LESSON – 15-2

MANAGING SOFTWARE PACKAGES

LEARNING OUTCOMES

- Manage software packages with apt
- Understanding apt Filenames
- querying apt Database
- Understand repositories

Manage software packages with apt

- apt (Advanced Package Tool) is a command-line utility used for managing software packages
- Linux distributions, such as Debian, Ubuntu, and Linux Mint.
- Simplifies the process of installing, updating, and removing software.
- Example:
  - apt install packagename
  - apt-get install packagename

Understanding .deb Filenames

- apt works with .deb package files. The filename format for a .deb package is:

  package-name_version-revision_architecture.deb

- **package-name**: The name of the software (e.g., vlc, apache2).
- **version:** The upstream version number of the software (e.g., 3.0.11, 2.4.41).
- **revision:** The Debian-specific version for a package built from the same upstream version. It's used to track changes or fixes to the package itself without changing the software's version (e.g., 1build1, 5ubuntu1).
- **architecture:** The system architecture the package is built for (e.g., amd64, i386, arm64). The all architecture is used for packages that are not specific to any hardware, such as shell scripts or configuration files.
- **deb:** The file extension for a Debian package.

Manage software packages with apt

- For example, a filename like

  vlc_3.0.11-1_amd64.deb

- package-name: vlc
- version: 3.0.11
- revision: 1
- architecture: amd64

Querying the ***apt*** Database

- The apt database stores information about all the packages available in your configured repositories and those installed on your system.
- The ***apt*** command is a high-level tool for package management, and it's what most users will interact with.
- Search for a package: Use *apt search <package_name>*
  - **apt search vlc** -> will list all packages whose name or description contains "vlc".

Querying the ***apt*** Database

- Show package information: Use apt show <package_name>
  - **apt show vlc** -> provides a comprehensive overview of the package (description, dependencies, size etc)
- List installed packages: Use apt list --installed to see all packages currently on your system.
  - **apt list --installed | grep 'apache2'** -> useful for checking if a specific package is installed.

dpkg Command

- ***dpkg*** is a lower-level tool that directly manages installed packages on the system.
- apt uses dpkg behind the scenes.
- Query an installed package: *dpkg -s <package_name>*

  dpkg -s apache2 -> will show if the package is installed and its current status.
- List files of an installed package: *dpkg -L <package_name>*

  dpkg -L apache2 -> is helpful for locating files installed by the package.

dpkg Command

- Find which package owns a file: *dpkg -S <file_path>*

  dpkg -S /etc/apache2/apache2.conf -> will return the name of the package that owns this configuration file.

Querying a *.deb* Package File

You can inspect a .deb file that is not yet installed on your system using the dpkg command.

- Get information from a .deb file: *dpkg -I <package_file.deb>*

dpkg -I vlc_3.0.11-1_amd64.deb -> shows the package name, version, architecture, dependencies, and a brief description.

- List contents of a .deb file: *dpkg -c <package_file.deb>*

dpkg -c vlc_3.0.11-1_amd64.deb -> useful for inspecting the contents of the package before installation.

Understanding ***apt*** Repositories

- An apt repository is a centralized location where software packages and their metadata are stored.

- Think of it as a digital library for software. When you use apt to install, update, or remove packages, it retrieves information and files from these repositories.

- Key components of a repository:
  - Packages: The .deb files themselves.
  - Metadata: Information about the packages, including their names, versions, dependencies, and descriptions. This metadata is organized into **Packages.gz** and **Release files**.

Specifying Which Repositories to Use

- The list of repositories your system uses is located in the /etc/apt/sources.list file. You can also add individual repository files in the /etc/apt/sources.list.d/ directory.
- Each line in sources.list follows a specific format:

```
deb [arch=amd64] http://archive.ubuntu.com/ubuntu/ focal main restricted universe
multiverse
```

- You can edit these files with a text editor or, more commonly, use a command-line tool like add-apt-repository to add a new repository easily.

Creating Your Own Repository

- Allows you to host and manage custom packages. The process involves a few key steps:
  - Host the Repository: Set up a web server (like Apache or Nginx) to serve the repository files.
  - Organize Packages: Place your .deb files into a directory structure on the server.
  - Create Metadata: Use a tool like dpkg-scanpackages to generate the Packages file from your .deb files. This file contains the metadata apt needs to index your packages. You then compress this file into Packages.gz.
  - Sign the Repository: Create a GPG key and use it to sign the repository's metadata. This ensures the integrity and authenticity of the packages for users.
  - Add to Client's System: Instruct users to add your repository's URL and public GPG key to their apt configuration files.

Using ***apt***

- apt is the primary command for managing packages.

Most common commands:

- Update Repository Metadata: ***sudo apt update***
  - downloads the latest package lists from all configured repositories. It doesn't install or upgrade packages, it only updates the cache of available packages.
- Install a Package: ***sudo apt install <package_name>***
  - downloads and installs the specified package and its dependencies.
- Upgrade All Packages: ***sudo apt upgrade***
  - upgrades all installed packages to their latest versions, but it will not remove any packages or install new ones to satisfy dependencies.

Using ***apt***

- Full System Upgrade: ***sudo apt full-upgrade***
  - more aggressive upgrade that will also remove old packages or install new ones if needed to resolve dependency issues.

Updating Packages

- The process of updating packages is a two-step process:
  - Update the cache: ***sudo apt update***
    to refresh the local list of available packages from the repositories.
  - Upgrade the packages: ***sudo apt upgrade***
    to install the newer versions of the packages that are already on your system.

Upgrading Ubuntu OS

- Upgrade a Ubuntu system from one major release to the next

  do-release-upgrade

- For LTS (Long-Term Support) releases, do-release-upgrade will only offer to upgrade to the next LTS version by default.

- To upgrade from one LTS release to a non-LTS release or to a newer LTS, you may need to use the ***-d*** or ***-p*** flags.

Removing Packages

- Packages are removed using the apt package manager.
- Three primary commands, each with a slightly different function:

  remove, ***purge,*** and ***autoremove.***

Removing Packages

- ***apt remove***: removes the package's binary files but leaves its configuration files behind.
- This is useful if you think you might reinstall the package later and want to keep your settings.

```
sudo apt remove <package_name>
```

Example: ***sudo apt remove apache2***

- Removes the Apache web server but keeps its configuration files (e.g., in /etc/apache2).

Removing Packages

- ***apt purge***: removes the package's binary files and also purges its configuration files. This is the best option for completely deleting a package and all of its associated settings.

- This is useful if you think you might reinstall the package later and want to keep your settings.

sudo apt purge <package_name>

Example: ***sudo apt purge apache2***

- This command removes the Apache web server and all its configuration files.

Removing Packages

- ***apt autoremove***: removes packages that were automatically installed to satisfy dependencies for other packages but are no longer needed.
- useful to free up disk space by cleaning up unnecessary dependencies.

```
sudo apt autoremove
```