Package Managers

Presentation by Harsh Khandeparkar

Two Perspectives

Package Consumer

Search, Install, Remove packages

Package Maintainer

Create and Distribute packages
(for pacman)

What is a package manager?

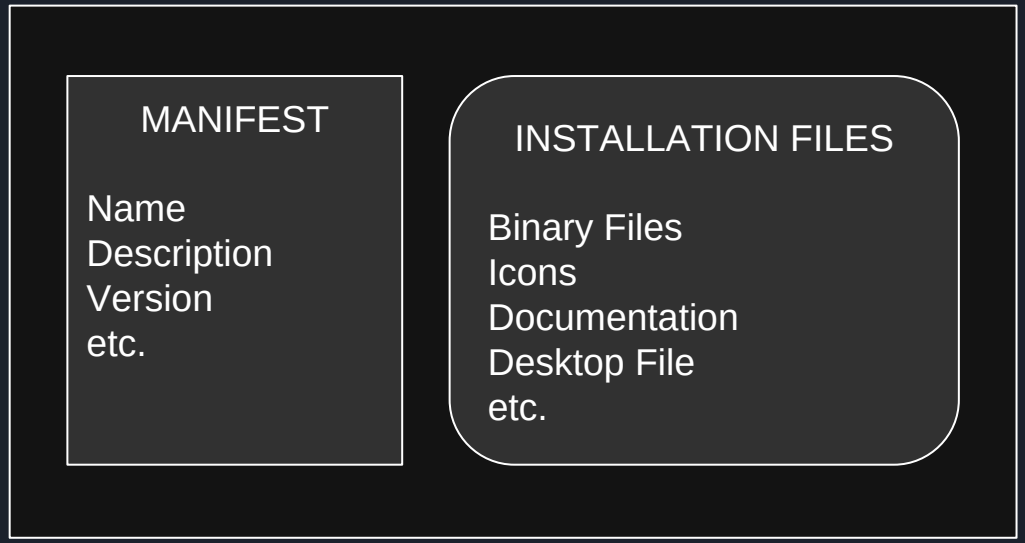
Wikipedia Definition: A *package manager* or *package-management system* is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs [packages] for a computer in a consistent manner.

What is a Package?

A package format is a type of distributable archive containing software, tools or other assets and additional metadata needed.

Typically contains a manifest file (which contains metadata about the package) and executable files, source code, or any other files which will be installed on the computer.

Package


MANIFEST
Name
Description
Version
etc.
INSTALLATION FILES
Binary Files
Icons
Documentation
Desktop File
etc.

Package Consumer Perspective

Basic Features

1. **Searching Packages:** Package managers store a local database of packages. This information can be searched using commands or through a graphical interface.

2. **Installing Packages:** A package or a list of packages from the local database can be installed with a simple command or using a graphical interface.

3. **Upgrading:** All the installed packages or a list of packages can be upgraded using a single command.

4. **Removing:** Installed packages can also be removed.

Examples of package managers

Package Manager	Distro	Packaging Format
apt	Debian and Debian Based eg: Ubuntu	.deb
dnf	Fedora	.rpm
yum	Red Hat Enterprise Linux	.rpm
pacman	Arch Linux	.tar.zst
flatpak	Cross-platform (Installed OOTB on Pop!_OS, Linux Mint, Elementary OS)	.flatpak
snap	Cross-platform (Installed OOTB on Ubuntu)	.snap

Differences

The primary differences between package managers are

1. The packaging format.
2. The CLI/GUI interfaces.
3. The list of packages available in the repositories.

Command Examples

Command	apt	dnf	yum	pacman	flatpak	snap
Installing	apt install [name]	dnf install [name]	yum install [name]	pacman -S [name]	flatpak install [name]	snap install [name]
Searching	apt search [query]	dnf search [query]	yum search [query]	pacman -Ss [query]	flatpak search [query]	snap find [query]
Upgrading	apt upgrade	dnf upgrade	yum update	pacman -Syu	flatpak update	snap refresh
Removing	apt remove [name]	dnf remove [name]	yum remove [name]	pacman -R [name]	flatpak uninstall [name]	snap remove [name]

Pacman

Pacman (stands for *pac*kage *mana*ger) is a core component of Arch Linux. It uses a simple binary packaging format with *zstd* compression.

Pacman can be configured by editing the `/etc/pacman.conf` file. Options such as number of parallel downloads, terminal colors, etc. can be configured.

Repositories are collections of packages which are stored on one or more server called “mirrors”. Pacman syncs the local repository database with a mirror on running the `pacman -Sy` command.

The list of mirrors used by pacman is defined in `/etc/pacman.d/mirrorlist`.

Further Reading: `pacman.conf(5)` (man page)

Pacman Repositories

There are three main official repositories:

- core: Contains essential packages such as those required for booting and configuring Arch linux, and their dependencies.
- extra: Contains non-essential but widely used packages such as display drivers, window managers, and other applications.
- community: Contains packages that have been adopted by “Trusted Users” from the Arch User Repository (AUR). These packages may be eventually moved to core or extra.

Arch User Repository (AUR) is a community-driven repository but instead of containing built packages, AUR contains PKGBDUILDs only. Any user may contribute to the AUR.

AUR Web Interface: https://aur.archlinux.org

Package Maintainer Perspective

Building a Package for Pacman

The pacman package also includes tools for building and distributing packages.

A PKGBUILD file, created by the maintainer of the package, contains all the information necessary to *create* a package that can later be distributed.

The script makepkg is used to build a package from a given PKGBUILD file. While the PKGBUILD file might build the package from source code, the output package contains distributable binary/executable files.

Sample AUR Package: visual-studio-code-bin

PKGBUILD

A PKGBUILD file is a BASH script containing certain variables and functions based on which makepkg does the following:

1. Checks if dependencies required by the package are installed.
2. Retrieves the source file(s) from the specified source(s).
3. Unpacks and builds the distributable files. eg: Compiles source code into binary executables.
4. Generates the package file. This file can be installed with the pacman -U command.

Man page: makepkg(8)

PKGBUILD Variables (* = mandatory)

- `pkgname`*: Name of the package.
- `pkgdesc`: Description.
- `pkgver`*: Version
- `pkgrel`*: Release number. Differentiates between two builds of the same version of the software.
- `arch`*: Architectures on which the packages is intended to build and work. eg: `x86_64` (official supported by Arch), `arm` etc.
- `license`: Not mandatory but recommended to include with packages distributed publicly.
- `url`: Link to the original software website/source code.
- `depends`: Dependencies of the package.
- `makedepends`: Dependencies only required to make (build) the package.
- `source`: List of files needed to build the package. Can be a URL to a file or a git repository. `makepkg` won't proceed if some of the files were not retrieved successfully.
- `sha256sums` or `sha384sums` or `sha512sums`: Array of SHA checksums corresponding to the sources for integrity checks.

More variables: https://wiki.archlinux.org/title/PKGBUILD

PKGBUILD Functions

- `prepare()`: Used to prepare the source files for building. Tasks such as extracting archives can be done in this function. (tarball and zip sources are automatically extracted)
- `build()`: Used to build the final files which will be installed. eg: Compiling source code into executable binaries.
- `package()`: Used to put the compiled files into the package directory from where `makepkg` creates the package.

More Functions: https://wiki.archlinux.org/title/Creating_packages

Further Reading

Publishing packages to AUR: https://wiki.archlinux.org/title/AUR_submission_guidelines

Additional guidelines for VCS packages: https://wiki.archlinux.org/title/VCS_package_guidelines

namcap (used for testing packages): https://wiki.archlinux.org/title/Namcap

Credits

- Visual Studio Code (Sample AUR package): https://aur.archlinux.org/packages/visual-studio-code-bin
- Mygame (Sample code used for making a PKGBUILD): https://github.com/Cooldude069/Mygame
- Google Slides Template: “Focus”

References

- https://en.wikipedia.org/wiki/Package_manager
- https://en.wikipedia.org/wiki/List_of_software_package_management_systems#Linux
- https://wiki.archlinux.org/title/pacman
- https://wiki.archlinux.org/title/Official_repositories
- https://wiki.archlinux.org/title/Arch_User_Repository
- https://wiki.archlinux.org/title/Arch_Build_System
- https://wiki.archlinux.org/title/Creating_packages
- https://wiki.archlinux.org/title/PKGBUILD
- https://wiki.archlinux.org/title/VCS_package_guidelines
- https://man.archlinux.org/man/core/pacman/pacman.conf.5.en
- https://linuxconfig.org/comparison-of-major-linux-package-management-systems
- https://www.linuxfordevices.com/tutorials/centos/dnf-command
- https://computingforgeeks.com/install-and-manage-flatpak-applications-on-linux/
- https://docs.flatpak.org/en/latest/flatpak-command-reference.html#flatpak-update
- https://access.redhat.com/articles/yum-cheat-sheet
- https://www.omgubuntu.co.uk/2016/12/simple-guide-snapd-commands