

04 - Package Management

CS 2043: Unix Tools and Scripting, Spring 2017 [1]

Stephen McDowell

February 1st, 2017

Cornell University

Table of contents

1. Package Management
2. System Specific Package Managers
3. Other Managers

Some Logistics

- Last day to add is today!
- My office hours...
- Today's slides & taking notes on em.

Package Management

Package Management Overview

- If you had to give one reason why Unix systems are superior to Windows: Package Management.
- Provides the capability to install almost anything you can think of from your terminal.
- Update to the latest version with one command.
 - No more download the latest installer nonsense!
- Various tools can be installed by installing a *package*.
 - A package contains the files and other instructions to setup a piece of software.
 - Many packages depend on each other.
 - High-level package managers download packages, figure out the dependencies for you, and deal with groups of packages.
 - Low-level managers unpack individual packages, run scripts, and get the software installed correctly.
- In general, these are "pre-compiled binaries": no compilation necessary. It's already packaged nice and neat just for you!

Package Managers in the Wild

- GNU/Linux:
 - Two general families of *packages* exist: **deb**, and **rpm** (low-level).
 - High-level package managers you are likely to encounter:
 - Debian/Ubuntu: **apt-get**.
 - Some claim that **aptitude** is superior, but I will only cover **apt-get**. They are roughly interchangeable.
 - SUSE/OpenSUSE: **zypper**.
 - Fedora: **dnf** (Fedora 22+).
 - **zypper** and **dnf** use **SAT**-based dependency solvers, which many argue is fundamentally superior. Though the dependency resolution phase is usually not the slowest part...installing the packages is. See [3] for more info.
 - RHEL/CentOS: **yum** (until they adopt **dnf**).
- Mac OSX:
 - Others exist, but the only one you should ever use is **brew**.
 - Don't user others (e.g. **port**), they are outdated / EOSL.

Using Package Managers

- Though the syntax for the commands are different depending on your OS, the concepts are all the same.
 - This lecture will focus on **apt-get**, **dnf**, and **brew**.
 - The **dnf** commands are almost entirely interchangeable with **yum**, by design.
 - Note that **brew** is a "special snowflake", more on this later.
- What does your package manager give you? The ability to
 - **install** new packages you do not have.
 - **remove** packages you have installed.
 - **update*** installed packages.
 - update the lists to search for files / updates from.
 - view **dependencies** of a given package.
 - a whole lot more!!!

* See next slide for a potential **update** pitfalls.

A Note on **update**

- The **update** command has importantly different meanings in different package managers.
- Some (**deb**) do not default to system (read linux kernel) updates.
- Some (**rpm**) DO default to system updates!
 - Even this is not true, it really depends on your OS:
 - Fedora: default is *yes*.
 - RHEL: default is *no*.
 - Know your operating system, and look up what the default behavior is.
- The difference lies somewhat in philosophy, and somewhat in the differences between the two.
- If your program needs a specific version of the linux kernel, you need to be very careful!
- In the end, it actually has less to do with the type of package manager, but more to do with who is packaging things.

A Note on Names and their Meanings

- You may see packages of the form:
 - `<package>.i[3456]86` (e.g. `.i386` or `.i686`):
 - These are the **32-bit** packages.
 - `<package>.x86_64`: these are the **64-bit** packages.
 - `<package>.noarch`: these are independent of the architecture.
- Development installations can have as many as three packages you need to install, e.g. if you need to compile / link against a package in a C/C++ or often times even Python, Java, and many more languages.
 - The header files are usually called something like:
 - **deb**: usually `<package>-dev`
 - **rpm**: usually `<package>-devel`
 - The library you will need to link against:
 - If applicable, **lib**`<package>` or something similar.
 - Many of these may also have binaries (executables), which are just provided by `<package>`.

Example Development Installation

- For example, if I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on Fedora, I would have to install
 - `libXrandr`: the library.
 - `libXrandr-devel`: the header files.
 - Not including `.x86_64` is OK / encouraged, your package manager knows which one to install.
 - Though in certain special cases you may need to get the `32-bit` library as well.
 - In this case, if I were compiling a program that links against `libXrandr`, but I want to release a pre-compiled `32bit` library, it must be installed in order for me to link against it.
- The `deb` versions should be similarly named, but just use the `search` functionality of `find` to find the right names.
- This concept has no meaning for `brew`, since it compiles everything.

System Specific Package Managers

Debian / Ubuntu Package Management

- Installing and uninstalling:
 - Install a package:
`apt-get install <pkg1> <pkg2> ... <pkgN>`
 - Remove a package:
`apt-get remove <pkg1> <pkg2> ... <pkgN>`
 - Only one **pkg** required, but can specify many.
 - "Group" packages are available, but still the same command.
- Updating components:
 - Update lists of packages available: `apt-get update`.
 - No arguments, it updates the whole list (even if you give args).
 - Updating currently installed packages: `apt-get upgrade`.
 - If you instead specify a **package** name, it will only update / upgrade that package.
 - Update core (incl. kernel): `apt-get dist-upgrade`.
- Searching for packages:
 - Different command: `apt-cache search <pkg>`

RHEL / Fedora (**yum** and **dnf**)

- Installing and uninstalling:
 - Install a package:
`dnf install <pkg1> <pkg2> ... <pkgN>`
 - Remove a package:
`dnf remove <pkg1> <pkg2> ... <pkgN>`
 - Only one **pkg** required, but can specify many.
 - "Group" packages are available, but different command:
`dnf groupinstall 'Package Group Name'`
- Updating components:
 - Update EVERYTHING `dnf upgrade`.
 - **update** exists, but is essentially **upgrade**.
 - Specify a **package** name to only upgrade that package.
 - Updating repository lists: `dnf check-update`
- Searching for packages:
 - Same command: `dnf search <pkg>`

dnf: Cautionary Tales

- **WARNING:** if you install package **Y**, which installs **X** as a dependency, and later **remove Y**
 - By default, **X** will be removed!
 - Refer to [4] for workarounds.
 - Generally, it's impossible to know you needed to **mark** until its too late.
- Solution?
 - Basically, **pay attention to your package manager.**
 - It gets removed because nothing *explicitly* depends on it.
 - So one day you may realize "OH NO! I'm missing package **X**"...
 - ...so just **dnf install X.**
 - So while **mark** is available, personally I don't use it.
 - Sad face, I know. Just the way of the world.

OSX Package Management: Install **brew** on your own

- Sitting in class right now with a Mac?
- **WAIT UNTIL LATER TO FOLLOW THESE.** You will want to make sure you do not have to interrupt the process.
 1. Make sure you have the "Command Line Tools" installed.
 - Instructions are on the First Things First Config Page
 2. Visit <http://brew.sh/>
 3. Copy-paste the given instructions in the terminal *as a regular user (not **root!**)*.
 5. VERY IMPORTANT: READ WHAT THE OUTPUT IS!!!! It will tell you to do things, and you *have* to do them.
Specifically:
"You should run 'brew doctor' **before** you install anything."

OSX: Using **brew**

- Installing and uninstalling:
 - Install a *formula*:
`brew install <fmla1> <fmla2> ... <fmla2>`
 - Remove a formula:
`brew uninstall <fmla1> <fmla2> ... <fmlaN>`
 - Only one **fmla** required, but can specify many.
 - "Group" packages have no meaning in **brew**.
- Updating components:
 - Update **brew**, all *taps*, and installed formulae listings. This does not update the actual software you have installed with **brew**, just the definitions (more on next slide): `brew update`.
 - Update just installed formulae: `brew upgrade`.
 - Specify a **formula** name to only upgrade that formula.
- Searching for packages:
 - Same command: `brew search <formula>`

OSX: One of These Kids is Not Like the Others (Part I)

- Safe: confines itself (by default) in `/usr/local/Cellar`:
 - No **sudo**, plays nicely with OSX (e.g. Applications, **python3**).
 - Non-linking by default. If a conflict is detected, it will tell you.
 - Really important to read what **brew** tells you!!!
- **brew** is modular. There is a main list of repositories, but there are also additional *taps*:
 - A tap is effectively another repository list, like what a `.rpm` or `.deb` would give you in linux.
 - Common taps people use:
 - `brew tap homebrew/science`
Various "scientific computing" tools, e.g. **opencv**.
 - `brew tap caskroom/cask`
Install `.app` applications! Safe: installs in the "Cellar", symlinks to `~/Applications`, but *now these update with brew all on their own!*
E.g. `brew cask install vlc`

OSX: One of These Kids is Not Like the Others (Part II)

- **brew** installs *formulas*.
 - A formula is *not* a pre-compiled binary, it is a **ruby** script that provides rules for where to download something from / how to compile it.
 - You download a **bottle** that gets *poured*: download source and compile (ish).
 - Though more time consuming, can be quite convenient!
 - **brew options opencv**
 - **brew install --with-cuda --c++11 opencv**
 - It really really really is magical. No need to understand the **opencv** build flags, because the authors of the **brew** formula are kind and wonderful people.
 - **brew reinstall --with-missed-option formula**
- Of course, there is a whole lot more that **brew** does, just like the other package managers.

OSX: One of These Kids is Not Like the Others (Part III)

- You REALLY need to pay attention to **brew** and what it says. Seriously.
- Example: after installing **opencv**, it tells me:

```
==> Caveats
Python modules have been installed and Homebrew's site-packages is not
in your Python sys.path, so you will not be able to import the modules
this formula installed. If you plan to develop with these modules,
please run:
  mkdir -p /Users/sven/.local/lib/python2.7/site-packages
  echo 'import site; site.addsitedir("/usr/local/lib/python2.7/site-packages")' >> \
    /Users/sven/.local/lib/python2.7/site-packages/homebrew.pth
# (continued onto newline so you can read, it gives you copy-paste format!)
```

- Obviously I want to use **opencv** with **Python**, so I am going to follow what **brew** tells me to do.
- If it may cause problems, it will tell you what the problems might be.

Less Common Package Management Operations

- Sometimes when dependencies are installed behind the scenes, and you no longer need them, you will want to get rid of them.
 - `apt-get autoremove`
 - `dnf autoremove`
 - `brew doctor`
- View the list of repositories being checked:
 - `apt-cache policy` (well, sort of...`apt` doesn't have it)
 - `dnf repolist [enabled|disabled|all]`
 - Some repositories for `dnf` are *disabled* by default (with good reason). Usually you want to just
`dnf enablerepo=<name> install <thing>`
e.g. if you have `rawhide` (development branch for fedora).
 - `brew tap`

Other Managers

Like What?

- There are so many package managers out there for different things, too many to list them all!
- Ruby: **gem**
- Anaconda Python: **conda**
- Python: **pip**
- Python: **easy_install** (but really, just use **pip**)
- Python3: **pip3**
- \LaTeX : **tlmgr** (uses the CTAN database)
- Perl: **cpan**
- Sublime Text has its own package manager: Package Control.
- Many many others...

Like How?

- Some notes and warnings about Python package management.
- Notes:
 - If you install something with **pip**, and try to use it with Python3, it will not work. You have to also install it with **pip3**.
 - OSX Specifically: advise only using **brew** or Anaconda Python. The system Python can get really damaged if you modify it, you are better off leaving it alone.
 - So even if you want to use **python2** on Mac, I strongly encourage you to install it with **brew**.
- Warnings:
 - Don't mix **easy_install** and **pip**. Choose one, stick with it.
 - But the internet told me if I want **pip** on Mac, I should **easy_install pip**
 - NO! Because this **pip** will modify your *system* python, **USE BREW**.
 - Don't mix **pip** with **conda**. If you have Anaconda python, just stick to using **conda**.

Like **thefuck**

- Let's install something!

```
$ pip install thefuck
```

- What does it do? Justify your emotions when you get something wrong...
- Checkout the GitHub page in [2]

References I

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.

Previous cornell cs 2043 course slides.

[2] V. Iakovlev.

Magnificent app which corrects your previous console command.

<https://github.com/nvbn/thefuck>.

[3] Linux.com.

What you need to know about fedora's switch from yum to dnf.

<https://www.linux.com/learn/tutorials/838176-what-you-need-to-know-about-fedoras-switch>

[4] Reddit.com.

Dnf remove package, keep dependencies??

https:

[//www.reddit.com/r/Fedora/comments/3pqr9/
dnf_remove_package_keep_dependencies/](https://www.reddit.com/r/Fedora/comments/3pqr9/dnf_remove_package_keep_dependencies/).