Building and Packaging SAGE into a Mainstream GNU/Linux Environment

Bobby Moretti

May 25, 2006

Traditional UNIX Build Process

- Using GNU Autoconf and Automake, we generate a `configure` script
- Normally, we execute `./configure`, which (ideally) tests the functionality on our machine, generating a makefile

Traditional UNIX Build Process

- Using GNU Autoconf and Automake, we generate a `configure` script
- Normally, we execute `./configure`, which (ideally) tests the functionality on our machine, generating a makefile
- Then `make`, which builds the application

Traditional UNIX Build Process

- Using GNU Autoconf and Automake, we generate a `configure` script
- Normally, we execute `./configure`, which (ideally) tests the functionality on our machine, generating a makefile
- Then `make`, which builds the application
- Then `make install`, which should install the application (copying files and setting permissions)

How do we build SAGE?

- We download and untar
- We run `make`... there is no `configure` script
- SAGE is done, and we run it *in situ*.

```
bob@localhost$ tar xf sage-1.3.2.tar
bob@localhost$ make
[ lots of output ]
bob@localhost$ ./sage
```

and we are then greeted by the SAGE prompt.

Pros and cons of a monolithic distribution

- Pros:
  - SAGE can be easily made to run in its own `chroot` environment
  - We don't mess with the user's install
  - The user's install doesn't mess with us
  - Doesn't integrate with the environment
  - Almost "just works"
- Cons:
  - Doesn't integrate with the environment
  - Doesn't take advantage of the user's python libraries
  - Builds its own python, GAP, Maxima, Singular, etc.
  - Users will likely have two copies of many of these (especially python)
  - A 50 MB tarball of source code (may not seem to large, but we can do much better)

sage-libdist

We already have a monolithic release. . . wouldn't it be nice to have a SAGE that installs like a normal UNIX application?

- SAGE libdist: a distribution that is meant to be installed into an existing python in an existing GNU/Linux distribution
- 17 MB tarball
- How it works:
  - We run `sage -libdist`
  - The script takes a standard SAGE source package, untars it, eliminates software that is commonly available (like python, singular, GAP, etc.) and retars it
  - We untar this package, and run `python setup.py install`
  - The install script checks for a few depenedencies
  - If they are met, it builds SAGE sans those dependencies
  - Now `./sage` will run from the system's python interpreter (reading from the system's python libraries, the system's install of Maxima, GAP, etc.)

sage-libdist

- This is a good start, but it's not enough.
- SAGE still runs from its own build directory
- We need to move SAGE somewhere once it has finished being built à la `make install`
- Do you have experience with UNIX software that doesn't use the standard build tools? See me.

Package Management

- Most GNU/Linux distributions come with their own package management system
- Ideally, *any* software you want to run is available by typing a simple command, or using a simple GUI
- Ideally, it just works
- With some Linux distributions, it actually does (`apt-get install octave` in Debian/Ubuntu... and voilà, Octave is now installed)
- VERY convenient for the user!
- We would like to see SAGE installed this way, packaged on all the mainstream distros
- The goal:

```
bob@localhost$ apt-get install sage
```

w00t.

First try: Making a package for Ubuntu GNU/Linux

Why Ubuntu?

- Ubuntu has a very large userbase

First try: Making a package for Ubuntu GNU/Linux

Why Ubuntu?

- Ubuntu has a very large userbase
- Geared for the desktop (as opposed to the server)

First try: Making a package for Ubuntu GNU/Linux

Why Ubuntu?

- Ubuntu has a very large userbase
- Geared for the desktop (as opposed to the server)
- Ubuntu runs Debian's APT/deb package management system, which is very robust, very well tested, and fairly standard

First try: Making a package for Ubuntu GNU/Linux

Why Ubuntu?

- Ubuntu has a very large userbase
- Geared for the desktop (as opposed to the server)
- Ubuntu runs Debian's APT/deb package management system, which is very robust, very well tested, and fairly standard
- William and Alex use Ubuntu

First try: Making a package for Ubuntu GNU/Linux

Why Ubuntu?

- Ubuntu has a very large userbase
- Geared for the desktop (as opposed to the server)
- Ubuntu runs Debian's APT/deb package management system, which is very robust, very well tested, and fairly standard
- William and Alex use Ubuntu
- I use Ubuntu

Debian Packages

- There are two types of packages: source and binary
- To make a binary package, we take a source package and build it
- We want to make a binary package, so first we need a source package

The anatomy of a Debian package

- We start with a standard source tarball and untar it into a directory
- We run dh_make, which "Debianizes" the directory
- This actually just creates a directory, called `debian`, and adds a few files
- In this directory, several files have been created, the most important being
  - `control`
  - `copyright`
  - `changelog`
  - `rules`

The control file

- Contains various important information about the package
- Has a nice, simple, (somewhat) intuitive syntax
- The most important being the dependencies:

```
Depends: python2.4 (>= 2.4), flex, bison, GAP, ...
Conflicts: foo
Recommends:
Suggests: gcc (>= 3.0)
Replaces: bar (<< 5), bar-sage (<= 7.6)
```

Towards a binary package

- A `Makefile`, with instructions for Debian's `dpkg-buildpackage`
- `dh_make` tries to give us a nice one, based on the source's current `Makefile`
- However, with something as complicated as SAGE (14000 files to install!) we need some more custom engineering
- Does anyone know the canonical way of doing this?
- Once we have `rules`, we're in good shape. We use various debian package testing tools to test the integrity of the package
- Then we run `dpkg-buildpackage` to create our `.deb` file
- We add our own APT repository containing the `.deb`
- `bob@localhost$ sudo apt-get install sage`

In the near future...

I plan to:

- Come up with a `make install` target system
  - Thoroughly read the Debian New Maintainers tutorial (59 pages!)
  - Read the GNU autoconf installation framework documentation
  - Hopefully the solution will end up being obvious, once I see it
- Get a simple sage-monolithic debian package running
- Test it a bunch
- Put together a sage-libdist package
- Add it to a custom APT repository (users will add a custom repository to their `/etc/apt/sources.list` file)
- Eventually, get it added to a standard testing universe Ubuntu repository
- Finally, add it to the standard Ubuntu, Debian, Gentoo (...Slackware, Fink,...) package repositories