17 – Package Management, for real this time

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano March 4, 2019

Cornell University

Table of Contents

- 1. Package Management
- 2. System Specific Package Managers

Package Management

Package Management Overview

- If I had to give only one reason why Unix systems are superior to Windows: Package Management.
- · Can install almost anything with ease 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.

Many different philosophies

- Monolithic binary packages: one big "app" with everything bundled together
 - · docker containers, most windows programs
- Small binary packages: separate common code into independently-installed "libraries"
 - · MSI files, Ubuntu, most of linux
- Source-based packages: no installers at all! Compile all your programs
 - · language-based package managers, brew, portage
- Benefits to all approaches
 - · monolithic binary: fastish install, very independent programs
 - · small binary: very fast install, less wasted space
 - source-based: fastest code, smallest install, easy to use open-source

Package Managers in the Wild

GNU/Linux:

- Low-level: two general families of binary packages exist: deb, and rpm.
- · High-level package managers you are likely to encounter:
 - · Debian/Ubuntu: apt-get, apt, aptitude.
 - · SUSE/OpenSUSE: zypper.
 - Fedora: dnf (Fedora 22+) / yum.
 - RHEL/CentOS: **yum** (until they adopt **dnf**).
 - · Arch: pacman
 - Gentoo: Portage, emerge (my favorite)

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- Don't user others (e.g. **port**), they are outdated / EOSL.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 each package manager is different, the concepts are all the same.
 - This lecture will focus on apt, dnf, emerge, 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!!!

A Note on **update**

- These "subcommands" are by category, not name: update is not always called update
- The update command has importantly different meanings in different package managers.
- · Most do **not** default to system (read linux kernel) updates.
 - · Fedora does; most others do not.
- It depends on your operating system, and package manager.
 - Know your operating system, and look up what the default behavior is.
- If your program needs a specific version of the linux kernel, you need to be very careful!
 - · very, very few programs care about your kernel version.

A Note on Names and their Meanings

- Package names sometimes specify architecture:
 - [3456x]86 (e.g. .i386 or .i686 or x86):
 - These are the **32-bit** packages.
 - x86_64 or amd64: these are the 64-bit packages.
 - · noarch: these are independent of the architecture.
- Ubuntu / fedora often splits packages into smaller pieces:
 - 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.
 - The binaries (executables), often provided by just <package>.
 - Most relevant for C and C++, but also Python and others.
 - · Use the **search** functionality of your package manager.

Example Development Tool Installation

- If I needed to compile and link against Xrandr (X.Org X11 libXrandr runtime library) on ubuntu, I would have to install
 - · libxrandr2: the library.
 - · libxrandr-dev: the header files.
 - Usually don't explicitly include the architecture (e.g. .x86_64), it's inferred
 - If you're getting link errors, try installing explicit 32/64-bit version.
 - · just google your error
- Splitting devel files more common for binary package managers, less for source-based ones.

System Specific Package Managers

Debian / Ubuntu Package Management (apt)

- Installing and uninstalling:
 - Install a package:

```
apt install <pkg1> <pkg2> ... <pkgN>
```

- · Remove a package:
 - apt 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 update.
 - No arguments, it updates the whole list (even if you give args).
 - · Updating currently installed packages: apt upgrade.
 - Specify a **package** name to only update / upgrade that package.
 - · Update core (incl. kernel): apt dist-upgrade.
- Searching for packages:
 - Different command: apt-cache search <pkg>

RHEL / Fedora Package Managers (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>
- · yum and dnf (Dandified Yum) nearly interchangeable: [2].

Gentoo package manager (portage with emerge)

- · source-based package manager: compiles your packages
 - just runs a special bash script to compile
 - very, very fine-grained control over dependencies and features
 - · use the latest software specialized to your hardware!
- USE flags control special "optional" features
 - · would be separate packages on ubuntu
 - · Want java or emacs integration? USE="java emacs..."
- Installing, uninstalling, and updating
 - emerge package to install
 - emerge -v --depclean to remove
 - explicitly checks to ensure other packages don't need it first
 - emerge -uND @world to upgrade everything
 - flags are "update", "newuse" (if you turned on a feature), "deep" (also check dependencies for this stuff)

Cautionary Tales

- WARNING: if you install package Y, which installs X as a dependency, and later remove Y
 - · Sometimes X will be removed immediately!
 - · Sometimes X will be removed during a cleanup operation later
- · Solution?
 - · Basically, pay attention to your package manager.
 - Install packages explicitly that you need
 - Check lists of packages when removing things
- Why does this happen at all?
 - Linux splits things into dependencies: avoids lots of extra copies
 - Side effect: dependencies are visible to you; you can use directly
 - · In windows: dependencies are hidden

Package Management is a core Philosophy

- · Most of what makes a Linux distribution is its package manager
- Reflects Distribution's philosophy
 - Ubuntu: "just work" and don't think too hard
 - · Fedora: "latest everything" but keep it stable+not too hard
 - · Arch: I want to understand how my distro works.
 - Gentoo: I do understand how my distro works.

If you're thinking of installing Linux, by the way...

Ubuntu

- Benefits: easy install, out-of-the-box setup, common things "just work"
- Drawbacks: too much magic; system "just work" scripts break if you need to do too many uncommon things and aren't really careful

· Fedora

- Benefits: still pretty easy to install, lots of good "get started quick" stuff. Good in a VM too
- Drawbacks: a little less stable; can change deep system things but also not hard to break your system that way.

If you're thinking of installing Linux, by the way...

Arch

- Benefits: wealth of knowledge, really helps you understand why your system works and what makes it work
- Drawbacks: limited automagic. Takes real time to set things up, or change important things.

Gentoo

- Benefits: similar to Arch, plus the source-based Portage package manager is pure gold. Great if you're doing serious programming/systems work, or if you really need a thing from github that was released last week, or you have a limited environment. Great way to really learn Linux.
- Drawbacks: absolutely no automagic. Takes real time to set things up, compiling is time-consuming, all the docs think you know what you're doing.

References

- [1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. "Previous Cornell CS 2043 Course Slides".
- [2] Jack Wallen. What You Need to Know About Fedora's Switch From Yum to DNF. 2015. URL: https://www.linux.com/learn/tutorials/838176-what-you-need-to-know-about-fedoras-switch-from-yum-to-dnf.