TASK

Additional Reading - Installation

Visit our website

INTRODUCTION

If you would like to install additional third-party packages, you should be aware of best practices relating to installation. The use of a package manager is highly recommended, and containerisation is important when different projects or applications require specific requirements.

PACKAGE MANAGERS

Package managers provide a means to install packages, organise them in your file system, and manage dependencies between packages. A dependency is when a package depends on another particular version of another package to function. If you have two different packages that rely on different versions of another package, this creates problems if you don't have a package manager. It can also be very time-consuming to find and install all the dependencies for a package. Package managers also check for known vulnerabilities that may pose a security risk. You can use a package manager to share packages you have created with others.

There are two types of package managers: those for **operating systems** and those for **programming languages**. Package managers are linked to software repositories where the packages or software are stored.

Some common operating system package managers include:

- **Chocolatey** for Windows (**Chocolatey repository**)
- **HomeBrew** for MacOS (**Homebrew repository**)
- Linux – each distribution has its own package manager
    - **DNF** for Fedora
    - **APT** for Ubuntu

As mentioned, programming languages typically have language-specific package managers. Examples of some programming languages and commonly used package managers are summarised in the table below.

Language	Package Manager	Software Repository
Python	`pip`	**PyPi** (Python Package Index)
Python	`conda`	**Anaconda Packages**
Java	`Maven`	**Maven Central**
Java	`Gradle`	**Maven Central**
JavaScript	`npm`	**Node Package Manager**
Javascript	`yarn`	**Yarn Registry**

Python

pip is the general-purpose package manager for Python. It is used to install, upgrade, and manage Python packages from the PyPI repository. On the other hand, **conda** is more specialised towards data science programming, providing easy installation of both Python packages and other data science tools. This makes it an ideal choice for data-related tasks, scientific computing, machine learning, and other data-intensive projects.

Java

Maven is the primary package manager for Java projects. Among its vast collection of libraries and plugins, it excels in managing project dependencies, build automation, and project documentation. For projects that prioritise a build tool with flexibility and performance, **Gradle** is a good alternative. It offers powerful build customisation options and enjoys widespread usage in app development.

JavaScript

npm is the de facto package manager for JavaScript and Node.js projects, providing a vast array of packages, libraries, and tools for frontend and backend development. However, for projects that emphasise faster and more reliable package installation, **yarn** is a good alternative. Built on top of npm, yarn is known for its enhanced performance, parallel installations, and caching capabilities.

The package manager you use depends on your preferences and the project's specific needs. Some key considerations to keep in mind are:

- **Project requirements:** ensure that the package manager you choose supports the necessary libraries and dependencies.

- **Community support:** look for an active and supportive community to provide resources and troubleshooting assistance.
- **Performance and reliability:** assess performance factors like speed and consistency vs comprehensive package registry management, especially for large complex projects.
- **Compatibility:** consider compatibility to ensure that it integrates seamlessly with your tools and workflows.
- **Security:** research the security measures and best practices followed by the package manager to safeguard your project from potential vulnerabilities.
- **Package versions:** check whether the package manager is frequently updated to ensure your project benefits from bug fixes and new features.
- **Documentation:** always review the documentation and ease of use for a smooth development process.

CONTAINERISATION

As you start working on software applications and larger projects that need to be deployed, you will also need to consider how to bundle your code with all the third-party packages it relies on.

Docker is a widely adopted and versatile method of containerisation that supports multiple programming languages, making it a popular choice among developers across various technology stacks. It enables you to create lightweight and portable containers that encapsulate your application, simplifying the deployment and management process across different environments.

Python

For Python projects, the `venv` module is a well-known containerisation method. The **venv module** allows you to create isolated virtual environments for Python projects, each with its own Python interpreter and package dependencies. Additionally, Anaconda allows you to create specialised conda **virtual environments**, which is particularly useful for Data Science projects, to package and manage your projects efficiently.

JavaScript

The primary containerisation method for JavaScript applications is the Node.js `npm` module, which seamlessly integrates with Docker containers. Additionally, the `nvm`

Node Version Manager is another popular way to install, upgrade, and manage different Node.js and `npm` releases on Unix, macOS, and Windows WSL platforms.

THE COMMAND LINE

To install the packages you will need to become familiar with the command line.

The command line is a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text. With the command line, you can quickly issue instructions to your computer, getting it to do precisely what you want it to do. The command line is rarely used by most end users since the advent of the Graphical User Interface (a more visual way of interacting with a computer using items such as windows, icons, menus, etc.). However, you will find it helpful to use the command line when installing third-party packages and other actions such as version control with Git.

Take note:

You will need to ensure that you have **administrator access** on Windows or **superuser access** on macOS, Fedora or Ubuntu to install packages.

Finding The Command Line Shell

On Windows, you can easily access the command line interface by clicking the Start menu and typing `'powershell'` (for Windows PowerShell) or `'cmd'` (for Windows Command Prompt) in the search box. Alternatively, you can find the command line interface under 'Programs' and open it by clicking the application.

With macOS, you can access the command line interface by opening the Applications folder, navigating to Utilities and then launching Terminal. Alternatively, you can search for `'terminal'` to find the application to launch.

Common Commands

The following table provides a selection of commonly used PowerShell and Unix commands to help get you started with the command line.

For a comprehensive list of commands, visit **Powershell commands**, **Command Prompt commands**, and **Unix commands** to explore more command line operations.

Description	Windows cmd	Windows Powershell (alias)	Mac OS/Unix
Displays the current working directory	`chdir`	`pwd`	`pwd`
Changes the directory	`cd`	`cd`	`cd`
Move up one level in the directory	`cd..`	`cd..`	`cd..`
Displays a list of a directories files and subfolders	`dir`	`dir`	`ls`
Print contents of a text file	`type`	`type`	`cat`
Create a new directory	`mkdir`	`mkdir`	`mkdir`
Remove files and directories	`del / rmdir`	`del / rmdir`	`rm`
Move or rename files and directories	`move / ren / rd`	`move / ren`	`mv`
Copy files and directories	`copy`	`copy`	`cp`
Clear the screen	`cls`	`cls`	`clear`
Quit the terminal	`exit`	`exit`	`q`

Code Hack

You can get detailed information, including instructions, examples and usage guidelines, on the various commands available in both Windows Powershell and Unix-based systems.

For **Windows Powershell**, open the terminal and run `Get-Help`. To get help on a specific command, run `Get-Help <cmd-name>`.

For **Windows Command Prompt**, open the terminal and run `help`. To get help on a specific command, run `help <cmd-name>`.

For **Unix-based systems**, open the terminal and run `man` to access the manual pages. To get help on a specific command, run `man <cmd-name>`. Additionally, you can also run the `whatis <cmd-name>` to get a brief description of the specified command.