Software installation and management

Maxim Storetvedt

Department of Computing, Mathematics, and Physics

January 17, 2018

Western Norway
University of
Applied Sciences

Operating system Installation

- Modern Linux distributions have straightforward procedures for basic installation
  - Live images with user-oriented GUIs
  - Simple configuration options
  - Built-in partition tools
- Great for users, but not so much for system admins
  - Imagine repeating the same process on 100 machines!

Operating system Installation

- Fortunately, tools for automating the installation exist
  - **Kickstart**, for distributions based on Red Hat Linux (CentOS, Fedora, Qubes)
  - Debian-installer, for distributions based on Debian Linux (Ubuntu, Mint, Elementary)
  - Bsdinstall, for FreeBSD systems

Operating system Installation

- Kickstart
  - Red Hat developed tool for automated installations
  - Essentially a scripting interface to the standard installer software, Anaconda.
  - Flexible, with automated hardware detection
  - Behaviour is controlled by a single config file
    - **ks.cfg**
  - This file is divided into three parts:
    - A command section, for language, keyboard and time zone options. The source of the distribution is also specified here
    - A list of packages to install
    - A collection of shell commands to run. Can be set to run before and/or after the installer
  - A GUI tool for customising ks.cfg is available

Operating system Installation

- Network installation
  - Installing from the network can simplify deployment, and is a good option for sites with more than ten or so systems
  - A completely hands-free installation can be provided by PXE – The Preboot Execution Environment
    - A standard from Intel that allows booting from a network interface

Operating system Installation

- PXE
  - Acts like a miniature OS in the ROM of a system's network card
  - Exposes network capabilities through a standardised API for the system BIOS
    - Allows netbooting of any PXE enabled PC without special drivers

Operating system Installation

- PXE installation process
  - The client sends a DHCP request with a PXE flag
  - The DHCP server responds with the name of the boot image and the server containing it
  - This image is executed by the client, giving it a menu of pointers to the available OS boot images
- PXE can also be used to boot diskless systems such as thin clients

Operating system Installation

- Cobbler
  - An open source provisioning server
    - Can act as a boot server for PXE
  - Bundles all important netboot features
  - Helps manage the different OS images
  - Has template support that can be used with
    - Kickstart configuration
    - Disk partitions
    - Packages
    - Repositories
    - Other localisation requirements

Managing Packages

Managing Packages

- Traditionally, Linux software has been distributed as compressed archives, containing
  - The source code
  - Build files
  - Documentation
  - Configuration templates
- Inconvenient
  - The source archives had to be manually compiled and built for each program
  - A tedious and error prone process

Managing Packages

- Packaging systems
  - Emerged as a way to facilitate software management
  - Here, packages include all the files required to run a piece of software, including
    - Binaries
    - Dependency information
    - Configuration templates
  - More importantly: make the installation process as straightforward as possible
    - If an error occurs, a package can be backed out and reapplied
    - New versions of a software can be installed with a simple package update

Managing Packages

- Packaging systems
  - More than just automatic archive unpackers
    - Can run scripts at various points during installation
    - Adds new users and groups, runs sanity checks and customises settings according to the environment
  - Typically aware of config files, and will attempt avoid changing these
    - May do a backup before any changes
    - Or provide an alternative configuration file as a template
  - Theoretically, it should be possible to revert any changes back if an installed package breaks something
    - Emphasis on this only being **theoretical**!
    - Always do testing before applying new packages on a production system

Managing Packages

- Packaging systems
  - Define a dependency model that allows package maintainers to ensure that all required libraries and infrastructure their software depends on are installed on the system
  - However, the dependency graphs may be imperfect
    - May at times be impossible to install/update software due to dependencies having version incompatibilities

Package management systems

- Two package formats are commonly found within Linux systems
  - RPM (RPM Package Manager), used by Red Hat based Linux distributions
  - DEB (Debian packages), for Debian based Linux distributions
- These packages formats have each their own corresponding package systems

Package management systems

- The package management systems can be divided into two separate layers
  - A **lower level** that contains the tools used for installing, uninstalling and querying the packages
  - A **higher level** containing systems that can download packages from the internet, analyse interpackage dependencies, and upgrade the packages present on the system

Package management systems

- Lower level tools
  - The RPM and DEB package formats both have their own tools at the lower level for installation, uninstallation and package querying
    - RPM uses a tool with the same name: rpm
    - DEB uses dpkg

Package management systems

- rpm
  - Can be seen as multiple commands that share same name
  - The exact command can be specified by supplying the appropriate flag
    - -i (install)
    - -U (update)
    - -e (erase)
    - -q (query)
  - To update, say OpenSSH, the following command can be run after downloading the package containing the desired version:
    - **rpm -U openssh-6.6.1.rpm**, where openssh-6.6.1.rpm is the downloaded package

Package management systems

- rpm
  - However, note that if there are packages that depend on the OpenSSH version currently present on the system, the previous command will then fail
    - One possible solution can be to include the **–force** option, forcing rpm to install the new version anyway (and hope for the best)
    - Alternatively, the dependent packages could be updated as well.
  - A list of the dependent packages can be generated using the -q flag with the -whatrequires option: **rpm -q whatrequires openssh**
  - In fact, if their files were also supplied in the command on the previous page, rpm would automatically install these in the right dependency order (granted they support the new OpenSSH version)

Package management systems

- dpkg
  - In many ways similar to rpm, with slightly different syntax, e.g:
    - -i (for installing packages)
    - -r (for uninstalling (removing) packages)
    - -l (for listing all installed packages. No query here. Can also be used for searching
  - More “chatty” than rpm, giving more details on what it’s currently doing

High-level package management systems

- Also known as “metapackage management systems”
- Not concerned with installation specifics. These tasks are handled by tools such as rpm and dpkg
- Instead, these high-level package management systems aim to
  - Simplify the task of locating and downloading packages
  - Automate the process of updating and upgrading systems
  - Facilitate the management of interpackage dependencies

High-level package management systems

- It becomes evident that in order to achieve their goals, these systems must include more than client-side commands
  - Software must be found, located and downloaded
  - This while also being compatible
- The distribution maintainers should somehow provide an organised way for clients to access software when needed
  - But no single supplier can encompass “the whole world of Linux software”
- Solution: having multiple “software repositories"

High-level package management systems

- Software repositories
  - Package management systems tend to point to one or more web or ftp servers with repositories of software
  - These are commonly maintained by Linux distributers, and tailored for your specific distribution
  - But what should these repositories contain?
    - Formal releases with security updates?
    - Up-to-date versions of all the packages?
    - Useful third-party software, not officially supported?
    - Source code?
    - Binaries for other hardware architectures?
  - And which one of these should be prioritised by the repository?

High-level package management systems

- Repositories may contain all of the previously mentioned
- The package management systems must decide what to include in their software “world"
  - Some repository terminology to help structure the process
    - A **release** is a self-consistent snapshot of the package universe, and contains multiple components
    - A **component** is a subset of the software within a release, and consists of packages
    - A **package** is a piece of software, often in a package management format (e.g .deb or .rpm). These may be specific for a certain architecture
    - An **architecture** represents a class of hardware. The expectation is that machines within a specific class are similar enough to run the same binaries. E.g x86 and x64 systems.

High-level package management systems

- APT: the Advanced Package Tool
  - High-level package management system mainly used in (but not limited to) Debian based distributions
  - Mature and feature rich
    - Can update a full system full of software with a single **apt** command
    - Can also be automated to continuously keep a system updated without human intervention
  - Includes a suite of commands, such as
    - **apt-get** (for updates and installations), and
    - **apt-cache** (queries installed applications)
  - These can be wrapped in an omnibus **apt** command
    - E.g to install the OpenSSH client, you can either call apt-get directly:

      apt-get install openssh-client
    - Or simply call

      apt install openssh-client
    - Which will then redirect the command to the appropriate tool (apt-get)

High-level package management systems

- YUM: Yellowdog Updater, Modified
  - High-level package management system for distributions using RPM (Red Hat, CentOS & etc)
  - "Inspired" by APT
    - Thematically and implementationally similar
    - Can be considered "cleaner" in use, though slower than APT
  - Commands are also largely similar to those in APT
    - e.g **yum install openssh-client** to install the openssh-client
  - However, important differences exist
    - Calling **apt update** will update the information cache of available software in the repositories
    - While calling **yum update** will update every package on the system(!)
    - The command in yum for refreshing the information cache is **yum check-update**

Higher vs. Lower package management

So, to summarise:

- Higher-level package management tools, such as **apt** and **yum**, rely on lower-level tools, such as **rpm** and **dpkg**, to handle the actual installation and querying of packages
- Instead, **apt** and **yum** focus on downloading, managing and updating packages on the system from a set of provided software repositories
- It is fully possible to make do without either apt or yum, only using dpkg and rpm. However, you as a user would then need to manually download any required packages, install these, resolve any potential dependency conflicts and version mismatches, and keep them updated afterwards.

End

Questions?