Lecture 06: Software management or “How I Learned to Stop Worrying and Love APT” Part 2
Hands-on Unix system administration DeCal
2012-10-08

Make install sucks:

- not guaranteed to work consistently
- takes time (human and computational)
- disorganized
  - many files install to /usr/local, but not always, can install anywhere on system, and possibly overwrite or interfere with other things
  - What is installed? And what version? What installed software is vulnerable to a exploit? What software has a major bug?
- make uninstall target doesn't always exist or work cleanly
  - need to keep copy of source

Dependency hell:

- many dependencies, tedious
  - chains of dependencies (dependencies have dependencies themselves)
- conflicting dependencies
  - different versions and variants which may be incompatible, but no specific “canonical” documentation
- circular dependencies (dependencies may be dependent on original software)

Upgrading compounds problems:

- you need to keep track of updates (security updates and bugfixes) — so knowing what is installed, what versions, which vulnerable, is important
- you need to again install fetch, compile, and deal with dependency hell
- you need to ensure compatibility during update, update other software at the same time, and pray nothing breaks

Package management:

- Unix distribution provides a central repository of packages each with a different name
- packages can be source packages which are meant to be compiled during installation or binary which have already been compiled
- packages are maintained according to a policy manual
  - “upstream” is modified as necessary to ensure combinations of packages are compatible and act consistently
- metadata associated with each package identifies version, dependencies, checksum/signature
- package manager handles installing, smart upgrading (order of operations), configuring, and removing software
- blurs the boundaries between operating system and applications — OS is itself a bunch of packages
- saves disk space and memory space because software can use shared libraries
- possibly the greatest feature of Unix distributions, especially GNU/Linux

As compared with an installer (“Windows”):

- package management: single installation database managed the same way by operating system
- installer: each program manages its own installation in inconsistent ways, possibly recording this information its own format
- installers tend to be buggier, but more up-to-date since they are prepared as part of the “upstream” software
- package managers can keep track of all installed software, find updates, make clean removals, but they can lag behind, which is not necessarily a bad thing (with features come bugs)
- Microsoft now recommends MSI (Windows Installer format), which is a package system without the central repository to fetch, install, and update from

Debian package management:

- robust package management introduced in 1993 with Debian, a GNU/Linux distribution — the universal OS (flexible and powerful)
- source and binary packages: you can modify source packages, and build binary packages (*.deb archives) for different architectures and kernels, which are actually used for fast and clean installations
- types of dependencies (by name and version)
  - build: required dependencies for compiling source packages (e.g., gcc)
  - depends: required dependencies for installations (e.g., libc)
  - recommends: recommended dependencies (apt/aptitude installs by default)
  - suggests: suggested packages
  - also breaks, conflicts, provides, replaces
- different package versions for different releases of Debian OS
  - upgrade from release to release by upgrading packages
- dpkg: low-level Debian package manager and package format
- aptitude/apt: add networked capability to search, fetch, and install or upgrade from packages available through the Debian archives specified in source.list file

Why compile:

- even with binary package management, you may need a newer version which is unavailable (hasn’t yet been packaged) or you may need to apply a patch for further customization or bugfixes
- packaging software, i.e., developing packages, involves compiling software, then more, so Debian developers know dependency hell well to save us from it

Software freedom:

- software that can be freely run, studied, modified, adapted, improved, copied, distributed, and redistributed
- in contrast, proprietary software is restricted by copyright and contracts (NDAs, EULAs)
- aka open source, access to source code is a prerequisite
- package management “depends” on free software because software is packaged and patched in source form, can be compiled for desired architecture, with shared libraries, etc.

Useful commands:

- dpkg -i: install a Debian binary package file (typically handled through aptitude)
- dpkg -l: list all managed packages or specified package (does not include available packages, those are not handled by low-level dpkg), status, and version
- dpkg -L: list files installed (provided by) from specified package
- dpkg -S: find package which provides a file
- aptitude update: update local cache of available packages
- aptitude upgrade: fetch and upgrade to latest available all or specified packages (run aptitude update to update cache first)
- aptitude install: fetch and install package from repository