COMP09024 Unix System Administration

Lecture 7: Software Installation, Configuration and Management

Duncan Thomson/Hector Marco

UWS

Trimester 1 2020/21

Outline

7.1 The Components of Software

What is Software

'Software' is a wide term and covers a range of items:

- Executables, which may be:
  - Binaries (in 'machine code' for a particular CPU)
  - Scripts (written in a scripting language: `sh`, Perl or others
  - These may include command line programs, GUI-based applications and server or 'daemon' programs
- Libraries containing commonly used functions
- Data used by executables (eg word lists for spell checkers, brush shapes for image editors, etc)
- Documentation, including `man` pages
- Configuration and settings

Binaries and Executables

- All executable programs (binaries and scripts) must be executable by at least one user
- Binaries may be various formats - most common in modern Unix is ELF (Executable and Linking Format)
- Binaries include code and data from the program itself
- They may also include additional `library` code added at compile time from, or dependent on dynamically linked code stored elsewhere
- Scripts begin with the hash-bang (or 'shebang') `#!`, immediately followed by the full path to the interpreter, eg `/bin/sh`, `/usr/bin/awk`, `/usr/bin/python`

Executables Directories

- Executables are usually stored in `bin` directories
- These may be found in various places:
  - /usr/local/sbin (local applications)
  - /usr/local/bin (local applications)
  - /usr/sbin (sysadmin executables)
  - /usr/bin
  - /sbin (sysadmin executables)
  - /bin
- The `PATH` environment variable specifies the search path for executables:

```
user@debian~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

- `which` shows where in the path an executable is found

```
user@debian~$ which ping
/bin/ping
```

- When we execute `ping` in a shell, actually is `/bin/ping` the command executed!

Libraries

- Libraries are collections of commonly used routines
- Library code can be linked (statically) at build time
- Much more common is dynamically linking code:
  - Compiler notes which libraries (and versions) are required
  - When run, this code is loaded dynamically
  - Multiple programs can share a single copy of the library
- The dynamic linker (`ld.so`) loads libraries at run time
- `ldd` shows a list of what libraries a program requires
- The `LD_LIBRARY_PATH` environment variable specifies a list of directories to search for dynamic libraries
- Libraries are stored in `/lib`, `/usr/lib`, `/usr/local/lib` (and possibly others)
- Libraries can also be found in subdirectories of these for particular architectures and operating systems

Documentation: `man` Pages

- `/usr/share/man` or `/usr/man` hold `man` pages
- This contains subdirectories for sections of the manual:
  1. Executable programms and shell commands in `man1`
  2. System calls (functions provided by the kernel) in `man2`
  3. Library calls (functions within program libraries) in `man3`
  4. Special files (usually found in /dev) in `man4`
  5. File formats and conventions (eg /etc/passwd) in `man5`
  6. Games in `man6`
  7. Macro packages and miscellaneous conventions in `man7`
  8. System administration commands (usually only for root) in `man8`
  9. Kernel routines in `man9` (on Linux only)
- There may also be subdirectories for `man` pages in other languages (based on local names, eg `es`, `de`, `sv`, `pt_BR`)
- `whereis` locates both executables and `man` pages

Documentation: Formatting `man` Pages

- `man` pages sources are stored in a special format using `man` macros for the `nroff` text processor
- File suffix represents manual section
- Files may be compressed (eg with `gzip`)
- The `man` command then formats these for display (and paging) as appropriate
- These can also be formatted in other ways using `nroff` (or `groff`) with the `man` macro package
- For example, to format the `bash(1)` manual page in PostScript:

```
zcat /usr/share/man/man1/bash.1.gz |       \\
                groff -m man -T ps >bash.ps
```

- Preformatted `man` pages may be cached in `/var/cache/man/`

Other Documentation

- Other documentation may take various forms
- `info` is a hyperlinked documentation system:
  - `info` pages are stored in `.info` files (possibly compressed) in `/usr/share/info`
  - These can be navigated with the `info` command
  - Widely used by GNU software (alongside `man` pages, usually)
- Configuration files may contain comments and example configurations
- Printable documentation and example configurations may be stored in subdirectories of `/usr/share/doc/`

Configuration Files

- Unix configuration files are generally text-based
- Configuration files are stored either directly in `/etc` or in a subdirectory
- Configuration for software may in complex cases be split across multiple files, by either:
  - Naming files numerically (they are read in order)
  - Including files from a master config file
- A common convention for multiple configuration files stored in a directory is to suffix the directory name with a `.d`
- Configuration files are normally well commented — `#` is a common comment character, though not used in all cases
- Other ‘configuration’ files may include scripts for starting and stopping services

7.2 Compiling Software from Source

Software Sources

- Software can often be built from source
- This allows one version of the software bundle to be distributed, usable on multiple platforms
- But this requires compilation of binaries and correct installation by the system administrator
- Such bundles generally are in a single ‘archive’ file, usually compressed
- The steps to install are:
  1. Unpack the sources
  2. Configure appropriately for the target system
  3. Compile sources into executable code and libraries
  4. Install software into the appropriate directories

Unpacking Sources

- The most common format for distributed sources is a compressed `tar` archive (with a `.tar.gz` or `.tgz` suffix)
- This can be uncompressed and unpacked in separate steps, but more usual is using (for example):

```
tar xzf software-1.2.3.tgz
```

- `z` indicates that the archive has been compressed with `gzip` —use `j` for bzip, and `Z` for `compress`
- This normally creates a subdirectory called (eg) `software-1.2.3`
- This directory will normally contain (among other things):
  - Documentation files such as `README` and/or `INSTALL`
  - Subdirectories for source code, documentation, etc

Build Configuration and `./configure`

- Source code is designed to be built on many different types of Unix system (and sometimes even non-Unix systems)
- The next step is to configure the build system to use the appropriate system calls and library routines from the target system
- Usually, the autoconfigure system can determine such settings automatically
- This involves running the `./configure` script
- Other manual settings may also be specified at this stage, for example:

```
./configure -{}-prefix=/usr/local
```

- The output of this stage is a file called `Makefile`

Compiling

- Once the build environment is complete, the next stage is compiling
- Scripts (such as Perl, Python), web applications, or software which compiles into byte code for a virtual machine (such as Java or .NET programs) may not require this step
- Software in C (or C++) will require compiling however
- The C compiler may be called `cc` or `gcc`
- However the `make` command (on its own) will normally perform this step without manual intervention
- Other steps may be required to build libraries, documentation, or other components

Installing

- Finally the software can be installed to the appropriate location(s) using (usually) `make install`
- This may use the special `install` command, which copies a file (like `cp`), but also allows:
  - Specifying file owner and group owner
  - Specifying permissions
  - Stripping unrequired symbols
- This is normally the **only** step where `root` privileges are required

The make Command

- The build system uses a piece of software called make
- Uses Makefile to control building a piece of software
- The Makefile contains a list of 'targets' (the first in the file is the default) which can be given as arguments to make
- After each target and colon, dependencies are listed — if dependencies have more recent timestamp than target, the target will be rebuilt
- After each target, an indented list of commands provides a 'recipe' or set of build instructions for that target
- Makefiles may use both variables and various wildcards to make the system easier to use
- make can be used for all kinds of project (eg these slides)

An Example Makefile

```
# a variable specifying the install directory
INSTDIR=/usr/local/bin

# first (default) target is hello
# depends on hello.c, build with gcc command shown
hello: hello.c
        gcc -o hello hello.c

# 'install' target: depends on 'hello', and will
# build 'hello' first if 'hello.c' has changed
install: hello
        install -s -o root hello $(INSTDIR)
```

7.3 Packaging Systems

Binary Packaging

- While it's possible to compile an entire Unix system from source (eg LFS), this is difficult and slow
- Usually the OS can be installed from *binary* packages, which contain ready to use executables, libraries, etc
- There are a range of different package formats, some of which you may already have come across:
  - `.deb` packages are used by Debian and derivatives (eg Ubuntu and the `ipkg` format for embedded devices)
  - `.rpm` packages are used by RedHat and derivatives
  - `.pkg` packages are used by Solaris
  - `.apk` packages contains apps for the Android OS
- Such systems are designed to:
  - Note and resolve dependencies on other software
  - Preserve configuration from previous versions
  - Allow easy installation of all elements of software

Debian Packaging

- Based around the `.deb` file format:
- Includes dependency information of each package
- Allows for automatic or user-driven basic configuration, and labels configuration files to avoid overwriting or removal
- Allows pre/post install/removal scripts
- Allows cryptographic verification of packages
- The low-level package management tool is `dpkg`
- The APT (Advanced Packaging Tool) system is used
- Provides development tools for production of packages
- Used by the Debian distribution, but also derivatives such as Ubuntu, Linux Mint, and Knoppix
- The basis of a number of other package formats including `ipkg` (for embedded Unix) and `wpkg` (for Windows)

RedHat Packaging

- Uses the `.rpm` file format
- Includes dependency information in each package
- Allows scripts to be run pre/post installation
- Allows cryptographic verification of packages
- Low-level package management tool is `rpm`
- Higher level tools such as `yum` provide easier interaction with the system
- Developers make use of `SPEC` files to build packages
- Used by the RedHat distribution, and associated distributions such as Fedora, Novell, openSuSE

7.4 Debian Package Management

The Package Repository

- Debian packages can be split into sets depending on:
  - Debian release
  - Repository
- Releases are the version of the distribution:
  - This could be names such as `stable`, `testing` ...
  - ... or codenames such as `jessie`, `buster`
  - Ubuntu uses other codenames: `saucy`, `trusty` etc
- Repositories classify software in other ways:
  - Debian — 'freeness': `main`, `contrib`, `non-free`
  - Ubuntu — support/licence: `main`, `universe`, ...

.deb Files

- .deb consists of two sets of files:
  - Control information (lists of files, config files, scripts)
  - Installation files (executables, config, libraries, docs)

- Other index files in the Debian archive also hold other information about each package — eg version, dependencies, description, size, and so on

The dpkg Command

- dpkg is the low level command for managing Debian packages
- It can take various flags to control its operation
- Previously a wide range of these were used, but most useful on modern systems are:
  - -i Install package
  - -l lists installed packages
  - -r removes an installed package
  - -P purges an installed package (removes all files, including configuration files)
  - --configure configures a package
  - -S searches for a filename in installed packages
  - -L lists files in a package
- dpkg cannot:
  - Find or download software over the network
  - Resolve dependencies automatically

APT Commands

- APT was developed to improve manageability of packages
- It automatically resolves dependencies, and can download packages from a network repository
- `apt-get` updates, installs and removes packages:
  - `update` updates list of packages available
  - `upgrade` upgrades current packages
  - `dist-upgrade` upgrades to next release
  - `install` installs a package (and dependencies)
  - `remove` removes a package
  - `purge` removes a package and configuration
  - `autoremove` removes dependencies of old packages
- `apt-cache` accesses package lists and repositories:
  - `show` shows package details
  - `search` search for packages

Aptitude and Synaptic

- The `aptitude` command provides an alternative way to access some `apt-get` and `apt-cache` functionality
  - Subcommands include `update`, `safe-upgrade`, `dist-upgrade`, `install`, `remove`
- However used without arguments it provides a text-based GUI to APT
- Synaptic provides a full GUI-based package manager based on APT

Configuring APT

- Two main types of configuration are required for APT
- `/etc/apt/sources.list` lists the package repositories used, containing lines such as:

```
deb http://deb.debian.org/debian buster main
deb-src http://deb.debian.org/debian buster main
deb http://deb.debian.org/debian-security/ buster/updates main
deb-src http://deb.debian.org/debian-security/ buster/updates main
```

- `/etc/apt` also contains other configuration options for APT, which can be either in `/etc/apt/apt.conf` or files in subdirectories
- Documented in `apt.conf(5)` and elsewhere
- Example configuration:

```
Acquire::http::Proxy "http://146.191.228.22:9090/";
```

7.5 Security Updates

Vulnerabilities

- Software bugs are present in all systems
- Bugs which introduce a security weakness in software are known as vulnerabilities
- Vulnerabilities may be classified in various ways:
  - How the vulnerability has come about (buffer overflow; cross-site scripting; unvalidated input; injection; etc)
  - Consequences of the vulnerability (privilege escalation; information leakage; denial of service; code execution; etc)
  - Where the vulnerability can be exploited (remote, local)
- Some vulnerabilities (along with fixes or workarounds) are normally disclosed as soon as possible

Disclosure

- Disclosure is the process of publicising vulnerabilities
- Normally disclosure announcements include:
  - A brief description of the vulnerability
  - An idea of the seriousness of the vulnerability
  - Details on a workaround or applying a software update
- There are various fora for finding disclosures:
  - Industry wide databases/lists such as:
    - Bugtraq (http://www.securityfocus.com/archive/1)
    - Common Vulnerabilities and Exposures (http://cve.mitre.org/)
    - National Vulnerability Database (http://nvd.nist.gov/)
  - Per OS or distribution lists (eg Debian, Solaris)
  - Per-software product lists/pages (eg Apache, Linux, kernel)
- Best practice is subscribing to mailing lists and web sites appropriate for the software you manage

Updating Software

- The aim of tracking vulnerabilities and disclosures is to secure vulnerable systems
- This normally requires updating software
- Most organisations will have a policy on software updates (and security updates in particular)
- Security updates should be applied as soon as possible after disclosure is made
- On Debian, the following two commands (one after the other) apply updates to all installed packages

```
aptitude update
aptitude upgrade
```

- It is essential to do this on a regular basis

Summary

- Binaries, executables and (dynamic) libraries
- Documentation and `man` pages
- Configuration files
- Installing software from source code
- Configuration, building and installing
- `make` and `Makefile`
- Binary packages: `.deb` and `.rpm`
- Debian packages and `dpkg`
- APT commands and configuration
- Software vulnerabilities and disclosure
- Keeping up with disclosures and applying updates