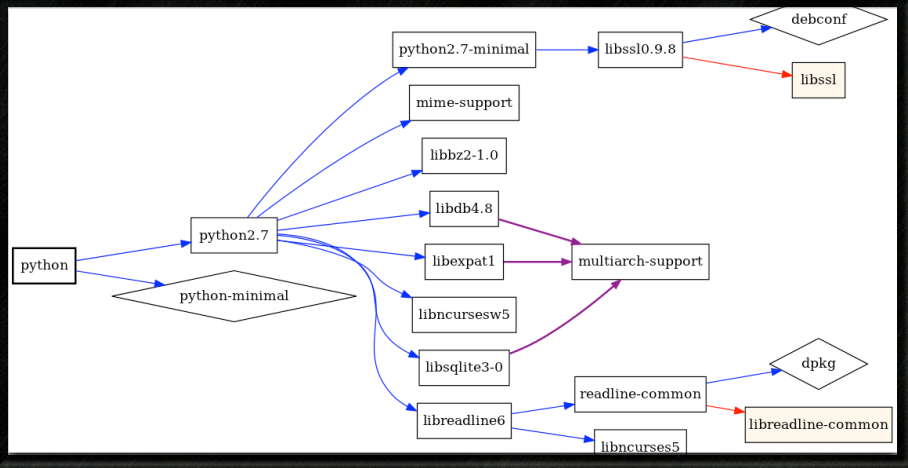


Managing Open Source Software on Workstations and Clusters

Theodore Kisner, LBNL

Why Use Open Source Software (OSS)?

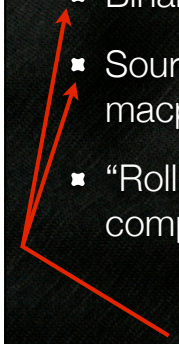
- Many useful tools for data processing and visualization
- Quality is usually good
- FREE! (as in purchase price == $0.00)
- No vendor lock-in
- Some people like the philosophy...
- Direct contact with developers if there are problems- even if you don't have a costly support contract.

Challenges

- For some specialized software needs or usage cases, OSS might not be sufficient (e.g. you require specific features that are not mainstream)
- Installation
  - Where do I find the software?
  - How do I put it on my computer?
  - How do I get rid of it later if I change my mind?
  - How do I upgrade to newer versions of the software?

Installation

- Binary package managers (APT, yum, zypper)
- Source-based package managers (portage, BSD ports, macports)
- “Roll your own”: manually download the sources and compile on your computer



Dependencies checked automatically,
official builds, good chance of working

Concrete Example: we want to install “matplotlib” to make plots from within python...

Binary Package Management (Ubuntu)

Search for what you want:

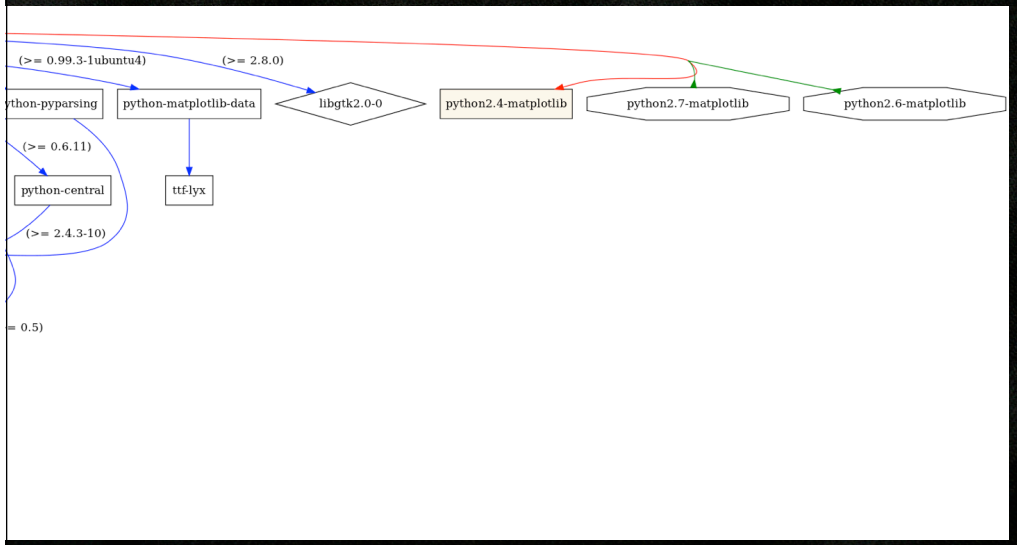
```
$> aptitude search matplotlib
p python-matplotlib      - Python based plotting system similar to Matlab
p python-matplotlib-data - Python based plotting system (data package)
p python-matplotlib-dbg  - Python based plotting system (debug extension)
p python-matplotlib-doc  - Python based plotting system (documentation package)
v python2.6-matplotlib   -
v python2.7-matplotlib   -
```

And... Install it!

```
$> aptitude install python-matplotlib
The following NEW packages will be installed:
  blt{a} python-dateutil{a} python-matplotlib python-matplotlib-data{a}
  python-pyparsing{a} python-tk{a} python-tz{a} tcl8.5{a} tk8.5{a} ttf-lyx{a}
0 packages upgraded, 10 newly installed, 0 to remove and 34 not upgraded.
Need to get 7,903 kB of archives. After unpacking 28.9 MB will be used.
Do you want to continue? [Y/n/?]
```

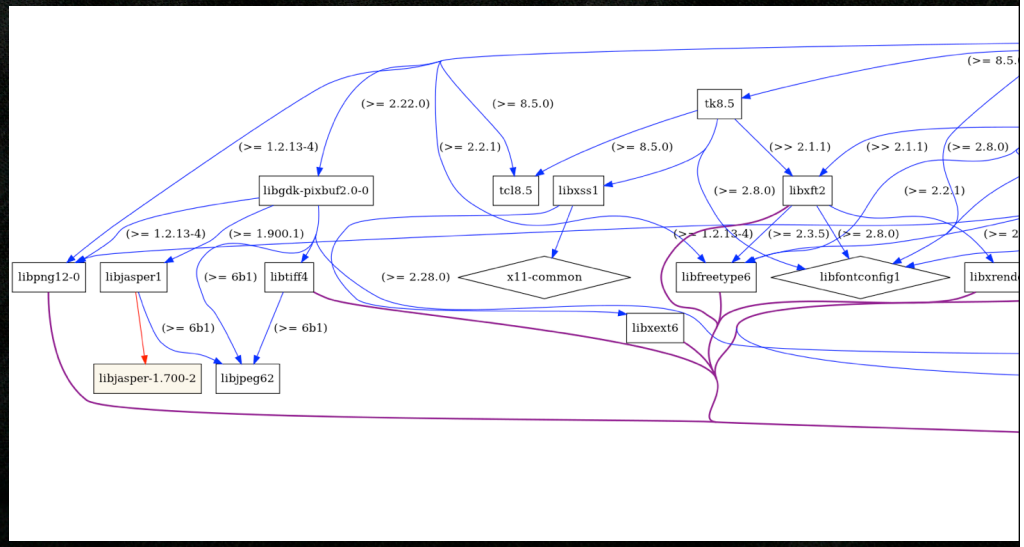
Binary Package Management (Ubuntu)

Dependency tracking



Binary Package Management (Ubuntu)

Dependency tracking



Binary Package Management (Ubuntu)

- Other tools to track which packages were install automatically and remove if no longer needed in the future.
- In a binary package management scheme, all the time-consuming building of packages is done on servers somewhere else!
- Scenario: developer uploads new source code to build server. Code is built automatically. After testing, it is available for you to download as an upgrade.

Source Package Management (Macports)

Search for what you want:

```
$> port search matplotlib
py-matplotlib @0.99.0 (python, graphics, math)
    matlab-like syntax for creating plots in python
py-matplotlib-basemap @0.99.4 (python, graphics, math)
    matplotlib toolkit for plotting data on map projections
py25-matplotlib @1.0.1 (python, graphics, math)
    matlab-like syntax for creating plots in python
py25-matplotlib-basemap @1.0.1 (python, graphics, math)
    matplotlib toolkit for plotting data on map projections
py26-matplotlib @1.0.1 (python, graphics, math)
    matlab-like syntax for creating plots in python
py26-matplotlib-basemap @1.0.1 (python, graphics, math)
    matplotlib toolkit for plotting data on map projections
py27-matplotlib @1.0.1 (python, graphics, math)
    matlab-like syntax for creating plots in python
py27-matplotlib-basemap @1.0.1 (python, graphics, math)
    matplotlib toolkit for plotting data on map projections
Found 8 ports.
```

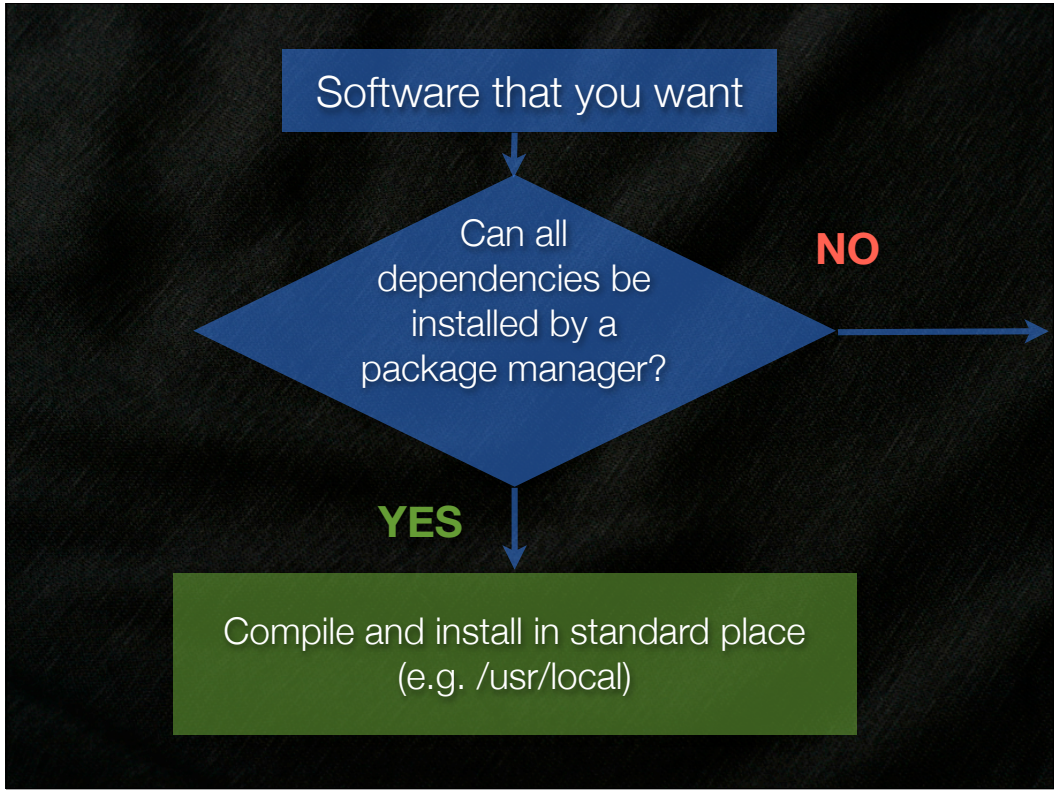
Source Package Management (Macports)

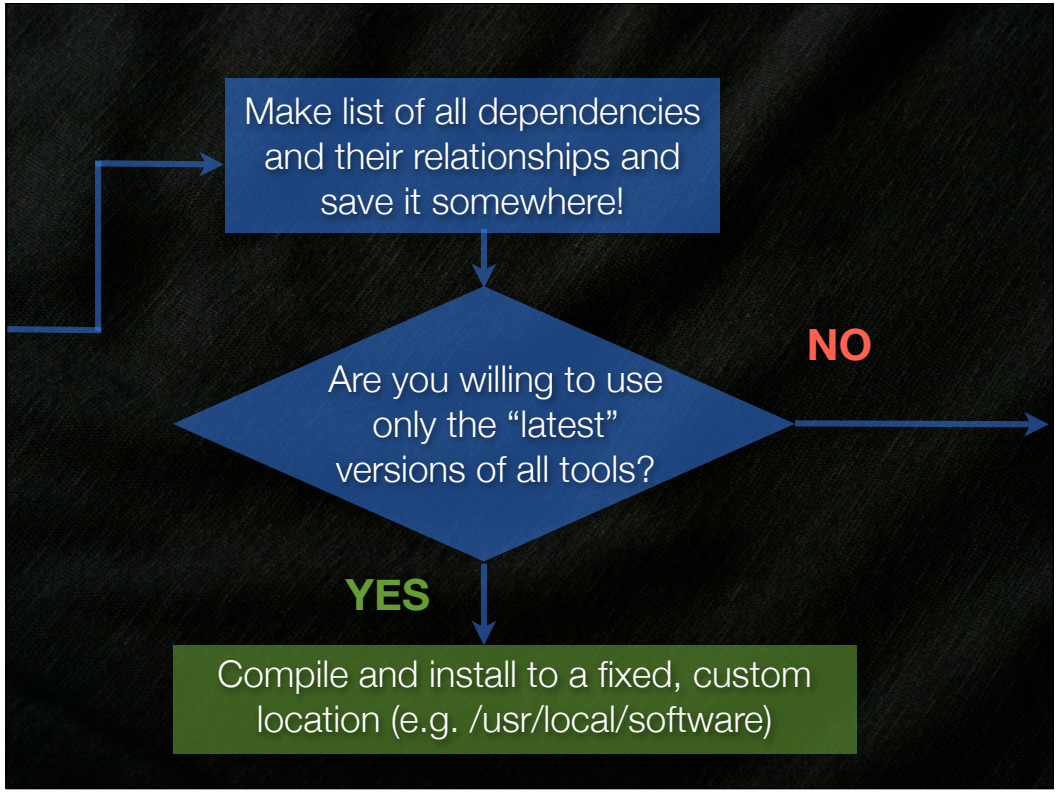
And... install it!

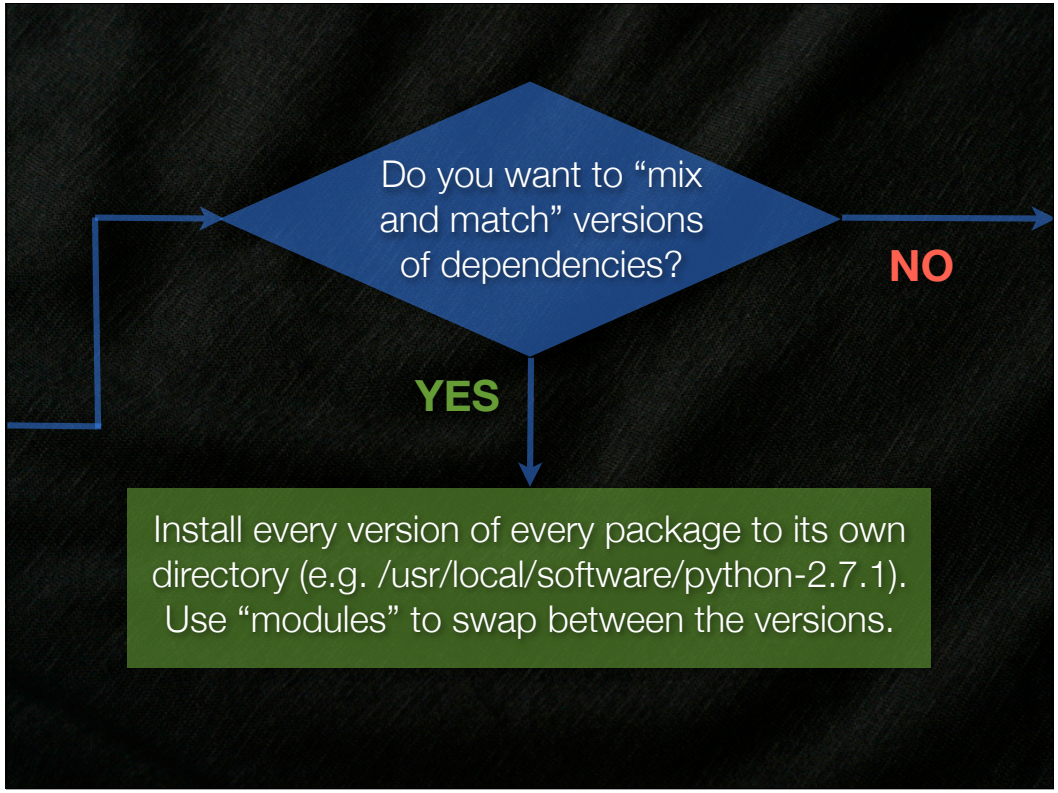
```
$> port install py27-matplotlib
--->  Computing dependencies for py27-matplotlib
--->  Dependencies to be installed: py27-configobj py27-dateutil py27-tz py27-
pyobjc-cocoa py27-pyobjc py27-py2app py27-bdist_mpkg py27-macholib py27-
modulegraph py27-altgraph py27-tkinter
--->  Fetching py27-configobj
--->  Attempting to fetch configobj-4.6.0.zip from http://cdnetworks-
us-2.dl.sourceforge.net/configobj
--->  Verifying checksum(s) for py27-configobj
--->  Extracting py27-configobj
--->  Configuring py27-configobj
--->  Building py27-configobj
--->  Staging py27-configobj into destroot
--->  Installing py27-configobj @4.6.0_0
--->  Activating py27-configobj @4.6.0_0
--->  Cleaning py27-configobj
--->  Fetching py27-tz
   <SNIP>  (more downloading, compiling, laptop churning, etc)  <SNIP>
--->  Installing py27-matplotlib @1.0.1_1+tkinter
--->  Activating py27-matplotlib @1.0.1_1+tkinter
--->  Cleaning py27-matplotlib
```

Why “Roll Your Own” Software Install?

- No package exists from the OS distribution, or existing package is too old
- Need to compile the software with non-standard options.
- You don’t have root access and the administrators are unresponsive (this will never happen!)
- Where to start: Decide what software you want and how it is going to be used- this will determine the best place to install.


Software that you want
Can all dependencies be installed by a package manager?
NO
YES
Compile and install in standard place (e.g. /usr/local)


Make list of all dependencies and their relationships and save it somewhere!
Are you willing to use only the “latest” versions of all tools?
NO
YES
Compile and install to a fixed, custom location (e.g. /usr/local/software)


Do you want to “mix and match” versions of dependencies?
NO
YES
Install every version of every package to its own directory (e.g. /usr/local/software/python-2.7.1). Use “modules” to swap between the versions.

Make directory for each “snapshot” of all package versions (e.g. /usr/local/software_20110617). Change values of PATH, LD_LIBRARY_PATH, MANPATH, etc when you want to switch between them.

Details of Software Building

- Remember: YOU are the package manager
- Look at all the dependencies and sketch out a picture of which tools depend on which.
- Download all the source tarballs. Start at the lowest level tools and work up the dependency chain.
- Use the same compiler version for all software. Record how you called “configure” for each piece of software and / or any edits to Makefiles, etc that you did.
- If you upgrade one tool, recompile anything that depends on it.

Considerations for Multi-user Systems

- If you break things, more than one person will complain!
- Use local disk for compilation (faster; no risk of clock skew), and install to shared filesystems if desired.
- When “rolling your own” and upgrading some of the software, rebuild everything and install to a new prefix. Then swap environment variables to the new location.
- Can use “modules” to allow users to load old versions of software for debugging / validation.

Conclusions

- Use the recommended package management system(s) for your platform when ever possible!
- Manual installation of large software packages with many dependencies is painful.
- It would be nice to have a source-based package manager that would work across all UNIX systems, similar to FreeBSD Ports / Macports.