**Day 7 Task: Understanding package manager and systemctl**

**What is a package manager in Linux?**

In simpler words, a package manager is a tool that allows users to install, remove, upgrade, configure and manage software packages on an operating system. The package manager can be a graphical application like a software center or a command line tool like apt-get or pacman.

You’ll often find me using the term ‘package’ in tutorials and articles, To understand package manager, you must understand what a package is.

**What is a package?**

A package is usually referred to an application but it could be a GUI application, command line tool or a software library (required by other software programs). A package is essentially an archive file containing the binary executable, configuration file and sometimes information about the dependencies.

**Different kinds of package managers**

Package Managers differ based on packaging system but same packaging system may have more than one package manager.

For example, RPM has Yum and DNF package managers. For DEB, you have apt-get, aptitude command line based package managers.

## Tasks

1. You have to install docker and jenkins in your system from your terminal using package managers

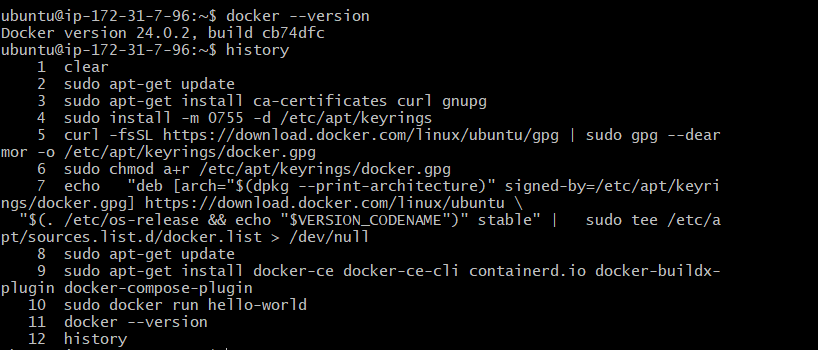
### Install using the apt repository

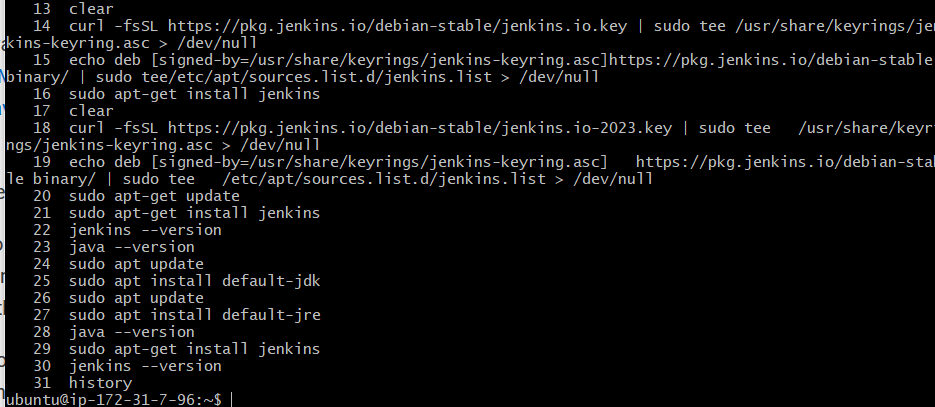
Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

#### **Set up the repository**

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:
2. $ sudo apt-get update
3. $ sudo apt-get install ca-certificates curl gnupg
4. Add Docker’s official GPG key:
5. $ sudo install -m 0755 -d /etc/apt/keyrings
6. $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
7. $ sudo chmod a+r /etc/apt/keyrings/docker.gpg
8. Use the following command to set up the repository:
9. $ echo \
10. "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
11. "$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null





**Task 2:** Systemctl vs Service

-       systemctl is basically a more powerful version of the service.

-       With service, you can only do commands related to the service (i.e. status, reload, restart) whereas with systemctl you can use more advanced commands such as:

“systemctl is-failed name.service # check if service failed to load”

# Day 8 Task: Basic Git & GitHub for DevOps Engineers.

## What is Git?

Git is a version control system that allows you to track changes to files and coordinate work on those files among multiple people. It is commonly used for software development, but it can be used to track changes to any set of files.

With Git, you can keep a record of who made changes to what part of a file, and you can revert back to earlier versions of the file if needed. Git also makes it easy to collaborate with others, as you can share changes and merge the changes made by different people into a single version of a file.

## What is Github?

GitHub is a web-based platform that provides hosting for version control using Git. It is a subsidiary of Microsoft, and it offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. GitHub is a very popular platform for developers to share and collaborate on projects, and it is also used for hosting open-source projects.

## What is Version Control? How many types of version controls we have?

Version control is a system that tracks changes to a file or set of files over time so that you can recall specific versions later. It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

There are two main types of version control systems: centralized version control systems and distributed version control systems.

1. A centralized version control system (CVCS) uses a central server to store all the versions of a project's files. Developers "check out" files from the central server, make changes, and then "check in" the updated files. Examples of CVCS include Subversion and Perforce.
2. A distributed version control system (DVCS) allows developers to "clone" an entire repository, including the entire version history of the project. This means that they have a complete local copy of the repository, including all branches and past versions. Developers can work independently and then later merge their changes back into the main repository. Examples of DVCS include Git, Mercurial, and Darcs.

## Why we use distributed version control over centralized version control?

1. Better collaboration: In a DVCS, every developer has a full copy of the repository, including the entire history of all changes. This makes it easier for developers to work together, as they don't have to constantly communicate with a central server to commit their changes or to see the changes made by others.
2. Improved speed: Because developers have a local copy of the repository, they can commit their changes and perform other version control actions faster, as they don't have to communicate with a central server.
3. Greater flexibility: With a DVCS, developers can work offline and commit their changes later when they do have an internet connection. They can also choose to share their changes with only a subset of the team, rather than pushing all of their changes to a central server.
4. Enhanced security: In a DVCS, the repository history is stored on multiple servers and computers, which makes it more resistant to data loss. If the central server in a CVCS goes down or the repository becomes corrupted, it can be difficult to recover the lost data.

Overall, the decentralized nature of a DVCS allows for greater collaboration, flexibility, and security, making it a popular choice for many teams.

If you already have Git installed, you can get the latest development version via Git itself:

git clone <https://github.com/git/git>

* + Git init -> Initializing a new git repo
  + Git status-> Gives the status of the branch like main/master branch, committed changes
  + Git checkout -b dev -> comes out of master branch and create a branch dev branch

## HOW TO CONFIG GIT

1. **Set your name:** In the terminal, enter the following command to set your name

git config --global user.name "Your Name"

Replace **"Your Name"** with your actual name. This sets your name to be associated with your commits.

1. **Set your email address:** Enter the following command to set your email address

git config --global user.email "youremail@example.com"

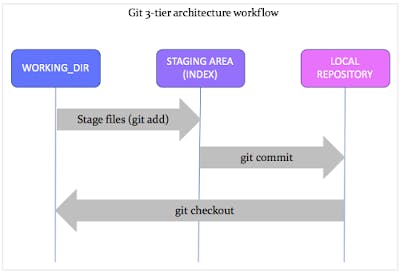
Replace "[youremail@example.com](mailto:youremail@example.com)" with your actual email address. This associates your email address with your commits.

git --version - To Check git is installed successfully

git config --list - To Check all the names and email

## What Three Stage Architecture

The three main stages or areas where changes to files are tracked and managed. These stages are also known as the Git workflow



**Working Directory**

* The working directory is where you make changes to your files
* It represents the current state of your project's files and directories on your local machine
* git init - Create an **Empty GIT Repository** in that directory and converts an existing directory into a **WORKING DIRECTORY**
* After making changes to files in the working directory,
* git add . - This Command Helps to add all the changes to the staging area.

COPY

COPY

COPY

COPY

COPY

COPY

COPY

COPY

git init

git add .

**Staging Area**

* The staging area is an intermediate stage where you prepare and organize changes before committing them to the Git repository or Local repository
* The staging area acts as a snapshot of the changes you want to include in the next commit to the local repository.
* git status - You can review the changes in the staging area

COPY

COPY

COPY

COPY

COPY

COPY

COPY

COPY

git status

**Local Repository**

* The repository where the changes are permanently recorded and stored.
* When you are satisfied with the changes in the staging area
* git commit -m "Your commit message" - commits represent a specific version where changes are permanently recorded and stored in the local repository
* Each Commit contains - a unique identifier [ Hash ID ] and A commit message

COPY

COPY

COPY

COPY

COPY

COPY

COPY

COPY

git commit -m "Your commit message"

1. What is Git and why is it important?

Git is a version control system where the developers work together and commit their changes without any change in the main line.

Importance:

* + 1. Distributed system where it can handle projects of any size from big to small.
    2. Open source software where developers can use without paying money.
    3. Provides cheap branching ideas, convenient staging and multiple workflows.
    4. Several people modifications to be merged into a single source.

2.What is difference Between Main Branch and Master Branch??

Main Branch: There is no much difference between main and master branch.

Main Branch is first branch made when the initialize a GIT repo, when we create the commit git identifies the snapshot of files.

Master Branch: After cloning the project from the remote server to the local repository, the resulting local repo have a default branch of master.

1. Can you explain the difference between Git and GitHub?
2. Git: Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.
3. GitHub: GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

To switch to another branch:

We use $git checkout <branch-name to enter>

### ****Creating a Branch**** and merging it

* + git branch <Branch name> - this command help to create a branch
  + git branch - this command help to see all the branch.

The current working branch will be in green \*

**CHANGING OR ADDING NEW FEATURES TO THE NEW BRANCH**

* + git checkout <branch name>

1. git checkout <branchname> - go to the main branch or master branch. to which branch u need to merge the branch
2. git merge <branchname> - the command will help u to merge the branch
   * If we make any changes in the sub branches first we have to commit those changes in the sub branch and merge those changes to the main branch
3. git merge <branch name> - command to merge the branch
4. git merge <branch name>

[**git-add[1]**](https://git-scm.com/docs/git-add)Add file contents to the index

[**git-branch[1]**](https://git-scm.com/docs/git-branch)List, create, or delete branches

[**git-checkout[1]**](https://git-scm.com/docs/git-checkout)Switch branches or restore working tree files