. Then we iterate through the "satisfy order" (i.e. deb, rpm, src) checking for presence of our package based on satisfy_deb, satisfy_rpm, and source lines from our recipe
3. deb and rpm satisfiers will attempt to us sudo to install the missing package
4. source satisfier is a little bit more involved, it steps a package through the following states
5. typically GNU Radio and OOT modules are source only recipes

Source satisfaction process:

- None (package is not installed)
- Fetch (package sources downloaded and extracted)
- Configure (package has been configured and prepared for build)
- Make (package has been compiled)
- Installed (package has been installed into the prefix)

Source Satisfaction: The Fetch Stage
source: URI1, URI2, ...
Valid URI formats are

- Local files
  file://local/path/pkg.tar.gz
- Remote Files
  wget://https://github.com/foo/foo.tar.gz
- GIT Repos
  git://http://www.gnuradio.org/git/gnuradio.git
- SVN Repos
  svn://http://svn.code.sf.net/p/gnss-sdr/code/trunk/

Source is placed in pybombs/src/PKGNAME where the original recipe file was PKGNAME.lwr
for local and remote files .tar.gz, .tar.bz2, and several other variants are extracted for you automatically

Source Satisfaction: The Configure Stage

```
configure {
cmake .. -DCMAKE_BUILD_TYPE=$cmakebuildtype
-DCMAKE_INSTALL_PREFIX=$prefix $config_opt
}
```

- essentially a set of shell commands
- executed inside $configuredir which defaults to src/PKGNAME/ unless overridden in the recipe
- often inherited from a template recipe (cmake.lwt, autoconf.lwt, etc)

Source Satisfaction: The Make Stage

```
make {
make -j4
}
```

- Often the simplest but longest stage
- executed inside $makedir which defaults to src/PKGNAME/ unless overridden in the recipe

Source Satisfaction: The Install Stage

```
install {
make install
}
```

- Typically executes an install target for the package's build system
- Can conduct its own copying in the case of a deficient package make system
- executed inside $installdir which defaults to src/PKGNAME/ unless overridden in the recipe

A Recipe for GNU Radio

depends: make mcpp boost fftw cppunit swig gsl uhd git python cheetah wxpython numpy lxml pygtk pycairo cmake pyqt4 pyqwt5 gcc ice
category: common
source: git://http://www.gnuradio.org/git/gnuradio.git
gitbranch: master
var config_opt = " -DENABLE_GR_FCD=ON -DENABLE_DOXYGEN=OFF -DENABLE_GR_LOG=ON -DENABLE_PERFORMANCE_COUNTERS=ON -DCMAKE_CXX_FLAGS=-fpermissive "
configuredir: build
makedir: build
installdir: build
depends: cmake
configure {
cmake .. -DCMAKE_BUILD_TYPE=$cmakebuildtype -DCMAKE_INSTALL_PREFIX=$prefix $config_opt
}
make {
make -j4
}
install {
make install
}
uninstall {
make uninstall

}

What the GNU Radio recipe really looks like

depends: make mcpp boost fftw cppunit swig gsl uhd git python cheetah wxpython numpy lxml pygtk pycairo cmake pyqt4 pyqwt5 gcc ice

category: common

source: git://http://www.gnuradio.org/git/gnuradio.git

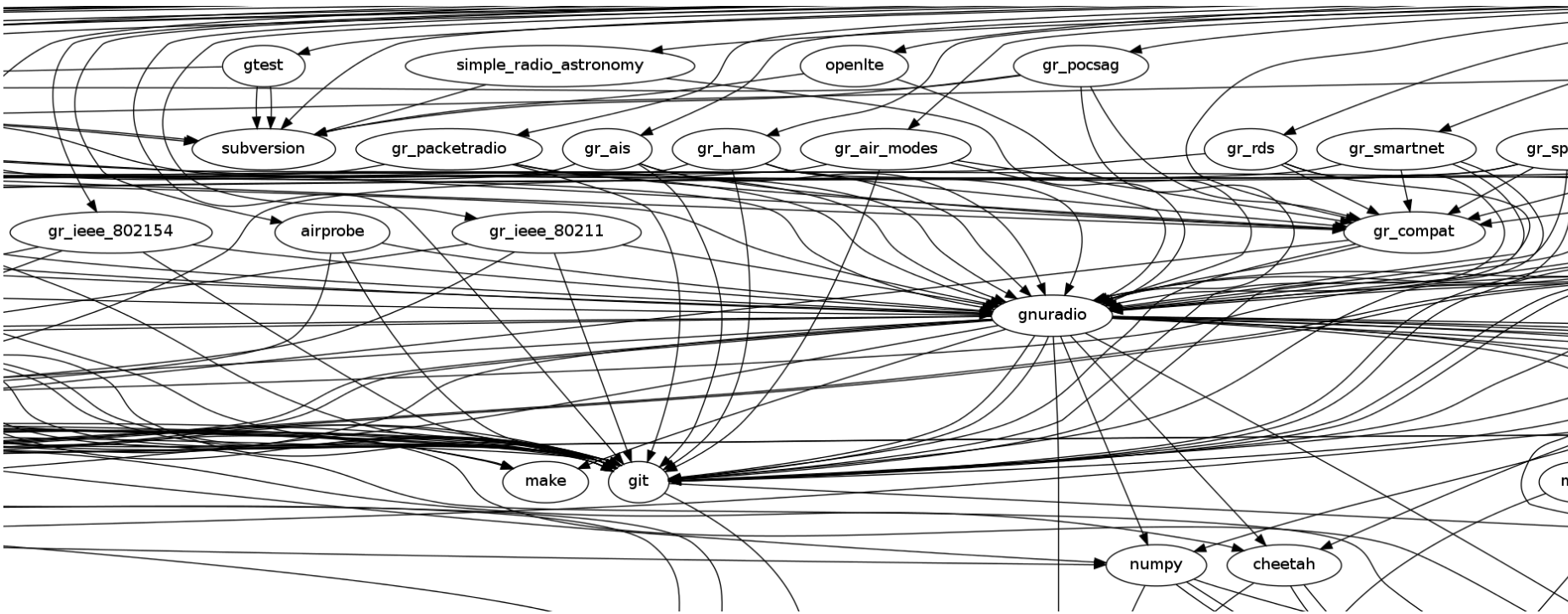
gitbranch: master

var config_opt = " -DENABLE_GR_FCD=ON
-DENABLE_DOXYGEN=OFF -DENABLE_GR_LOG=ON
-DENABLE_PERFORMANCE_COUNTERS=ON
-DCMAKE_CXX_FLAGS=-fpermissive "

inherit: cmake

Recipe Dependence Graphs help visually trace build problems

./pybombs digraph –all



satisfying requirements: distro binary packages (naming and versions - sometimes vary between distros) source builds direct from repository builds

The "App Store"

- Provides a minimal graphical front-end to installing and removing packages
- Started as kind of a joke
- Take it or leave it, it does help get the point of this effort across
- "Apps" are just OOT modules - each has its own maintainer and quality control
- "Store" is probably a misnomer, everything is free
- If this upsets you ... 18i44snyGMnrJiR6rm4fNEiBmZqTjd3ARX

Parting Tips

Don't

- run pybombs as sudo
- use /usr/local/ as your prefix
- expect a source overlay to always work flawlessly, there are a LOT of Linux distributions out there and this is of non-trivial complexity

Do

- Customize recipes to your liking
- Use github to fork and submit tickets and pull requests
- Create and submit recipes or repos for your personal OOT modules
- Provide large console logs and lots of info when submitting failure reports
- Get people set up in uniform pybombs environments in your lab when starting with GNU Radio

Questions?

- What gives you trouble?
- How does your lab get GNU Radio environments set up?
- Do you use many OOT modules?

Future Efforts

- Recipes with options (i.e. osmo-sdr can pull in hackrf, rtl-sdr, etc or not)
- Variable dependencies to suit variable options
- Do we want this complexity? Right now a recipe set/pybombs branch well defines what options will be installed
- pybombs gr-3.6 branch is now live on github as well, needs OOT testing

Everything released under GPLV3 Available at
http://github.com/pybombs/