

# Software Installation and Service Management



# Foreword

- This chapter describes the concepts and operation commands of three software package installation methods on openEuler: RPM, source code, and YUM, as well as the concept and operation manner of the systemd management service.

# Objectives

- On completion of this course, you will understand the concepts and operation commands of:
  - RPM
  - Source code
  - YUM
  - systemd

# Contents

## **1. Software Package Management**

- RPM Software Package

- RPM Commands (Install, Query, Upgrade, and Uninstall)

## 2. Software Package Management with DNF

## 3. Installation Through Source Code

## 4. Service Management with systemd

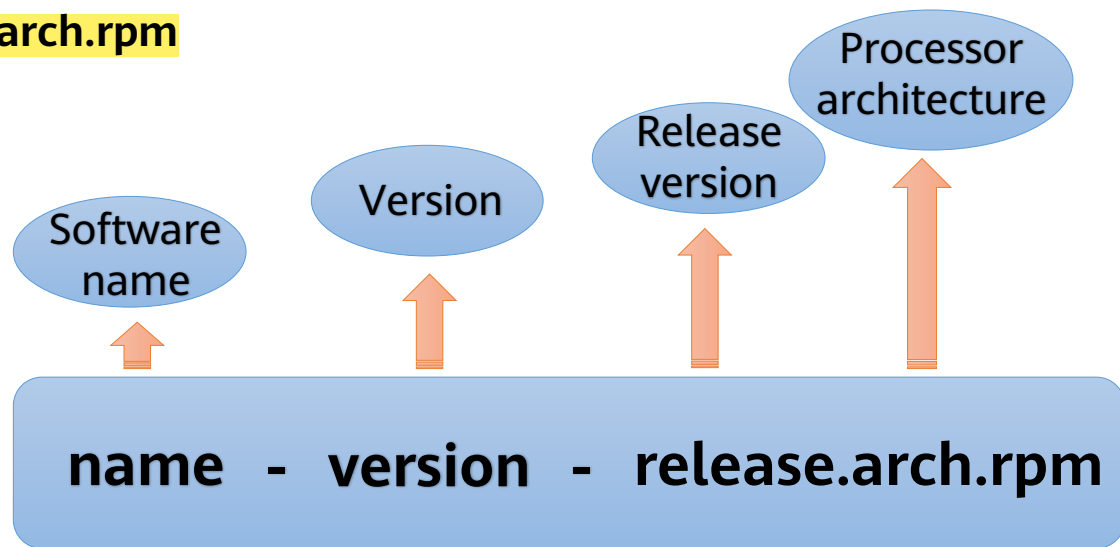
# Overview: Linux Software Package Management

- Linux software packages are classified into source code and binary packages, which have different formats, for example, RPM and TGZ. RPM and compilation are two primary software installation methods.
- RPM is the primary tool for installing software and is easy to use.
- Different distributions of OSs such as openLinux, SUSE, and CentOS, also use RPM to manage software packages.
- The software packaging format and tool varies according to the related platform. Debian and Ubuntu use the DEB package and apt-get source installation methods to manage software packages. FreeBSD uses the ports and TXZ packaging formats and the make and pkg tools.



# RPM Software Package Management

- RPM is a packaging and automatic installation tool that can download packages from the Internet. It generates .rpm files to install, uninstall, and maintain applications.
- **Name format:**
  - **name-version-release.arch.rpm**



# RPM: Advantages and Disadvantages

- Advantages:
  - Simple, convenient, and compatible
  - Parameter information is recorded in a database for easy software query, upgrade, or uninstallation.
- Disadvantages:
  - The installation environment must be the same as the packaging environment.
  - Dependencies must be processed before you uninstall the software to prevent other software being unusable.

# Contents

## **1. Software Package Management**

- RPM Software Package
  - RPM Commands (Install, Query, Upgrade, and Uninstall)

## 2. Software Package Management with DNF

## 3. Installation Through Source Code

## 4. Service Management with systemd



# Common RPM Options

- RPM is usually used for installation, deletion, upgrade, refresh, and query.
- Syntax: **rpm** [OPTION...]
- Main options:
  - **-i**: specifies the software package to be installed.
  - **-h**: uses a number sign (#) to display the process and progress of RPM installation.
  - ! ▫ **-v**: displays details of the installation process.
  - **-U**: upgrades a specified software package.
  - **-q**: queries whether a specified software package has been installed in the system or queries the content of a specified RPM package.
  - **-a**: views all software packages that have been installed in the system.
  - ! ▫ **-V**: queries the version of an installed software package.
  - **-c**: displays all configuration files.
  - **-p**: queries or validates files in a software package.

# RPM Commands - Install

- Syntax:

```
rpm -i example.rpm
```

```
rpm -iv example.rpm
```

```
rpm -ivh example.rpm
```

- Main options:

- **-i**: installs a package.
- **iv**: installs a package and displays information about the file being installed.
- **ivh**: installs a package, and displays information about the file being installed (including the progress).

# RPM Commands - Uninstall

- Syntax:

```
rpm -e example.rpm
```

```
rpm -e -nodeps example.rpm
```

```
rpm -e -allmatches example.rpm
```

- Main options:

- Dependency between packages needs to be considered when you uninstall an RPM software package.
- If the package dependency is not considered during uninstallation, run the **nodeps** command to **forcibly uninstall a package without checking its dependency**.
- If a software package has **multiple versions**, run the **allmatches** command to uninstall them in batches.

# RPM Commands - Upgrade

- Syntax:

```
rpm -U example.rpm  
rpm -Uvh example.rpm  
rpm -F example.rpm  
rpm -Fvh example.rpm
```

- Main options:

- **-Uvh:** replaces an existing package with a new package.
- **-Fvh:** upgrades an existing package.

# RPM Commands - Query

- Syntax:

```
rpm -q example.rpm
```

```
rpm -qa
```

- Main options:

- **-q**: queries whether a software package is installed.
- **-qa**: queries all installed software packages.
- **-qf**: queries all software packages that have once been installed.
- **-qp**: queries uninstalled software packages.
- **-ql**: queries the file list and full path in an installed software package.
- **-qi**: queries details about a software package.
- **-qc**: queries the configuration file in an installed software package.
- **-qd**: queries the help document in an installed software package.

# RPM Commands - Main Options

- **-qa**: queries all installed software packages.
- **-qi**: queries details about a software package.

```
[root@localhost ~]# rpm -qa
openvswitch-2.12.0-5.oe1.aarch64
tk-8.6.8-4.oe1.aarch64
scap-security-guide-0.1.39-4.oe1.noarch
libtar-1.2.20-17.oe1.aarch64
libwbclient-4.11.6-5.oe1.aarch64
```

- **-ql**: queries the file list and full path in an installed software package.

```
[root@localhost ~]# rpm -ql python3-libxml2-2.9.8-9.oe1.aarch64
/usr/lib64/python3.7/site-
packages/__pycache__/drv_libxml2.cpython-37.opt-1.pyc
/usr/lib64/python3.7/site-
packages/__pycache__/drv_libxml2.cpython-37.pyc
/usr/lib64/python3.7/site-
packages/__pycache__/libxml2.cpython-37.opt-1.pyc
```

```
[root@localhost ~]# rpm -qi python3-libxml2-2.9.8-9.oe1.aarch64
Name       : python3-libxml2
Version    : 2.9.8
Release    : 9.oe1
Architecture: aarch64
Install Date: Wed 22 Apr 2020 04:42:38 PM CST
Group      : Development/Libraries
Size       : 1411076
License    : MIT
```

# Contents

1. Software Package Management
- 2. Software Package Management with DNF**
  - Overview: DNF Tool
    - Managing Software Packages with DNF
3. Installation Through Source Code
4. Service Management with systemd



# Generation of the DNF Tool

- YUM is a Linux software management tool that automatically downloads RPM packages from a specified server and installs them. It serves as a software repository that manages software packages, and resolves the dependency problem between software packages, achieving a higher efficiency. Further, it needs the DNF tool.
  - DNF can be used because the YUM performance is poor, the memory usage is too high, and the dependency parsing speed is low. In addition, YUM is very reliant on YUM repo files. If such a file is faulty, YUM-related operations may fail. In this case, the DNF tool is used to overcome the YUM bottlenecks and improve memory usage, dependency analysis efficiency, and running speed for better use experience.

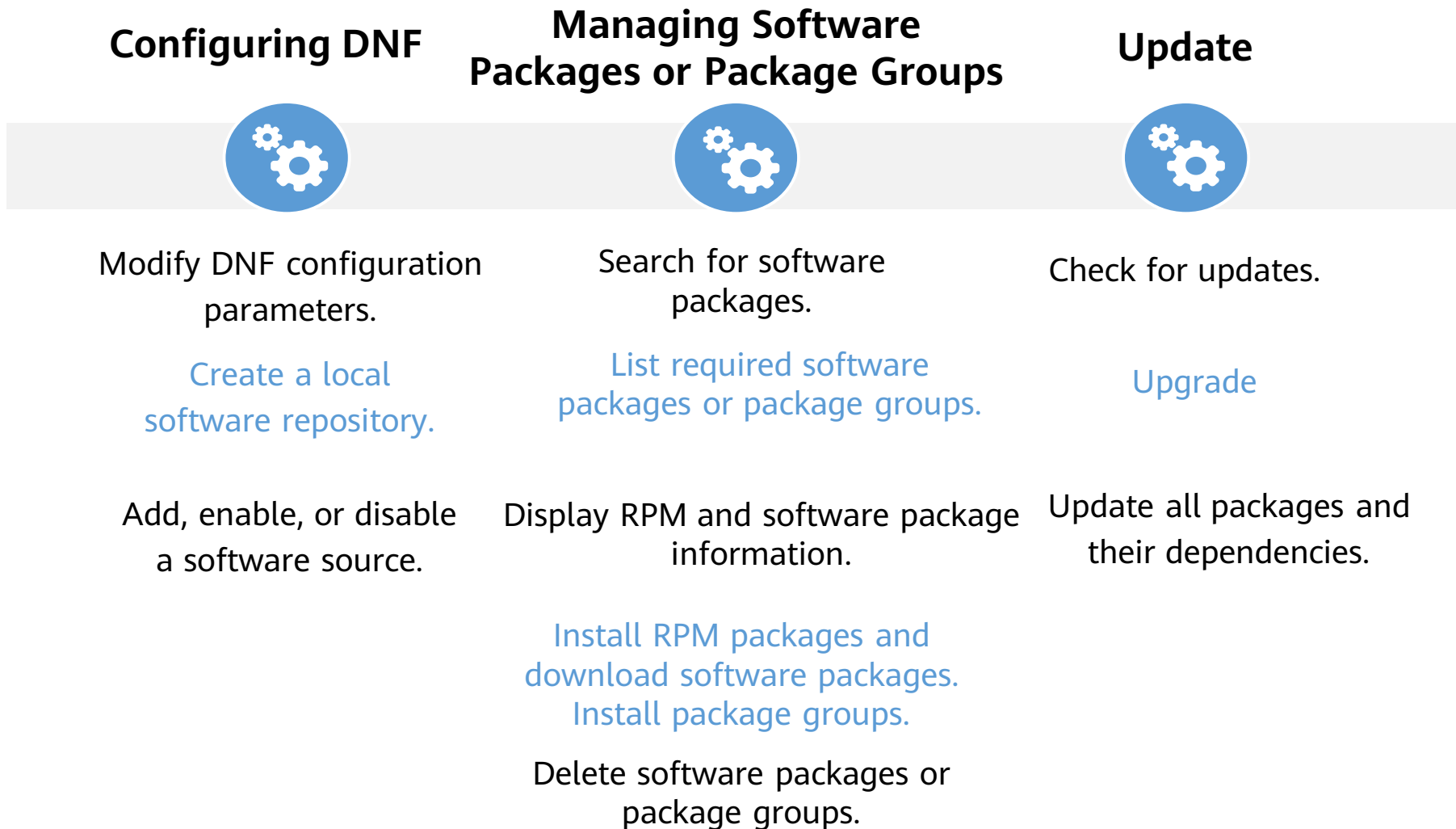
# Overview: DNF Tool

- **Dandified YUM (DNF)** is a Linux software package management tool that manages RPM software packages.
- DNF can query software package information, obtain required software packages from a specified software repository, and install, uninstall, and update the software packages by automatically processing dependencies.
- DNF is fully compatible with YUM and provides YUM-compatible command lines and APIs for extensions and plugins.
- **Only user `root` can use DNF.**

# Contents

1. Software Package Management
- 2. Software Package Management with DNF**
  - Overview: DNF Tool
    - Managing Software Packages with DNF
3. Installation Through Source Code
4. Service Management with systemd

# Managing Software Packages with DNF



# DNF - Software Source Service

- Software sources are free repositories used for installing Linux applications.
- A software source can be a network server, a CD-ROM, or even a hard drive directory.

- Advantages of Linux software sources:

- Certain software can be automatically downloaded and installed by using a tool.
- A software source allows you to receive critical security updates and handle security risks.
- The complex software dependency problem can be eliminated, boosting the installation speed.

# DNF Configuration File - `/etc/dnf/dnf.conf`

- The main DNF configuration file is `/etc/dnf/dnf.conf`. The **main** part in this file stores the global DNF configuration.
- Run the **`cat /etc/dnf/dnf.conf`** command to view parameters of **main**.
- Parameters:
  - **cachedir**: cache directory, which is used to store RPM packages and database files.
  - **best**: always tries to install the latest version during package upgrade. If the latest version cannot be installed, the system displays a cause of the failure and stops installation. The default value is **True**.
  - **installonly\_limit**: specifies the number of packages that can be synchronously installed and are listed in the **installonlypkgs** command. The default value is **3**. You are advised not to decrease the value.
  - **clean\_requirements\_on\_remove**: deletes dependencies that are no longer used during **dnf remove**. If a software package is installed using DNF instead of an explicit user request, the software package can only be deleted through **clean\_requirements\_on\_remove**. That is, the software package is introduced as a dependency. The default value is **True**.

# Configuring DNF - Modifying Configuration Parameters

- You can customize software repositories, but each repository name must be unique to avoid conflicts. You can add one or more repositories to change the location of the software source to be installed.
- Run the **vim /etc/dnf/dnf.conf** command to add one or more repositories to a file.
- Parameter description:
  - **name=repository\_name**: specifies a string that describes a software repository.
  - **baseurl=repository\_url**: specifies a software repository URL, such as the location where the **HTTP** protocol is used (<http://path/to/repo>), the location where the **FTP** protocol is used (<ftp://path/to/repo>), or the **local location** (<file:///path/to/local/repo>).



# Creating Local Software Repositories

- To create a software repository, perform the following steps:

- Run the following command to install **createrepo** as user **root**:

```
dnf install createrepo
```

- Place the required software package in a directory, for example, **/mnt/local\_repo/**.

- Run the following command to create a software repository:

```
createrepo --database /mnt/local_repo
```

# Adding Software Repositories

- To customize a software repository, add one or more repositories to the **/etc/dnf/dnf.conf** file or add the .repo file to the **/etc/yum.repos.d/** directory.
- Only user **root** can add the .repo file.
  - After the following command is successfully executed, the corresponding repository file is generated in the **/etc/yum.repos.d/** directory.

```
dnf config-manager --add-repo repository_url
```

# Enabling and Disabling Software Repositories

- After a software repository is added, run the following command to enable it as user **root** (the value of **repository** is the repository ID of the new .repo file):

```
dnf repolist # View the repository ID of the new .repo file.
```

```
dnf config-manager --set-enable repository
```

- If a software repository is no longer used, run the following command to disable it as user **root**:

```
dnf repolist # View the repository ID of the new .repo file.
```

```
dnf config-manager --set-disable repository
```

# Managing Software Packages

- DNF can be used to **install**, **query**, and **delete** software packages.
  - Run the following command to search for the required RPM software package by its name, abbreviation, or description:

```
dnf search term
```

```
[root@localhost ~]# dnf search term
created by dnf config-manager from https://repo.openeuler.org/openEuler-20.03-LTS/OS/aar
Last metadata expiration check: 0:00:06 ago on Tue 02 Jun 2020 03:27:13 PM CST.
===== Name & Summary Matched: term
xterm-help.aarch64 : Doc files for xterm
perl-Term-Cap.noarch : Perl termcap interface
gnome-terminal.aarch64 : A terminal emulator for GNOME
texlive-termcal-doc.noarch : Documentation for termcal
perl-Term-Cap-help.noarch : Documents for perl-Term-Cap
texlive-acroterm-doc.noarch : Documentation for acroterm
texlive-termlist-doc.noarch : Documentation for termlist
texlive-termmenu-doc.noarch : Documentation for termmenu
texlive-termcal-de.noarch : German localization for termcal
texlive-xwatermark-doc.noarch : Documentation for xwatermark
perl-TermReadKey-help.noarch : Documents for perl-TermReadKey
```

# Listing Software Packages

- Run the following DNF commands to list all installed and available RPM software packages:

```
dnf list all
```

```
dnf list glob_expression... # Display information about a specified RPM package.
```

- Example:

```
[root@localhost ~]# dnf list httpd
Last metadata expiration check: 0:48:05 ago on Tue 02 Jun 2020 03:27:13 PM CST.
Installed Packages
httpd.aarch64                2.4.34-15.oe1                @anaconda
```

# Displaying RPM Package Information

- Run the following DNF command to view information about an RPM package:


```
dnf info package_name...
```

- Example:

```
[root@localhost ~]# dnf info httpd
Last metadata expiration check: 0:55:43 ago on Tue 02 Jun 2020 03:27:13 PM CST.
Installed Packages
Name           : httpd
Version        : 2.4.34
Release        : 15.oel
Architecture   : aarch64
Size           : 8.8 M
Source         : httpd-2.4.34-15.oel.src.rpm
Repository     : @System
From repo      : anaconda
Summary        : Apache HTTP Server
URL            : https://httpd.apache.org/
License        : ASL 2.0
Description    : Apache HTTP Server is a powerful and flexible HTTP/1.1 compliant web
                  : server.
```

# Downloading, Installing, and Deleting RPM Packages

- Run the following DNF commands to download, install, and delete RPM packages.



```
dnf download package_name
dnf install package_name
dnf remove package_name...
```

- Example:

```
[root@localhost ~]# dnf download httpd
Last metadata expiration check: 0:10:52 ago on Tue 02 Jun 2020 04:24:40 PM CST.
httpd-2.4.34-15.oel.aarch64.rpm                               409 kB/s | 1.2 MB      00:03
```

```
[root@localhost ~]# dnf install httpd
Last metadata expiration check: 0:08:27 ago on Tue 02 Jun 2020 04:24:40 PM CST.
Package httpd-2.4.34-15.oel.aarch64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

```
[root@localhost ~]# dnf remove totem
No match for argument: totem
No packages marked for removal.
Dependencies resolved.
Nothing to do.
Complete!
```



# Managing Software Package Groups

- A package group is a collection of associated software packages, such as system tools.
  - Run the following DNF commands to view all installed and available software package groups:

```
dnf groups summary
```

```
dnf group list# List the software package group and the corresponding group ID.
```

```
[root@localhost ~]# dnf groups summary
Last metadata expiration check: 0:20:45 ago on Tue 02 Jun 2020 04:24:40 PM CST.
Installed Groups: 9
```

```
[root@localhost ~]# dnf group list
Last metadata expiration check: 0:21:08 ago on Tue 02 Jun 2020 04:24:40 PM CST.
Available Environment Groups:
  Minimal Install
  Server
Installed Environment Groups:
  Virtualization Host
Installed Groups:
  Container Management
  Development Tools
  Headless Management
  Legacy UNIX Compatibility
  Network Servers
  Scientific Support
  Security Tools
  System Tools
  Smart Card Support
```

# Displaying Package Group Information

- Run the following DNF command to list the mandatory and optional software packages in a package group:

```
dnf group info glob_expression...
```

- Example:

```
[root@localhost ~]# dnf group info "Development Tools"
Last metadata expiration check: 0:26:26 ago on Tue 02 Jun 2020 04:24:40 PM CST.

Group: Development Tools
Description: A basic development environment.
Mandatory Packages:
  autoconf
  automake
  binutils
  bison
  flex
  gcc
  gcc-c++
  gdb
  gettext
  glibc-devel
  huaweijdk-8
  libtool
  make
```

# Installing and Deleting Software Package Groups

- Each software package group has a name and an ID. You can install or delete a software package group by searching its name or ID.

```
dnf group install group_name/groupid
```

```
dnf group remove group_name/groupid
```

- Example:

```
[root@localhost ~]# dnf group install development
Last metadata expiration check: 0:31:43 ago on Tue 02 Jun 2020 04:24:40 PM CST.
No match for group package "pkgconf-pkg-config"
No match for group package "huaweijdk-8"
No match for group package "pkgconf-m4"
No match for group package "rpm-sign"
Dependencies resolved.
=====
Package                Architecture      Version           Repository        Size
=====
Installing Groups:
Development Tools

Transaction Summary
=====
Is this ok [y/N]: y
Complete!
```

```
[root@localhost ~]# dnf group remove development
Dependencies resolved.
=====
Package                Arch      Version           Repository        Size
=====
Removing:
asciidoc                noarch    8.6.10-3.oe1      @anaconda         975 k
autoconf                noarch    2.69-30.oe1       @anaconda         2.9 M
automake                 noarch    1.16.1-6.oe1      @anaconda         1.4 M
bison                   aarch64   3.5-2.oe1         @anaconda         1.1 M
byacc                   aarch64   1.9.20170709-9.oe1 @anaconda         155 k
ctags                   aarch64   5.8-27.oe1        @anaconda         367 k
diffstat                aarch64   1.62-3.oe1        @anaconda          81 k
flex                     aarch64   2.6.1-13.oe1      @anaconda         963 k
gcc-c++                 aarch64   7.3.0-20190804.h31.oe1 @anaconda        19 M
```

# Checking and Updating Software Packages

- Use DNF commands to check whether software packages or package groups in a system need to be updated, and update them as required.
  - Run the following commands to search for the required RPM software package by its name, abbreviation, or description:

```
dnf check-update
```

```
dnf update package_name / dnf group update group_name
```

- Example:

```
[root@localhost ~]# dnf check-update
Last metadata expiration check: 0:40:55 ago on Tue 02 Jun 2020 04:24:40 PM CST.
Obsoleting Packages
linux-firmware.noarch                20190815-4.oe1                @anaconda
  linux-firmware.noarch              20190815-4.oe1                @anaconda
linux-firmware.noarch                20190815-4.oe1                repository
  linux-firmware.noarch              20190815-4.oe1                @anaconda
linux-firmware.noarch                20190815-4.oe1                openEuler
  linux-firmware.noarch              20190815-4.oe1                @anaconda
linux-firmware.noarch                20190815-4.oe1                repo.openeuler.org_openEuler-20.03-LT
  linux-firmware.noarch              20190815-4.oe1                @anaconda
```

# Contents

1. Software Package Management
2. Software Package Management with DNF
- 3. Installation Through Source Code**
  - Overview: Source Code-based Software Installation
  - Procedure for Installing Software Through Source Code (configure/make/make install)
4. Service Management with systemd

# Overview: Source Code-based Software Installation

- Source code offers an alternative way to install software. On Linux, most software is released as source packages, which are more portable than binary software packages. Only one source package needs to be released for each software and can be executed by different users after compilation. However, the configuration and compilation processes are complex.
- RPM is preferred for software installation on openEuler, but source code may also be required in the following scenarios:
  - The RPM software package version is outdated, and compilation parameters do not apply to the current service.
  - No RPM software package is available for the software to be installed.
  - RPM software packages lack certain features.
  - Compilation parameters are optimized to boost performance.

# Source Code-based Software Installation: Advantages and Disadvantages

- Advantages:

- During a compilation process, you can flexibly set parameters as required to install the software.
- After local compilation, the software installed using source code is fully compatible with a local host.

- Disadvantages:

- The configuration and compilation processes are complex.
- The dependency package may not exist following a new software installation (or other problems). Consequently, software upgrade is complex and risky.



# Contents

1. Software Package Management
2. Software Package Management with DNF
- 3. Installation Through Source Code**
  - Overview: Source Code-based Software Installation
  - Procedure for Installing Software Through Source Code (configure/make/make install)
4. Service Management with systemd

# Procedure for Installing Software Through Source Code

- Procedure:

- Download and decompress a source package and verify its integrity.
- View the **README** and **INSTALL** files, which record software installation methods and precautions.
- Create a makefile by running the **./configure** script.
- Run the **make** command to automatically compile the source code into a binary file.
- Run the **make install** command to install the binary file compiled in the previous step to the corresponding directory. The default installation path is **/usr/local/**, and the corresponding configuration file is located in **/usr/local/etc** or **/usr/local/\*\*\*/etc**.

# Downloading and Decompressing a Source Package - Example

- The following uses the Python software as an example to describe the source code-based installation:
  - Download and decompress a source package and verify its integrity.

```
wget https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tgz
tar -zxvf Python-3.7.7.tgz
```

```
[root@localhost ~]# wget https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tgz
--2020-06-02 09:34:46-- https://www.python.org/ftp/python/3.7.7/Python-3.7.7.tgz
Resolving www.python.org (www.python.org)... 151.101.108.223, 2a04:4e42:36::223
Python-3.7.7/Objects/clinic/moduleobject.c.h
Python-3.7.7/Objects/clinic/odictobject.c.h
Python-3.7.7/Objects/bytearrayobject.c
Python-3.7.7/Objects/typeobject.c
Python-3.7.7/Objects/lnotab_notes.txt
Python-3.7.7/Objects/methodobject.c
Python-3.7.7/Objects/tupleobject.c
Python-3.7.7/Objects/obmalloc.c
Python-3.7.7/Objects/object.c
Python-3.7.7/Objects/abstract.c
Python-3.7.7/Objects/listobject.c
Python-3.7.7/Objects/bytes_methods.c
Python-3.7.7/Objects/dictnotes.txt
Python-3.7.7/Objects/typeslots.inc
[openeuler@host-192-168-0-183 ~]$ tar -zxvf Python-3.7.7.tgz
```

# Viewing the **README** File - Example

- The following uses the Python software as an example to describe the source code-based installation:
  - Go to the source code directory and view the **README** file.

```
cat Python-3.7.7/README.rst
```

```
[openeuler@host-192-168-0-183 Python-3.7.7]$ cat README.rst
This is Python version 3.7.7
=====

.. image:: https://travis-ci.org/python/cpython.svg?branch=3.7
   :alt: CPython build status on Travis CI
   :target: https://travis-ci.org/python/cpython/branches

.. image:: https://dev.azure.com/python/cpython/_apis/build/status/Azure%20Pipelines%20CI?branch
Name=3.7
   :alt: CPython build status on Azure Pipelines
   :target: https://dev.azure.com/python/cpython/_build/latest?definitionId=4&branchName=3.7

.. image:: https://codecov.io/gh/python/cpython/branch/3.7/graph/badge.svg
   :alt: CPython code coverage on Codecov
   :target: https://codecov.io/gh/python/cpython/branch/3.7

Copyright (c) 2001-2020 Python Software Foundation. All rights reserved.

See the end of this file for further copyright and license information.
```

# Generating a Makefile - Example

- The following uses the Python software as an example to describe the source code-based installation:
  - Run the `./configure` command to generate a makefile.

```
./configure --prefix=/usr/local/Python
```

```
creating Modules/Setup
creating Modules/Setup.local
creating Makefile

If you want a release build with all stable optimizations active (PGO, etc),
please run ./configure --enable-optimizations

[root@host-192-168-0-183 Python-3.7.7]# ./configure --prefix=/usr/local/Python
```

# Software Installation Through Source Code- Example

- The following uses the Python software as an example to describe the source code-based installation:

- Run the **make** command to start compilation.

```
make/make clean
```

```
renaming build/scripts-3.7/pydoc3 to build/scripts-3.7/pydoc3.7
renaming build/scripts-3.7/idle3 to build/scripts-3.7/idle3.7
renaming build/scripts-3.7/2to3 to build/scripts-3.7/2to3-3.7
renaming build/scripts-3.7/pyvenv to build/scripts-3.7/pyvenv-3.7
gcc -pthread      -Xlinker -export-dynamic -o Programs/_testembed Programs/_testembed.o libpython3.7m.a -lcrypt -lpthread -ldl  -lutil  -lm
sed -e "s,@EXENAME@,/usr/local/Python/bin/python3.7m," < ./Misc/python-config.in >python-config.py
LC_ALL=C sed -e 's,\$(\[([A-Za-z0-9_]*\)),\${\1}},g' < Misc/python-config.sh >python-config
[root@host-192-168-0-183 Python-3.7.7]# make
```

- Run the **make install** command to install the software.

```
make install
```

(Note: Related environment components may be missing during installation. You can use the YUM tool to download and install these components.)

# Contents

1. Software Package Management
2. Software Package Management with DNF
3. Installation Through Source Code
- 4. Service Management with systemd**
  - Introduction to systemd
  - Managing Services with systemd (systemctl)

# Introduction to systemd

- On Linux, systemd is a system and service manager compatible with SysV and LSB initialization scripts. Enabling systemd provides on-demand activation policies based on daemon processes.
- The systemd service supports snapshot and system status restoration, and maintains mount and self-mount points. In this way, finer logical control can be implemented between services based on the dependency relationship, providing higher parallel performance.



# systemd Unit Concepts

- The activation and monitoring system of systemd is based on units. A unit consists of a name and a type corresponding to a configuration file. The following are main unit types:
  - Service unit: system service
  - Target unit: a group of systemd units
  - Automount unit: file system mount point
  - Device unit: device file identified by a kernel
  - Mount unit: file system mount point
  - Path unit: a file or directory in a file system
  - Scope unit: externally created process
  - Snapshot unit: saving status of the systemd manager

# systemd Features

- systemd has the following features:
  - Fast activation
  - On-demand activation
  - Service lifecycle management by cgroups
  - Mount and automount point management
  - Transactional dependency management
  - Compatibility with SysVinit scripts
  - System status snapshots and system restoration

# Contents

1. Software Package Management
2. Software Package Management with DNF
3. Installation Through Source Code
- 4. Service Management with systemd**
  - Introduction to systemd
  - Managing Services with systemd (systemctl)

# Managing System Services

- A systemd user can run, stop, restart, display, and enable or disable system services by running the **systemctl** command.
- The **systemctl** command functions similar to the **sysvinit** command. However, you are advised to use the **systemctl** command to manage system services.
- Differences between **systemctl** and **sysvinit** are as follows:

ysvinit	systemd	Remarks
service NetworkManager start	<b>systemctl start NetworkManager</b>	Starts a service (without restarting an existing one).
service NetworkManager stop	<b>systemctl stop NetworkManager</b>	Stops a service (without restarting an existing one).
service NetworkManager reload	<b>systemctl reload NetworkManager</b>	Reloads a configuration file without interrupting the wait operation.

# systemctl - Displaying Services

- To view a running service, run the following command:

`systemctl list-units --type service` # To view all services, add **-all** to the end of the command line.

```
[root@localhost ~]# systemctl list-units --type service
```

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
atd.service	loaded	active	running	Deferred execution scheduler
auditd.service	loaded	active	running	Security Auditing Service
chronyd.service	loaded	active	running	NTP client/server
crond.service	loaded	active	running	Command Scheduler
dbus.service	loaded	active	running	D-Bus System Message Bus
dracut-shutdown.service	loaded	active	exited	Restore /run/initramfs on shutdown
getty@tty1.service	loaded	active	running	Getty on tty1
gssproxy.service	loaded	active	running	GSSAPI Proxy Daemon
hwclock-save.service	loaded	active	exited	Update RTC With System Clock
irqbalance.service	loaded	active	running	irqbalance daemon
iscsi.service	loaded	active	exited	Login and scanning of iSCSI devices
kdump.service	loaded	active	exited	Crash recovery kernel arming
kmod-static-nodes.service	loaded	active	exited	Create list of static device nodes
libstoragemgmt.service	loaded	active	running	libstoragemgmt plug-in server daemon
libvirtd.service	loaded	active	running	Virtualization daemon
lm_sensors.service	loaded	failed	failed	Hardware Monitoring Sensors

# systemctl - Displaying Service Statuses

- To view the status of a service, run the following command:

```
systemctl status name.service
```

- Parameter description:
  - **Loaded**: determines whether a service is loaded and whether the corresponding absolute path is enabled.
  - **Active**: determines whether a service is running and displays a time stamp.
  - **Main PID**: specifies a PID of the corresponding system service.
  - **CGroup**: specifies other information about the cgroup.

# systemctl - Related Operations

- Run the following **systemctl** commands to run, stop, restart, display, enable, or disable system services:

- Run a service.

```
systemctl start name.service
```

- Stop a service.

```
systemctl stop name.service
```

- Restart a service.

```
systemctl restart name.service
```

- Enable a service.

```
systemctl enable name.service
```

- Disable a service.

```
systemctl disable name.service
```

# systemctl - Other Operations

- Run the following **systemctl** commands to shut down, restart, or hibernate a system:

- Shut down a system.

```
systemctl poweroff # Shut down a system and power it off.
```

```
systemctl halt # Shut down a system but do not power it off.
```

- Restart a system.

```
systemctl reboot
```

- Suspend a system.

```
systemctl suspend
```

- Hibernate a system.

```
systemctl hibernate
```

```
systemctl hybrid-sleep # Suspend and hibernate a system.
```



# Quiz

1. Which of the following commands is not a Linux file operating command? ( )
  - A. cat
  - B. chmod
  - C. clear
  - D. more

# Summary

- This chapter describes the concepts and operations of three installation methods of openEuler (RPM, source code, and YUM), in addition to the concept and operations of the systemd management service.

# More Information

---

openEuler open-source community:

[https://openeuler.org/en/docs/20.03\\_LTS/docs/Releasenotes/release\\_notes.html](https://openeuler.org/en/docs/20.03_LTS/docs/Releasenotes/release_notes.html)

# Thank you.

把数字世界带入每个人、每个家庭、  
每个组织，构建万物互联的智能世界。  
Bring digital to every person, home, and  
organization for a fully connected,  
intelligent world.

**Copyright©2022 Huawei Technologies Co., Ltd.  
All Rights Reserved.**

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

