GETTING STARTED WITH PACKAGE MANAGEMENT

➢ SOFTWARE MANAGEMENT TOOLS:

- A software management system is a **collection of software tools** that automates the process of installing, upgrading, configuring, and removing computer programs.
- Each distribution of Linux has its own package management system.
- There are two software management tools in RHEL8.
  - **RPM:** Red Hat Package Manager
  - **YUM:** Yellow Dog Update Modifier

❖ RED HAT PACKAGE MANAGER (RPM):

- RPM is an **open packaging system,** which runs on **Red Hat** as well as **other Linux** and **UNIX systems.**
- You can use RPM to distribute, manage, and update software that you create for any of the operating systems like Red Hat Enterprise Linux, CentOS, and Fedora.
- Using RPM, we can **Installing, uninstalling, and upgrading packages.**
- RPM keeps the information of all the installed packages under **"/var/lib/rpm"** database.
- RPM packages typically have file names like **foo-1.0-1.i386.rpm.**
- The file name includes the **package name (foo), version (1.0), release (1),** and **architecture (i386).**

INSTALLING / UNINSTALLING / UPGRADING PACKAGES

SYNTAX: #rpm [options] package-name

-i	**:** Install a Package
-v	**:** Verbose Output
-h	**:** Shows Hash Programs
-U	**:** Upgrade a Package
-e	**:** Erase a Package
--force	**:** Installing Forcefully
--nodeps	**:** No dependencies

INSTALLING:

→ Installing Package:

```
#rpm -ivh rpm -ivh foo-1.0-1.i386.rpm
#rpm -ivh vsftpd*
```

→ Alternatively, the following command can also be used:

```
#rpm -Uvh foo-1.0-1.i386.rpm
```

UPGRADING:

→ Upgrading a package is similar to installing one. Type the following command at a shell prompt:

```
#rpm -Uvh foo-2.0-1.i386.rpm
```

CONFLICTING FILES:

→ To make RPM ignore this error, use the --replacefiles option:

```
#rpm -ivh --replacefiles foo-1.0-1.i386.rpm
```

UNRESOLVED DEPENDENCY:

RPM packages may sometimes depend on other packages, which means that they require other packages to be installed to run properly. If you try to install a package which has an unresolved dependency, output similar to the following is displayed:

```
error: Failed dependencies:
    bar.so.2 is needed by foo-1.0-1
Suggested resolutions: bar-2.0.20-3.i386.rpm
```

UNINSTALLING:

```
#rpm -e foo
error: Failed dependencies:
      foo is needed by (installed) bar-2.0.20-3.i386.rpm
```

To make RPM ignore this error and uninstall the package anyway (which may break the package dependent on it) use the **--nodeps** option.

FRESHENING:

- Freshening is similar to upgrading, except that only existent packages are upgraded. Type the following command at a shell prompt:

```
#rpm -Fvh foo-1.2-1.i386.rpm
```

→ Freshening works for single packages or package groups.

```
#rpm -Fvh *.rpm
```

QUERYING (WITH -q):

- The RPM database stores information about all RPM packages installed in your system. It is stored in the directory /var/lib/rpm/, and is used to query what packages are installed, what versions each package is, and any changes to any files in the package since installation, among others.
- You can also use the following Package Selection Options with -q to further refine or qualify your query:

SYNTAX: **#rpm [options] package-name**

-a	: All currently installed packages.
-c	: Displays a list of files marked as configuration files.
-d	: Displays a list of files marked as documentation.
-f	: The RPM database for which package owns.
-p	: The uninstalled package.
-i	: Displays package information.
-l	: Displays the list of files that the package contains.
-s	: Displays the state of all the files in the package.

→ To list installed packages / specific package:

```
#rpm -qa

#rpm -qa | grep -i foo

#rpm -q foo

#rpm -q vsftpd

#rpmquery vsftpd
```

→ To check configuration files:

```
#rpm -qc vsftpd
```

→ Information about a given package:

```
#rpm -qi vsftpd
```

→ To check installed files packagename:

```
#which useradd
#rpm -qf /usr/sbin/useradd
#which tree
#rpm -qf /usr/bin/tree
```

VERIFYING PACKAGE:

- It compares information about files installed from a package with the same information from the original package.
- Perhaps you have deleted some files by accident, but you are not sure what you deleted. To verify your entire system and see what might be missing:

```
#rpm  -Va
#rpm  -Va  vsftpd
```

→ To verify a package containing a particular file:

```
#rpm -Vf /usr/bin/foo
```

In this example, /usr/bin/foo is the absolute path to the file used to query a package.

→ To verify ALL installed packages throughout the system:

```
#rpm -Va
```

→ To verify an installed package against an RPM package file:

```
#rpm -Vp foo-1.0-1.i386.rpm
```

This command can be useful if you suspect that your RPM databases are corrupt.

❖ YELLOWDOG UPDATER MODIFIED (YUM):

- In RHEL 8, software installation is enabled by the new version of the **YUM tool (YUM v4),** which is based on the DNF (**Dandified YUM)** technology.
- It is a primary tool for installing, deleting, querying, and managing RedHat RPM software packages.
- YUM performs **automatic dependency resolution** on packages, it searches numerous **repositories** for packages and their dependencies.

YUM REPOSITORIES

- A YUM repository or repo is a storage location for holding and managing RPM Packages.
- REPOSITORIES Red Hat Enterprise Linux (RHEL) distributes content through different repositories.

BaseOS:

- It consists of the core set of the underlying operating system functionality that provides the foundation for all installations.
- This content is available in the RPM format and is subject to support terms similar to those in earlier releases of RHEL.

AppStream:

- AppStream Content in the AppStream repository includes additional user-space applications, runtime languages, and databases in support of the varied workloads and use cases.

NOTE: IMPORTANT Both the BaseOS and AppStream content sets are required by RHEL and are available in all RHEL subscriptions.

CodeReady Linux Builder:

It provides additional packages for use by developers. Red Hat does not support packages included in the CodeReady Linux Builder repository.

MAIN CONFIGURATION FILES (PRE-REQUISITES)

▪ **/etc/yum.conf**	**:** Config File
▪ **/etc/yum.repos.d/**	**:** Repo files Location
▪ **/var/log/yum.log**	**:** Logfile
▪ **/var/cache/yum/$basearch/$releasever**	**:** Cache Directory

VIEWING THE CURRENT DNF CONFIGURATIONS:

- The [main] section in the /etc/dnf/dnf.conf file contains only the settings that have been explicitly set. However, you can display all settings of the [main] section, including the ones that have not been set and which, therefore, use their default values.

  → Display the global DNF configuration:

  #dnf config-manager –dump

SETTING DNF MAIN OPTIONS:

- The /etc/dnf/dnf.conf file contains one [main] section. The key-value pairs in this section affect how DNF operates and treats repositories.

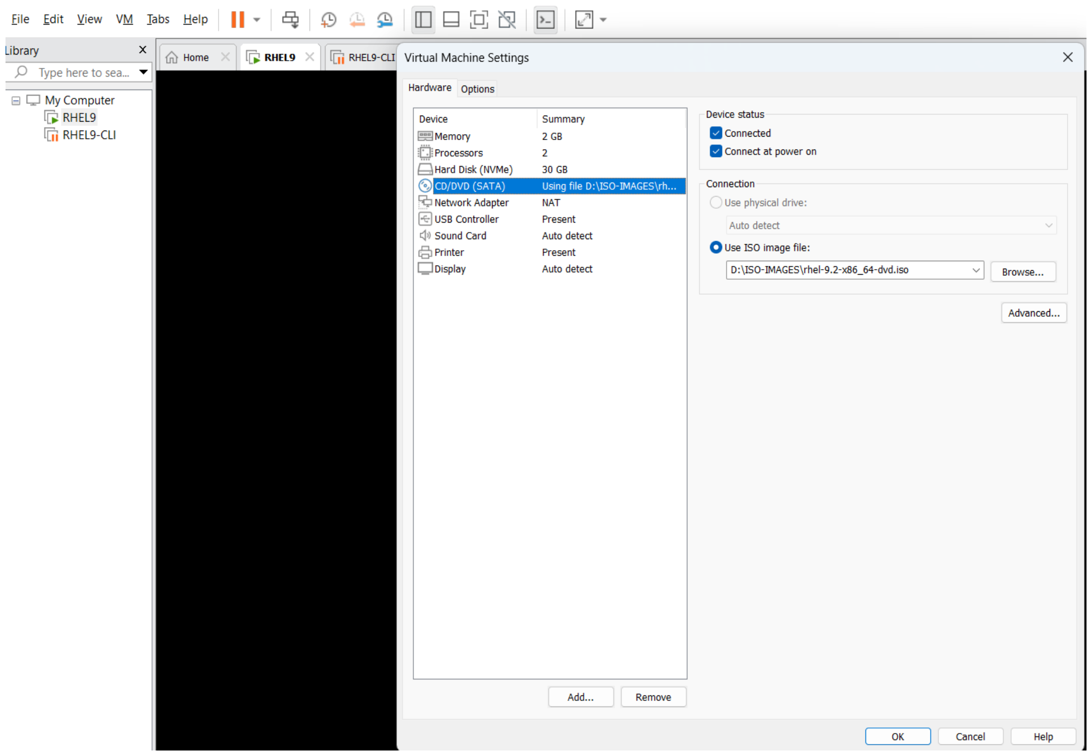
  Edit the **/etc/dnf/dnf.conf** file.

  Update the **[main]** section according to your requirements.

  Save the changes.

CONFIGURE THE LOCAL YUM/DNF REPOSITORY:

Go to VM-Settings, choose CD/DVD option and browse rhel9 iso image then click on ok.



Device	Summary
Memory	2 GB
Processors	2
Hard Disk (NVMe)	30 GB
CD/DVD (SATA)	Using file D:\ISO-IMAGES\rh...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Device status
- ☑ Connected
- ☑ Connect at power on

Connection
- ○ Use physical drive: Auto detect
- ● Use ISO image file: D:\ISO-IMAGES\rhel-9.2-x86_64-dvd.iso

→ Mounting ISO image:

```
#mount /dev/sr0 /mnt
#df -h
#ls /mnt
```

→ To mount iso image permanently:

```
#vim /etc/fstab
/dev/sr0      /mnt     iso9660    defaults    0 0
#mount -a
```

Creating a local repository: Go to Repo location and create a file with extension**.repo:**

#cd /etc/yum.repos.d/
#vim local.repo

```
[BaseOS]
    name=LocalRepo_BaseOS
    enabled=1
    gpgcheck=0
    baseurl=file:///mnt/BaseOS

[AppStream]
    name=LocalRepo_AppStream
    enabled=1
    gpgcheck=0
    baseurl=file:///mnt/AppStream
```

→ To clean the cache:

#dnf clean all

→ To list enable repositories:

#dnf repolist

YUM / DNF SYNTAX:

#yum/dnf [Options] Package-name

OPTIONS:

clean	list
repolist	search
repoinfo	info
history	provides
update	check-update
install	localinstall
remove	grouplist
groupinstall	groupremove

→ To search for a term in the name or summary of packages, enter:

#dnf search vsftpd

→ search for a term in the name, summary, or description of packages, enter:

#dnf search --all vsftpd

→ To search for a package name and list the package name and its version in the output, enter:

#dnf repoquery vsftpd

→ To search for which package provides a file, specify the file name or the path to the file:

#dnf provides ifconfig

→ Listing software packages:

#dnf list --all

#dnf repoquery

#dnf repolist

→ Display information about one or more available packages:

#dnf info <package_name>

→ List both installed and available groups:

#dnf group list

→ List mandatory, optional, and default packages contained in a particular group:

#dnf group info "<group_name>"

#dnf groupinfo “Development Tools”

→ To install packages from the repositories, enter:

#dnf install <package_name_1> <package_name_2> ...

#dnf install <path_to_file>

#dnf install /usr/bin/tree

→ To install a local RPM file, enter:

```
#dnf install <path_to_RPM_file>
#dnf install httpd -y
```

→ Install a package group:

```
#dnf group install <group_name_or_ID>
#dnf group install “Development Tools” -y
```

→ Checking for updates:

```
#dnf check-update
```

→ Updating packages:

```
#dnf upgrade
```

→ To update a single package, use:

```
#dnf upgrade package-name
```

→ To update a package group, use:

```
#dnf group upgrade group-name
```

→ To display a list of all the latest DNF transactions, use:

```
#dnf history
```

→ To display a list of all the latest operations for a selected package, use:

```
#dnf history list package-name
```

→ To display details of a particular transaction, use:

```
#dnf history info transactionID
```