# Red Hat Enterprise Linux 8.0 Beta

## Configuring basic system settings

A guide to configuring basic system settings on Red Hat Enterprise Linux 8

Last Updated: 2019-03-29

# Red Hat Enterprise Linux 8.0 Beta Configuring basic system settings

A guide to configuring basic system settings on Red Hat Enterprise Linux 8

## Legal Notice



## Abstract

This document describes basics of system administration on Red Hat Enterprise Linux 8. The title focuses on: basic tasks that a system administrator needs to do just after the operating system has been successfully installed, installing software with yum, using systemd for service management, managing users, groups and file permissions, using chrony to configure NTP, working with Python 3 and others.

# Table of Contents

# THIS IS A BETA VERSION!

Thank you for your interest in Red Hat Enterprise Linux 8.0 Beta. Be aware that:

- Beta code should not be used with production data or on production systems.
- Beta does not include a guarantee of support.
- Feedback and bug reports are welcome. Discussions with your account representative, partner contact, and Technical Account Manager (TAM) are also welcome.
- Upgrades to or from a Beta are not supported or recommended.

# PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:

  1. Go to the Bugzilla website.

  2. As the Component, use **Documentation**.

  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.

  4. Click **Submit Bug**.

# CHAPTER 1. GETTING STARTED WITH SYSTEM ADMINISTRATION IN RED HAT ENTERPRISE LINUX

The following sections provide an overview of the basic tasks that system administrators might need to perform just after Red Hat Enterprise Linux has been installed.

> **NOTE**
>
> Such tasks may include items that are usually done already during the installation proces, but they do not have to be done necessarily, such as the registration of the system. The sections dealing with such tasks provide a brief summary of how this can be achived during the installation and links to related documentation.
>
> For information on Red Hat Enterprise Linux installation, see Installing Red Hat Enterprise Linux 8.

> **NOTE**
>
> The following sections mention some commands to be performed. The commands that need to be entered by the `root` user have **#** in the prompt, while the commands that can be performed by a regular user, have **$** in their prompt.

Although all post-installation tasks can be achieved through the command line, you can also use the **Cockpit** tool to perform some of them.

## 1.1. WHAT COCKPIT IS AND WHICH TASKS IT CAN BE USED FOR

**Cockpit** is an interactive server administration interface. **Cockpit** interacts directly with the operating system from a real Linux session in a browser.

**Cockpit** enables to perform these tasks:

- Monitoring basic system features, such as hardware information, time configuration, performance profiles, connection to the realm domain
- Inspecting system log files
- Managing network interfaces and configuring firewall
- Handling docker images
- Managing virtual machines
- Managing user accounts
- Monitoring and configuring system services
- Creating diagnostic reports
- Setting kernel dump configuration
- Managing packages
- Configuring SELinux

- Updating software
- Managing system subscriptions
- Accessing the terminal

For more information on installing and using **Cockpit**, see Managing systems using the Cockpit web interface.

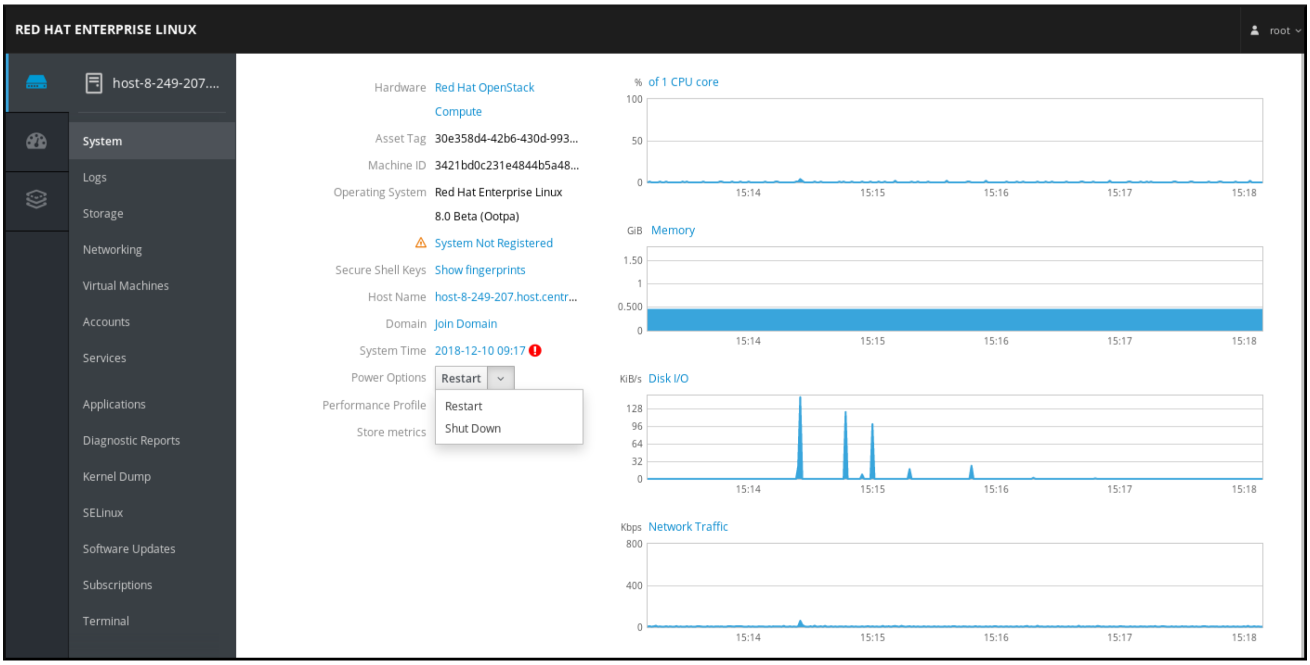
### 1.1.1. Using Cockpit for basic system administration tasks

For the most basic operations, use the `System` menu.

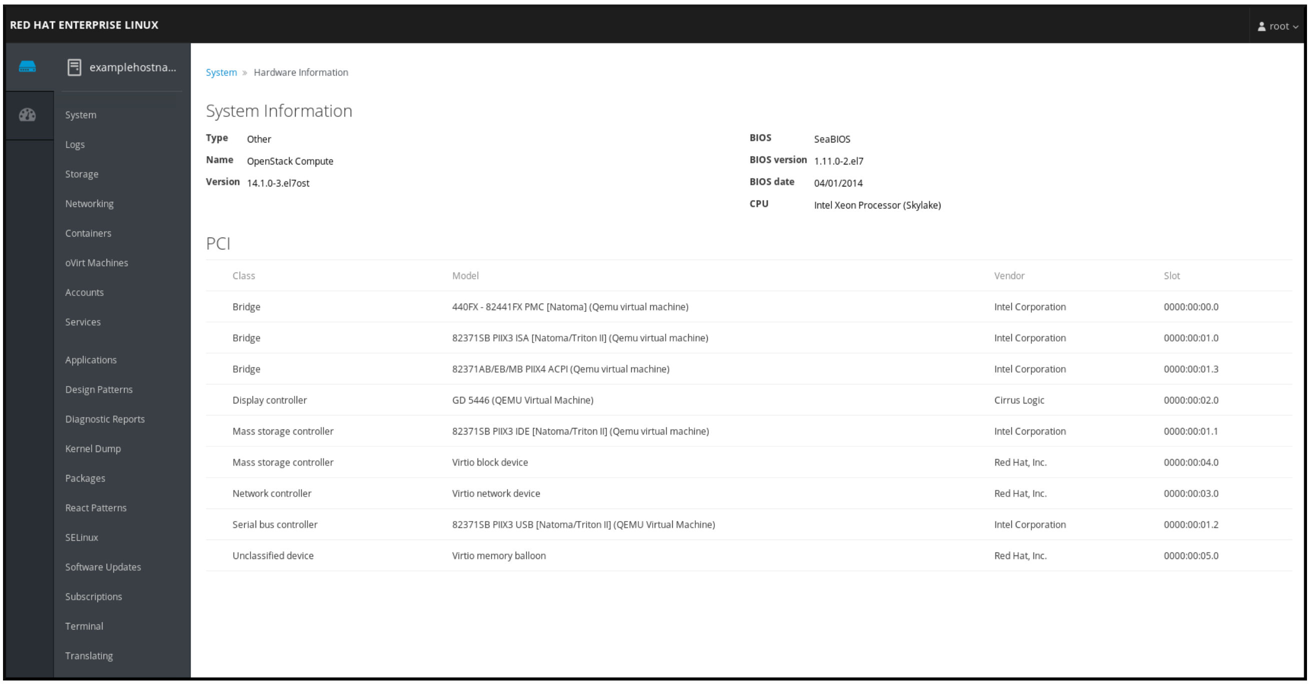
Such operations include for example:

- shutting down or restarting the system
- inspecting hardware information
- setting performance profiles
- setting hostname
- connecting to realmd domain

**Figure 1.1. System shutdown or restart in Cockpit**



### Figure 1.2. Inspecting hardware information in Cockpit



### Figure 1.3. Setting performance profiles in Cockpit

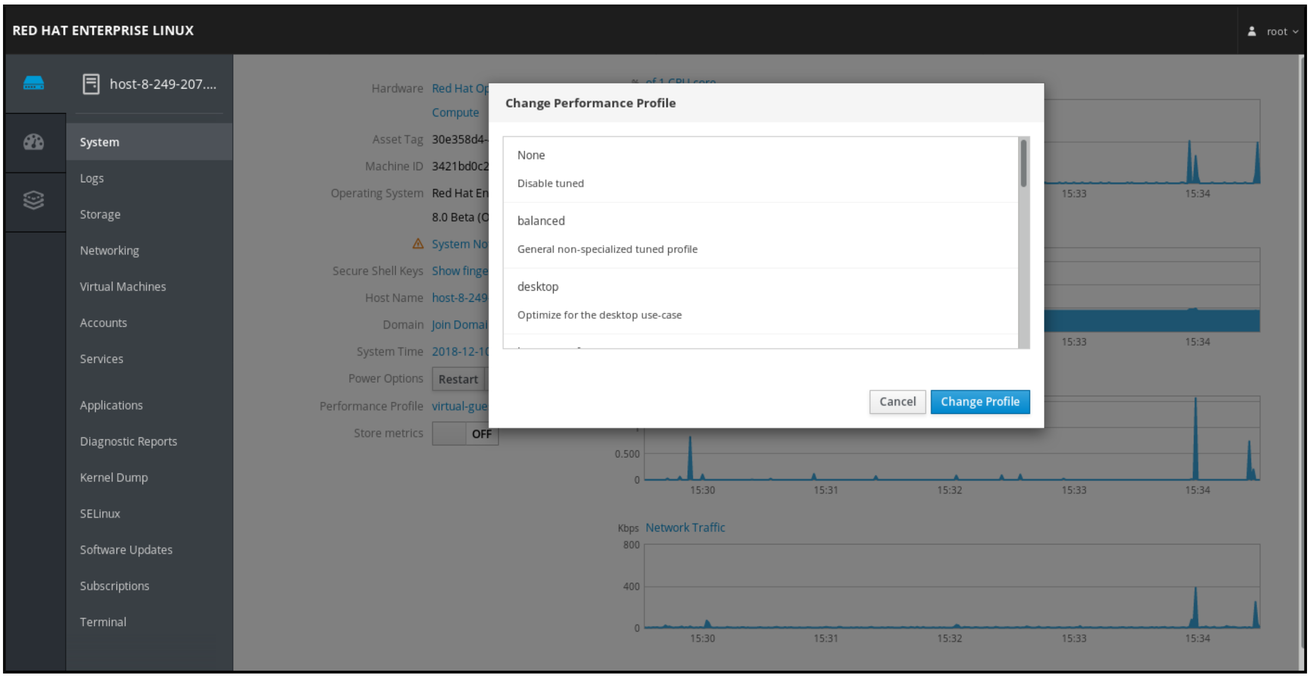


**Figure 1.4. Setting hostname in Cockpit**

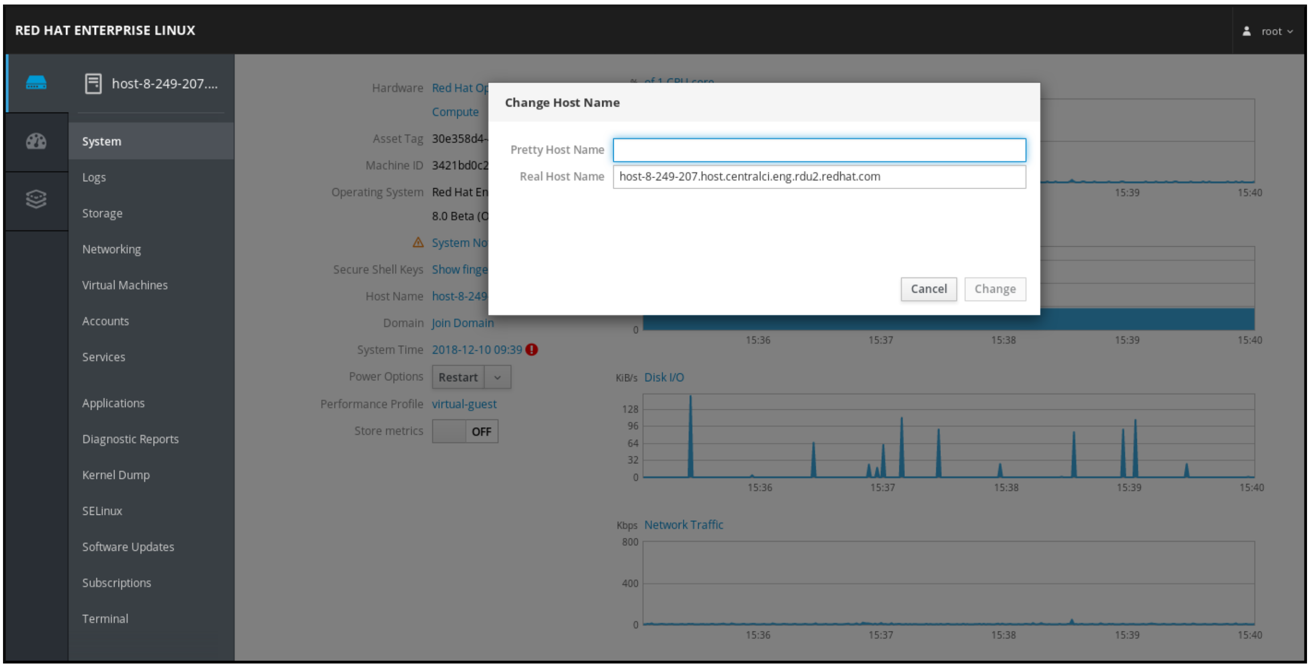
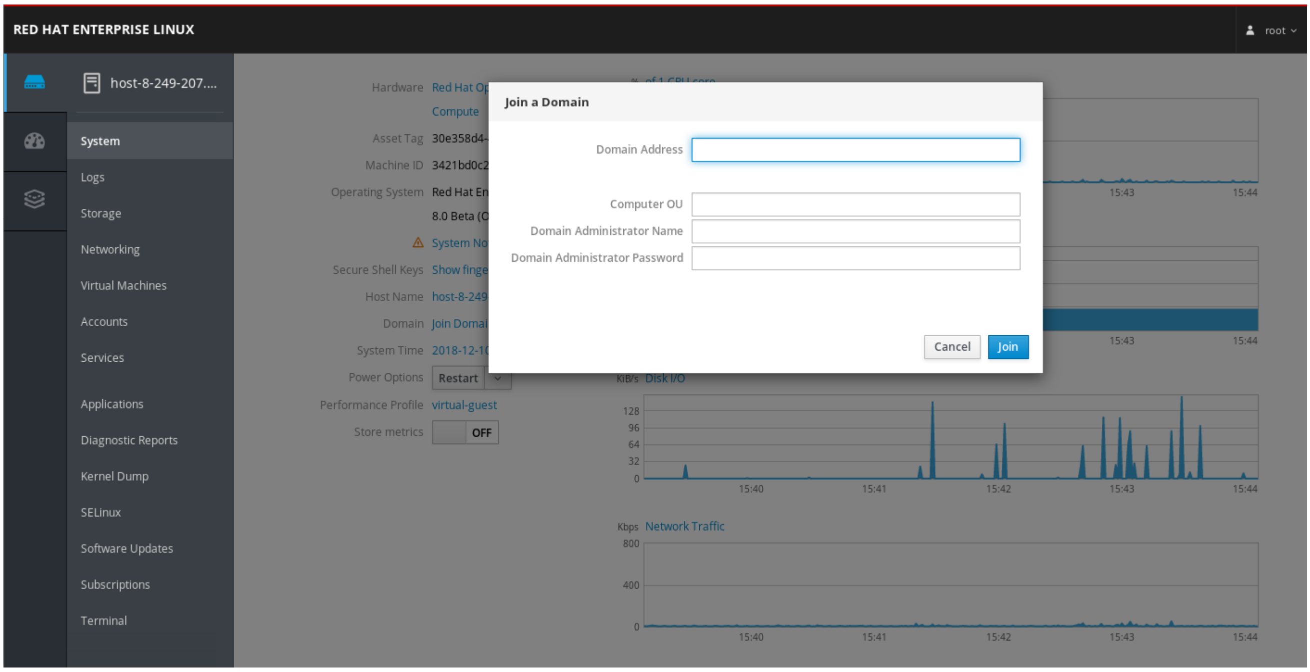



**Figure 1.5. Connecting to realmd domain in Cockpit**




# 1.2. BASIC CONFIGURATION OF ENVIRONMENT

Basic configuration of environment includes:

- Date and time
- System locales
- Keyboard layout

Setting of these items is normally a part of the installation process. For more information, see Installing Red Hat Enterprise Linux 8.

### 1.2.1. Introduction to configuring the date and time

Accurate timekeeping is important for a number of reasons. In Red Hat Enterprise Linux, timekeeping is ensured by the **NTP** protocol, which is implemented by a daemon running in user space. The user space daemon updates the system clock running in the kernel. The system clock can keep time by using various clock sources.

Red Hat Enterprise Linux 8 uses the **chronyd** daemon to implement **NTP**. **chronyd** is available from the **chrony** package. For more information on configuring and using **NTP** with **chronyd**, see Chapter 5, *Using the Chrony suite to configure NTP*.

#### Displaying the current date and time

- To display the current date and time, use one of the following commands:

  ```
  ~]$ date
  ~]$ timedatectl
  ```

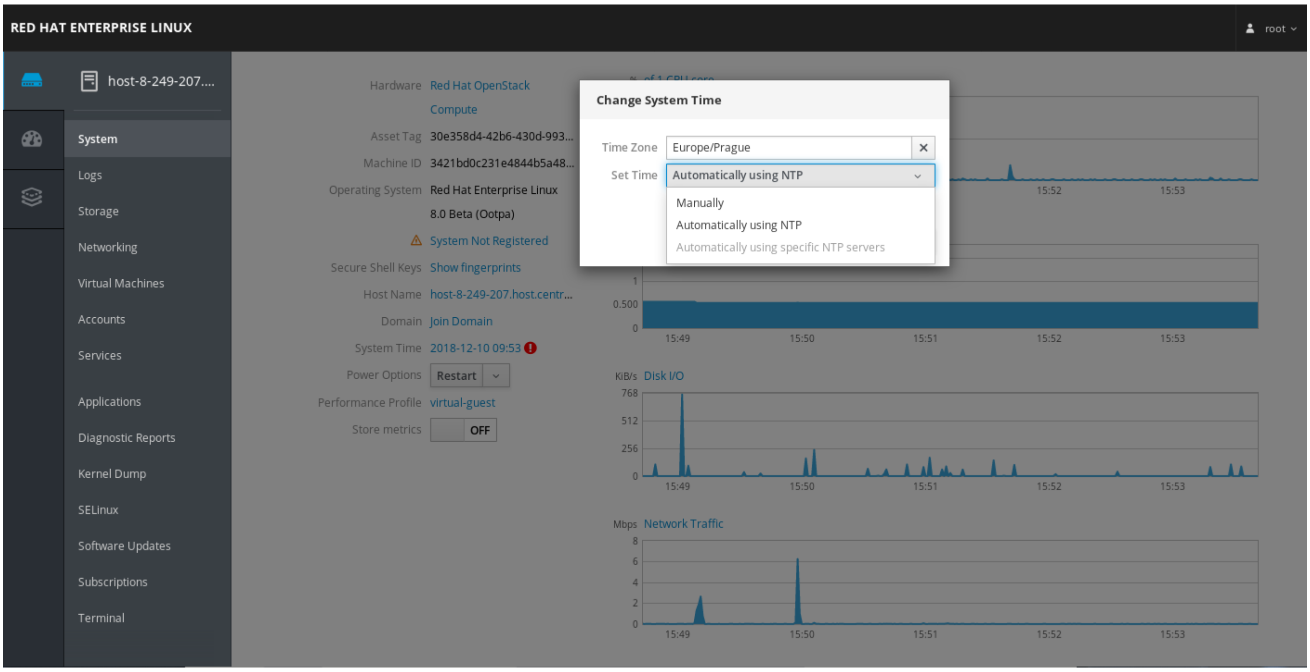
  Note that the **timedatectl** command provides more verbose output, including universal time, currently used time zone, the status of the Network Time Protocol (NTP) configuration, and some additional information.

For more information on configuring the date and time during the installation, see Installing Red Hat Enterprise Linux 8.

#### 1.2.1.1. Managing time configurations in Cockpit

You can use Cockpit to handle time configurations as well. Under the **System** option, you can configure the **NTP** protocol, or change the time zones manually.

**Figure 1.6. Managing time configurations in Cockpit**




### 1.2.2. Introduction to configuring the system locale

System-wide locale settings are stored in the **/etc/locale.conf** file, which is read at early boot by the **systemd** daemon. The locale settings configured in **/etc/locale.conf** are inherited by every service or user, unless individual programs or individual users override them.

Basic tasks to handle the system locales:

- Listing available system locale settings:

```
~]$ localectl list-locales
```

- Displaying current status of the system locales settings:

```
~]$ localectl status
```

- Setting or changing the default system locale settings:

```
~]# localectl set-locale LANG=locale
```

### 1.2.3. Introduction to configuring the keyboard layout

The keyboard layout settings control the layout used on the text console and graphical user interfaces.

Basic tasks to handle the keyboard layout include:

- Listing available keymaps:

```
~]$ localectl list-keymaps
```

- Displaying current status of keymap settings:

```
~]$ localectl status
```

- Setting or changing the default system keymap:

```
~]# localectl set-keymap
```

## 1.3. CONFIGURING AND MANAGING NETWORK ACCESS

### 1.3.1. Configuring network access during the installation process

Ways to configure network access during the installation proces:

- The **Network & Hostname** menu at the Installation Summary screen in the graphical user interface of the **Anaconda** installation program
- The **Network settings** option in the text mode of the **Anaconda** installation program
- The Kickstart file

When the system boots for the first time after the installation has finished, any network interfaces which you configured during the installation are automatically activated.

For detailed information on configuration of network access during installation process, see Installing Red Hat Enterprise Linux 8.

### 1.3.2. Managing network connections after the installation process using nmcli

Run the following commands to manage network connections using the **nmcli** utility.

> **NOTE**
>
> The **nmcli** utility has a powerful syntax completion feature when the **`Tab`** key is pressed twice. You need to have the **`bash-completion`** package installed to enable it.

To create a new connection:

```
~]# nmcli con add type type of the connection con-name connection name
ifname ifname ipv4.addresses ipv4 address ipv4.gateway gateway address
```

Here, replace:

- *type of the connection* by the required type of the device
- *connection name* by the required connection name
- *ifname* by the required device name
- *ipv4 address* by the required IPv4 address/netmask
- *gateway address* by the required gateway address

Note that *ipv4 address* and *gateway address* are optional settings, while all remaining settings are required.

You can also create a new connection in assisted mode. To do so, run this command, and follow the instructions that will prompt you for input of particular configuration settings of this connection:

```
~]# nmcli -a con add
```

To modify the existing connection:

```
~]# nmcli con mod connection name setting.property newvalue
```

Here, replace:

- *connection name* by the name of the connection that you want to modify
- *setting.property* by the configuration setting that you want to modify
- *newvalue* by the required value of this configuration setting

For example, to set the method of the configuration of IPv4 address (ipv4.method) to **`auto`** for the connection named **`enp0`**, use the following command:

```
~]# nmcli con mod enp0 ipv4.method auto
```

To edit a connection, run the following command:

```
~]# nmcli connection edit connection name
```

Here, replace *connection name* by the name of the connection that you want to edit.

To display all connections:

```
~]# nmcli con show
```

To display active connections:

```
~]# nmcli con show --active
```

To display all configuration settings of a particular connection:

```
~]# nmcli con show con-name connection name
```

Here, replace *connection name* by the name of the required connection.

Then, follow the instructions that will prompt you for input of particular configuration settings. To display all possible options of any configuration setting, use the **`print`** command in the editor.

### 1.3.3. Managing network connections after the installation process using nmtui

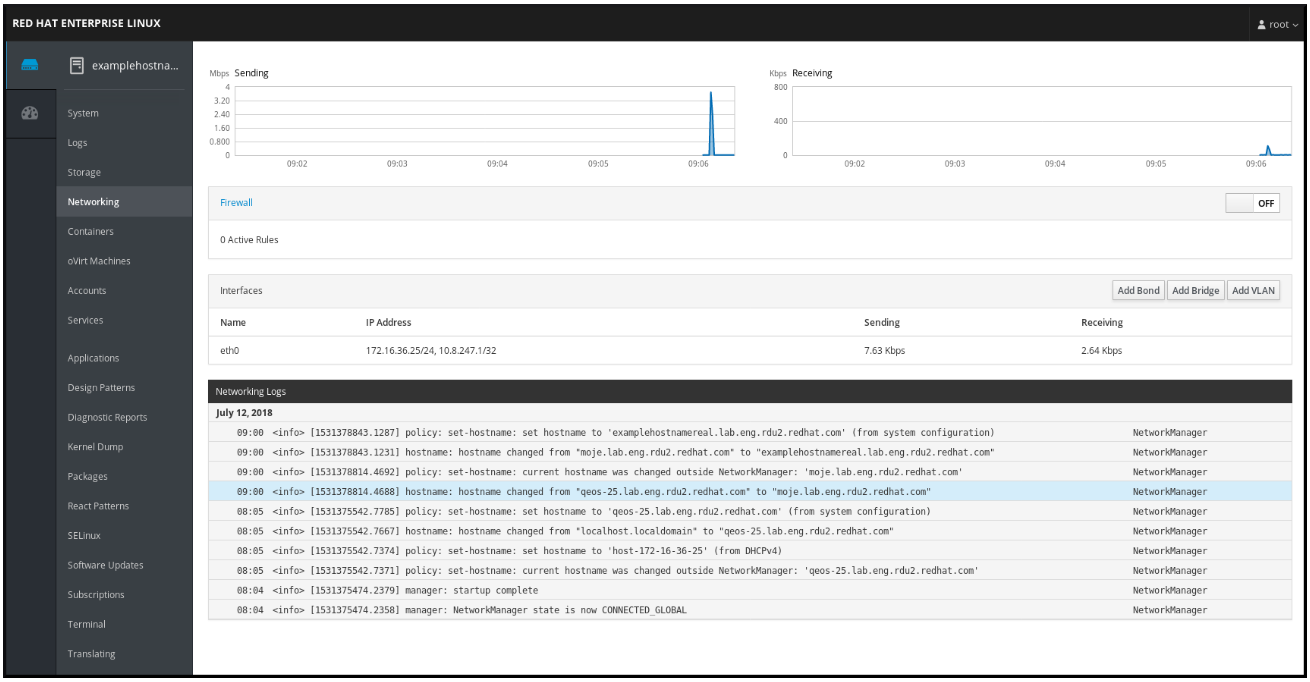
The **NetworkManager** text user interface (TUI) utility, **nmtui**, provides a text interface to configure networking by controlling **NetworkManager**.

### 1.3.4. Managing networking in Cockpit

In **Cockpit**, the **`Networking`** menu enables you:

- To display currently received and sent packets
- To display the most important characteristics of available network interfaces
- To display content of the networking logs.
- To add various types of network interfaces (bond, team, bridge, VLAN)

**Figure 1.7. Managing Networking in Cockpit**




# 1.4. THE BASICS OF REGISTERING THE SYSTEM AND MANAGING SUBSCRIPTIONS

## 1.4.1. What Red Hat subscriptions are and which tasks they can be used for

The products installed on Red Hat Enterprise Linux, including the operating system itself, are covered by subscriptions.

A subscription to Red Hat Content Delivery Network is used to track:

- Registered systems
- Products installed on those system
- Subscriptions attached to those product

## 1.4.2. Registering the system after the installation

Your subscription can be registered during the installation process. For more information, see Installing Red Hat Enterprise Linux 8.

If you have not registered your system during the installation process, you can do it afterwards by applying the following procedure. Note that all commands in this procedure need to be performed as the **`root`** user.

**Registering and subscribing your system**

1. Register your system:

   ```
   ~]# subscription-manager register
   ```

   The command will prompt you to enter your Red Hat Customer Portal user name and password.

2. Determine the pool ID of a subscription that you require:

```
~]# subscription-manager list --available
```

This command displays all available subscriptions for your Red Hat account. For every subscription, various characteristics are displayed, including the pool ID.

3. Attach the appropriate subscription to your system by replacing *pool_id* with the pool ID determined in the previous step:

```
~]# subscription-manager attach --pool=pool_id
```

## 1.5. INSTALLING SOFTWARE

This section provides information to guide you through the basics of software installation. It mentions the prerequisites that you need to fulfil to be able to install software, provides the basic information on software packaging and software repositories, and references the ways to perform basic tasks related to software installation.

### 1.5.1. Prerequisites for software installation

The Red Hat Content Delivery Network subscription service provides a mechanism to handle Red Hat software inventory and enables you to install additional software or update already installed packages. You can start installing software once you have registered your system and attached a subscription, as described in Section 1.4, “The basics of registering the system and managing subscriptions”.

### 1.5.2. Introduction to the system of software packaging and software repositories

All software on a Red Hat Enterprise Linux system is divided into RPM packages, which are stored in particular repositories. When a system is subscribed to the Red Hat Content Delivery Network, a repository file is created in the **`/etc/yum.repos.d/`** directory.

Use the **`yum`** utility to manage package operations:

- Searching information about packages
- Installing packages
- Updating packages
- Removing packages
- Checking the list of currently available repositories
- Adding or removing a repository
- Enabling or disabling a repository

For information on basic tasks related to the installation of software, see Section 1.5.3, “Managing basic software-installation tasks with subscription manager and yum”.

### 1.5.3. Managing basic software-installation tasks with subscription manager and yum

The most basic software-installation tasks that you might need after the operating system has been installed include:

- Listing all available repositories:

```
~]# subscription-manager repos --list
```

- Listing all currently enabled repositories:

```
~]$ yum repolist
```

- Enabling or disabling a repository:

```
~]# subscription-manager repos --enable repository
```

```
~]# subscription-manager repos --disable repository
```

- Searching for packages matching a specific string:

```
~]$ yum search string
```

- Installing a package:

```
~]# yum install package_name
```

- Updating all packages and their dependencies:

```
~]# yum update
```

- Updating a package:

```
~]# yum update package_name
```

- Uninstalling a package and any packages that depend on it:

```
~]# yum remove package_name
```

- Listing information on all installed and available packages:

```
~]$ yum list all
```

- Listing information on all installed packages:

```
~]$ yum list installed
```

## 1.6. MAKING SYSTEMD SERVICES START AT BOOT TIME

Systemd is a system and service manager for Linux operating systems that introduces the concept of systemd units.

This section provides the information on how to ensure that a service is enabled or disabled at boot time. It also explains how to manage the services through **Cockpit**.

### 1.6.1. Enabling or disabling the services

You can determine services that are enabled or disabled at boot time already during the installation process, or you can enable or disable a service on an installed operating system.

To create the list of services enabled or disabled at boot time during the installation process, use the **`services`** option in the Kickstart file:

```
services [--disabled=list] [--enabled=list]
```

> **NOTE**
>
> The list of disabled services is processed before the list of enabled services. Therefore, if a service appears on both lists, it will be enabled. The list of the services should be given in the comma separated format. Do not include spaces in the list of services.

To enable or disable a service on an already installed operating system:

```
~]# systemctl enable service_name
```

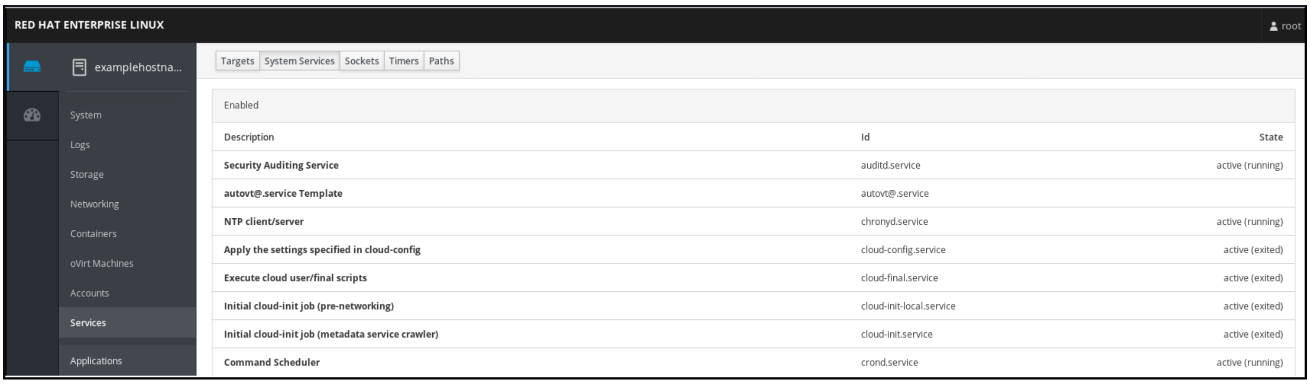
```
~]# systemctl disable service_name
```

For further details on enabling and disabling services, see Managing services with systemd.

### 1.6.2. Managing services in Cockpit

In **Cockpit**, select **`Services`** to manage systemd targets, services, sockets, timers and paths. There you can check their status, start or stop them, enable or disable them.

**Figure 1.8. Managing services in Cockpit**




## 1.7. ENHANCING SYSTEM SECURITY WITH A FIREWALL, SELINUX AND SSH LOGINGS

Computer security is the protection of computer systems from the theft or damage to their hardware, software, or information, as well as from disruption or misdirection of the services they provide. Ensuring computer security is therefore an essential task not only in the enterprises processing sensitive data or handling some business transactions.

Computer security includes a wide variety of features and tools. This section covers only the basic security features that you need to configure after you have installed the operating system. For detailed information on securing Red Hat Enterprise Linux, see Configuring and managing security.

## 1.7.1. Ensuring the firewall is enabled and running

### 1.7.1.1. What a firewall is and how it enhances system security

A firewall is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted, secure internal network and another outside network.

The firewall is provided by the **firewalld** service, which is automatically enabled during the installation. However, if you explicitly disabled the service, you can re-enable it, as described in Section 1.7.1.2, “Re-enabling the firewalld service”.

### 1.7.1.2. Re-enabling the firewalld service

In case that the **firewalld** service is disabled after the installation, Red Hat recommends to consider re-enabling it.

You can display the current status of **firewalld** even as a regular user:

```
~]$ systemctl status firewalld
```

If **firewalld** is not enabled and running, switch to the **root** user, and change its status:

```
~]# systemctl start firewalld
```

```
~]# systemctl enable firewalld
```

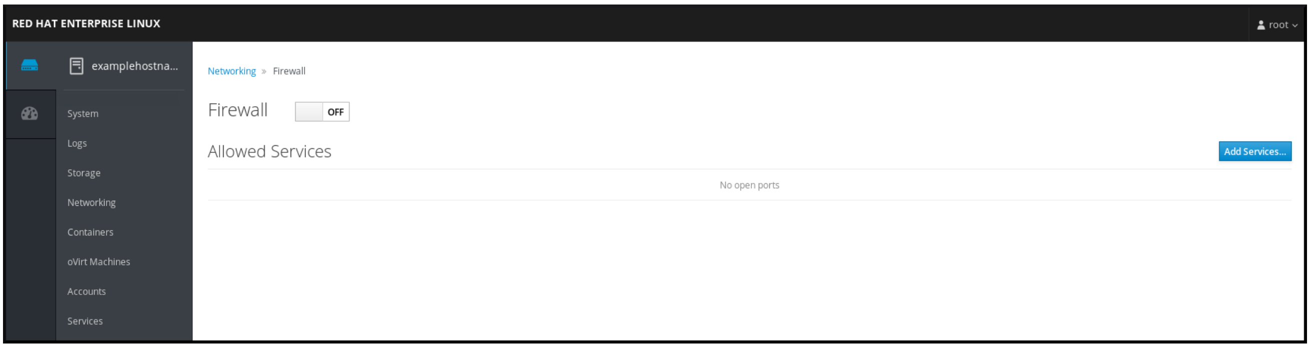
For detailed information on configuring and using firewall, see Using and configuring firewalls.

### 1.7.1.3. Managing firewall in Cockpit

In **Cockpit**, use the **Firewall** option under **Networking** to enable or disable the **firewalld** service.

By default, the **firewalld** service in **Cockpit** is enabled. To disable it, set **off** as shown below. Additionally, you can choose the services that you want to allow through firewall.

**Figure 1.9. Managing firewall in Cockpit**




## 1.7.2. Ensuring the appropriate state of SELinux

#### 1.7.2.1. What SELinux is and how it enhances system security

**Security Enhanced Linux (SELinux)** is an additional layer of system security that determines which process can access which files, directories, and ports.

**SELinux states**
**SELinux** has two possible states:

- Enabled
- Disabled

When **SELinux** is disabled, only Discretionary Access Control (DAC) rules are used.

**SELinux modes**
When **SELinux** is enabled, it can run in one of the following modes:

- Enforcing
- Permissive

Enforcing mode means that **SELinux** policies are enforced. **SELinux** denies access based on **SELinux** policy rules, and enables only the interactions that are particularly allowed. Enforcing mode is the default mode after the installation and it is also the safest **SELinux** mode.

Permissive mode means that **SELinux** policies are not enforced. **SELinux** does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode. Permissive mode is the default mode during the installation. Operating in permissive mode is also useful in some specific cases, for example if you require access to the Access Vector Cache (AVC) denials when troubleshooting problems.

For further information on **SELinux**, see Configuring and managing security.

### 1.7.3. Ensuring the Required State of SELinux

By default, **SELinux** operates in permissive mode during the installation and in enforcing mode when the installation has finished.

However, in some specific scenarios, **SELinux** might be explicitly set to permissive mode or it might even be disabled on the installed operating system. This can be set for example in the kickstart configuration.

> **IMPORTANT**
>
> Red Hat recommends to keep your system in enforcing mode.

To display the current SELinux mode, and to set the mode as needed:

**Ensuring the required state of SELinux**

1. Display the current **SELinux** mode in effect:

   ```
   ~]$ getenforce
   ```

2. If needed, switch between the SELinux modes.

The switch can be either temporary or permanent. A temporary switch is not persistent across reboots, while permanent switch is.

- To temporary switch to either enforcing or permissive mode:

```
~]# setenforce Enforcing
```

```
~]# setenforce Permissive
```

- To permanently set the **SELinux** mode, modify the *SELINUX* variable in the **`/etc/selinux/config`** configuration file.
  For example, to switch **SELinux** to enforcing mode:

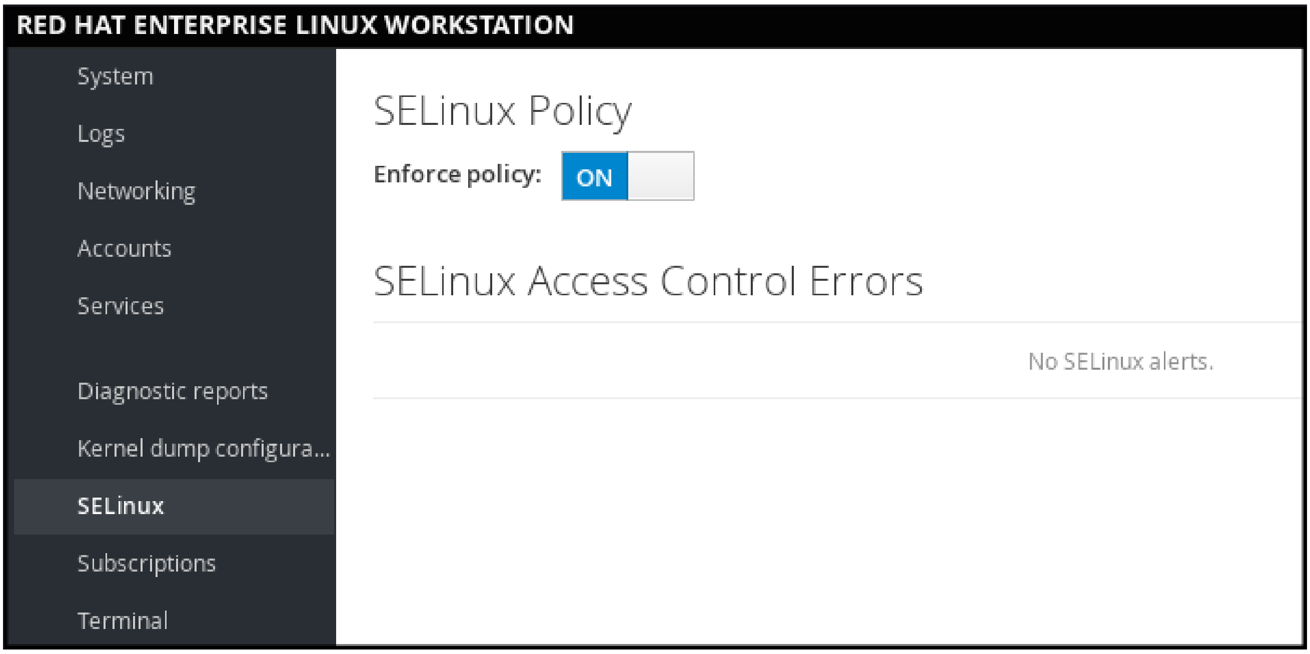
```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
```

#### 1.7.3.1. Managing SELinux in Cockpit

In **Cockpit**, use the **`SELinux`** option to turn **SELinux** enforcing policy on or off.

By default, **SELinux** enforcing policy in **Cockpit** is on, and **SELinux** operates in enforcing mode. By turning it off, you can switch **SELinux** into permissive mode. Note that such deviation from the default configuration in the /etc/sysconfig/selinux file is automatically reverted on the next boot.

**Figure 1.10. Managing SELinux in Cockpit**




### 1.7.4. Accessing system through SSH

#### 1.7.4.1. What SSH-based access is and how it enhances system security

If you want to secure your communication with another computer, you can use SSH-based authentication.

Secure Shell (SSH) is a protocol which facilitates client-server communication and allows users to log in to any host system running SSH remotely. SSH secures the connection. The client transmits its authentication information to the server using encryption, and all data sent and received during a session are transferred under the encryption.

SSH enables its users to authenticate without a password. To do so, SSH uses a private-public key scheme.

For more information about SSH, see Configuring and managing security.

### 1.7.4.2. Configuring key-based SSH access

To be able to use SSH connection, create a pair of two keys consisting of a public and a private key.

#### Creating the key files and Copying them to the Server

1. Generate a public and a private key:

   ```
   ~]$ ssh-keygen
   ```

   Both keys are stored in the **~/.ssh/** directory:

   - **~/.ssh/id_rsa.pub** - public key
   - **~/.ssh/id_rsa** - private key
     The public key does not need to be secret. It is used to verify the private key. The private key is secret. You can choose to protect the private key with the passphrase that you specify during the key generation process. With the passphrase, authentication is even more secure, but is no longer password-less. You can avoid this using the **ssh-agent** command. In this case, you will enter the passphrase only once - at the beginning of a session.

2. Copy the most recently modified public key to a remote machine you want to log into:

   ```
   ~]# ssh-copy-id USER@hostname
   ```

   As a result, you are now able to enter the system in a secure way, but without entering a password.

### 1.7.4.3. Disabling SSH root login

To increase system security, you can disable SSH access for the **root** user, which is enabled by default.

#### Disabling SSH root login

1. Access the **/etc/ssh/sshd_config** file:

   ```
   ~]# vi /etc/ssh/sshd_config
   ```

2. Change the line that reads **#PermitRootLogin yes** to:

```
PermitRootLogin no
```

3. Restart the **sshd** service:

```
~]# systemctl restart sshd
```

> **NOTE**
>
> When using **PermitRootLogin no**, the **root** user cannot login to system directly. Alternatively, the **root** user might be allowed to login, but using a password-less authentication method, typically the key-based authentication, described in Section 1.7.4.2, “Configuring key-based SSH access”. To ensure this, specify **PermitRootLogin prohibit-password** in the **/etc/ssh/sshd_config** file.

## 1.8. THE BASICS OF MANAGING USER ACCOUNTS

Red Hat Enterprise Linux is a multi-user operating system, which enables multiple users on different computers to access a single system installed on one machine. Every user operates under its own account, and managing user accounts thus represents a core element of Red Hat Enterprise Linux system administration.

### Normal and System Accounts

Normal accounts are created for users of a particular system. Such accounts can be added, removed, and modified during normal system administration.

System accounts represent a particular applications identifier on a system. Such accounts are generally added or manipulated only at software installation time, and they are not modified later.

> **WARNING**
>
> System accounts are presumed to be available locally on a system. If these accounts are configured and provided remotely, such as in the instance of an LDAP configuration, system breakage and service start failures can occur.

For system accounts, user IDs below 1000 are reserved. For normal accounts, you can use IDs starting at 1000. However, the recommended practice is to assign IDs starting at 5000. See Reserved user and group IDs for more information. The guidelines for assigning IDs can be found in the **/etc/login.defs** file:

```
# Min/max values for automatic uid selection in useradd
#
UID_MIN                  1000
UID_MAX                 60000
# System accounts
SYS_UID_MIN               201
SYS_UID_MAX               999
```

### What groups are and which purposes they can be used for

A group in an entity which ties together multiple user accounts for a common purpose, such as granting access to particular files.

### 1.8.1. Basic command-line tools to manage user accounts and groups

The most basic tasks to manage user accounts and groups, and the appropriate command-line tools, include:

- Displaying user and group IDs:

```
~]$ id
```

- Creating a new user account:

```
~]# useradd [options] user_name
```

- Assigning a new password to a user account belonging to *username*:

```
~]# passwd user_name
```

- Adding a user to a group:

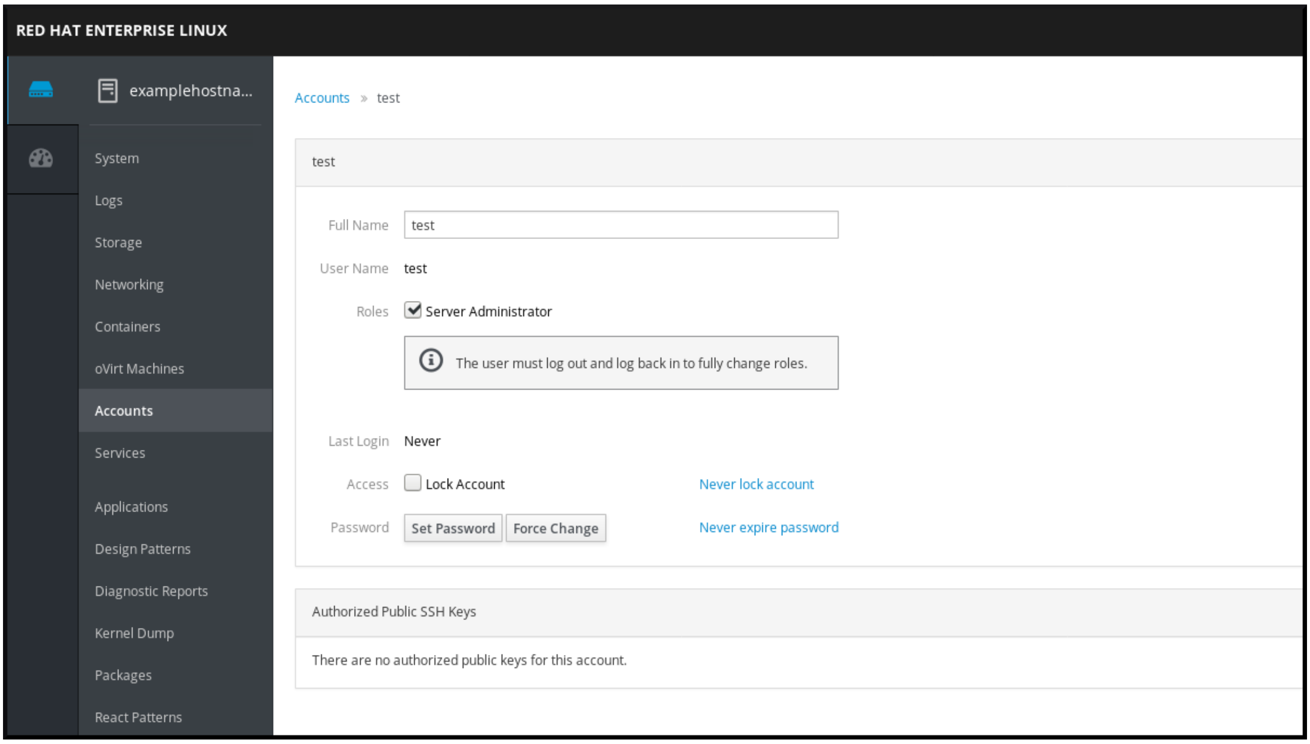
```
~]# usermod -a -G group_name user_name
```

For detailed information on managing users and groups, see Chapter 4, *Managing user and group accounts and setting permissions on files*.

### 1.8.2. Managing user accounts in Cockpit

To manage accounts in **Cockpit**, select the **`Accounts`** menu.

**Figure 1.11. Managing User Accounts in Cockpit**



## 1.9. DUMPING THE CRASHED KERNEL USING THE KDUMP MECHANISM

This section provides an introduction to the kernel crash dump mechanism, also called **kdump**, and briefly explains what **kdump** is used for in Section 1.9.1, “What kdump is and which tasks it can be used for”.

Activation of the **kdump** service is a part of the installation process, as described in Installing Red Hat Enterprise Linux 8. This section summarizes how to manually enable the **kdump** service if it is disabled after the installation in Section 1.9.2, “Ensuring that kdump is installed and enabled after the installation process”.

You can also use **Cockpit** to configure **kdump**. See Section 1.9.3, “Configuring kdump in Cockpit” for more information.

### 1.9.1. What kdump is and which tasks it can be used for

In case of a system crash, you can use the kernel crash dump mechanism called **kdump** that enables you to save the content of the system’s memory for later analysis. The **kdump** mechanism relies on the kexec system call, which can be used to boot a Linux kernel from the context of another kernel, bypass BIOS, and preserve the contents of the first kernel’s memory that would otherwise be lost.

When kernel crash occurs, **kdump** uses kexec to boot into a second kernel, a capture kernel, which resides in a reserved part of the system memory that is inaccessible to the first kernel. The second kernel captures the contents of the crashed kernel’s memory, a crash dump, and saves it.

For more detailed information about **kdump**, see Managing, monitoring and updating the kernel.

### 1.9.2. Ensuring that kdump is installed and enabled after the installation process

To ensure that **kdump** is installed and to configure it:

**Checking whether kdump is Installed and Configuring kdump**

1. To check whether **kdump** is installed on your system:

   ```
   ~]$ rpm -q kexec-tools
   ```

2. If not installed, to install **kdump**, enter as the **root** user:

   ```
   ~]# yum install kexec-tools
   ```

3. To configure **kdump**:
   Use either the command line or graphical user interface as described in Managing, monitoring and updating the kernel.

   If you need to install the graphical configuration tool:

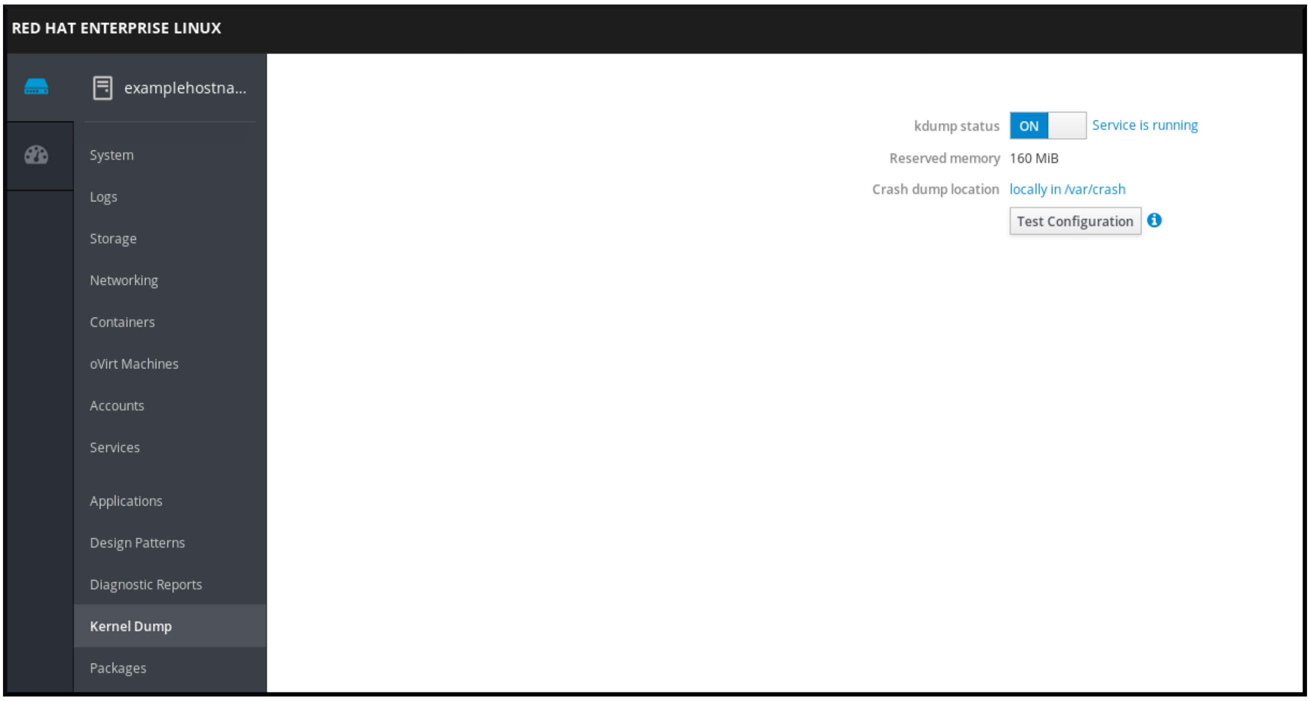
   ```
   ~]# yum install system-config-kdump
   ```

### 1.9.3. Configuring kdump in Cockpit

In **Cockpit**, select **Kernel dump configuration** to verify:

- the **kdump** status
- the amount of memory reserved for **kdump**
- the location of the crash dump files

**Figure 1.12. Configuring kdump in Cockpit**




## 1.10. PERFORMING SYSTEM RESCUE AND CREATING SYSTEM BACKUP WITH REAR

When a software or hardware failure breaks the operating system, you need a mechanism to rescue the system. It is also useful to have the system backup saved. Red Hat recommends using the Relax-and-Recover (ReaR) tool to fulfil both these needs.

### 1.10.1. What ReaR is and which tasks it can be used for

ReaR is a disaster recovery and system migration utility which enables you to create the complete rescue system. By default, this rescue system restores only the storage layout and the boot loader, but not the actual user and system files.

Additionally, certain backup software enables you to integrate ReaR for disaster recovery.

ReaR enables to perform the following tasks:

- Booting a rescue system on the new hardware
- Replicating the original storage layout
- Restoring user and system files

### 1.10.2. Quickstart to installation and configuration of ReaR

To install ReaR, enter as the `root` user:

```
~]# yum install rear genisoimage syslinux
```

Use the settings in the **/etc/rear/local.conf** file to configure ReaR.

### 1.10.3. Quickstart to creation of the rescue system with ReaR

To create the rescue system, perform the following command as the **root** user:

```
~]# rear mkrescue
```

### 1.10.4. Quickstart to configuration of ReaR with the backup software

ReaR contains a fully-integrated built-in, or internal, backup method called NETFS.

To make ReaR use its internal backup method, add these lines to the **/etc/rear/local.conf** file:

```
BACKUP=NETFS
BACKUP_URL=backup location
```

You can also configure ReaR to keep the previous backup archives when the new ones are created by adding the following line to **/etc/rear/local.conf**:

```
NETFS_KEEP_OLD_BACKUP_COPY=y
```

To make the backups incremental, meaning that only the changed files are backed up on each run, add this line to **/etc/rear/local.conf**:

```
BACKUP_TYPE=incremental
```

## 1.11. USING THE LOG FILES TO TROUBLESHOOT PROBLEMS

When troubleshooting a problem, you may appreciate the log files that contain different information and messages about the operating system. The logging system in Red Hat Enterprise Linux is based on the built-in **syslog** protocol. Particular programs use this system to record events and organize them into log files, which are useful when auditing the operating system and troubleshooting various problems.

### 1.11.1. Services handling the syslog messages

The syslog messages are handled by two services:

- The **systemd-journald** daemon
- The **rsyslog** service

The **systemd-journald** daemon collects messages from various sources and forwards them to the **rsyslog** service for further processing. The sources from which the messages are collected are:

- Kernel
- Early stages of the boot process
- Standard output and error of daemons as they start up and run

- Syslog

The **rsyslog** service sorts the syslog messages by type and priority, and writes them to the files in the **/var/log** directory, where the logs are persistently stored.

### 1.11.2. Subdirectories storing the syslog messages

The syslog messages are stored in various subdirectories under the **/var/log** directory according to what kind of messages and logs they contain:

- **var/log/messages** - all syslog messages except those mentioned below
- **var/log/secure** - security and authentication-related messages and errors
- **var/log/maillog** - mail server-related messages and errors
- **var/log/cron** - log files related to periodically executed tasks
- **var/log/boot.log** - log files related to system startup

#### 1.11.2.1. Managing the log files in Cockpit

In **Cockpit**, use the **Logs** option if you want to inspect the log files.

**Figure 1.13. Inspecting the log files in Cockpit**


## 1.12. ACCESSING RED HAT SUPPORT

To obtain support from Red Hat, use the Red Hat Customer Portal, which provides access to everything available with your subscription.

This section describes:

- Obtaining Red Hat support, in Section 1.12.1, “Obtaining Red Hat Support through Red Hat Customer Portal”
- Using the **SOS report** to troubleshoot problems, in Section 1.12.2, “Using the SOS report to troubleshoot problems”

### 1.12.1. Obtaining Red Hat Support through Red Hat Customer Portal

By using the Red Hat Customer Portal you can:

- Open a new support case
- Initiate a live chat with a Red Hat expert

- Contact a Red Hat expert by making a call or sending an email

To access the Red Hat Customer Portal, go to https://access.redhat.com.

### 1.12.2. Using the SOS report to troubleshoot problems

The **SOS report** collects configuration details, system information and diagnostic information from a Red Hat Enterprise Linux system. Attach the report when you open a support case.

Note that the **SOS report** is provided in the **sos** package, which is not installed with the default minimal installation of Red Hat Enterprise Linux.

To install the **sos** package:

```
~]# yum install sos
```

To generate an **SOS report**:

```
~]# sosreport
```

To attach the **sos report** to your support case, see the Red Hat Knowledgebase article How can I attach a file to a Red Hat support case?. Note that you will be prompted to enter the number of the support case, when attaching the **sos report**.

For more information on **SOS report**, see the Red Hat Knowledgebase article What is a sosreport and how to create one in Red Hat Enterprise Linux 4.6 and later?.

# CHAPTER 2. INSTALLING SOFTWARE WITH YUM

## 2.1. INTRODUCTION TO INSTALLING SOFTWARE ON RED HAT ENTERPRISE LINUX 8

On Red Hat Enterprise Linux 8, installing software is ensured by the new version of the **YUM** tool, which is based on the **DNF** technology.

**YUM** based on **DNF** has the following advantages over the previous **YUM v3** used on Red Hat Enterprise Linux 7:

- Increased performance
- New features available, most significantly the support for managing the modular content
- Well-designed stable API for integration with tooling

For detailed information about differences between the new **YUM** tool and the previous version **YUM v3** from Red Hat Enterprise Linux 7, see Changes in DNF CLI compared to YUM.

**NOTE**

Note, that upstream calls this tool **DNF**. As a result, some output returned by the new **YUM** tool in Red Hat Enterprise Linux 8 mentions **DNF**, and upstream documentation identifies the technology as **DNF**.

For installing software, you can use the **`yum`** command and its particular options in the same way as on Red Hat Enterprise Linux 7.

Selected yum plug-ins and utilities have been ported to the new DNF back end, and can be installed under the same names as in Red Hat Enterprise Linux 7. They also provide compatibility symlinks, so the binaries, configuration files and directories can be found in usual locations.

Note that the legacy **`Python API`** provided by **YUM v3** is no longer available. As a replacement, users are advised to migrate their plug-ins and scripts to the new **`DNF Python API`**, which is stable and fully supported. The **`DNF Python API`** is available here.

Also note that new Libdnf APIs, **`Libdnf C API`** and **`Libdnf Python API`**, are available in Red Hat Enterprise Linux 8. These new Libdnf APIs are currently unstable, and will most likely change during the Red Hat Enterprise Linux 8 life cycle.

## 2.2. INTRODUCTION TO YUM FUNCTIONALITY

**yum** is the Red Hat package manager that is able to query for information about available packages, fetch packages from repositories, install and uninstall them, and update an entire system to the latest available version. Yum performs automatic dependency resolution when updating, installing, or removing packages, and thus is able to automatically determine, fetch, and install all available dependent packages.

**yum** can be configured with new, additional repositories, or *package sources*, and also provides many plug-ins which enhance and extend its capabilities. Yum enables easy and simple package management.

**IMPORTANT**

**yum** provides secure package management by enabling Gnu Privacy Guard (GPG), also known as GnuPG, signature verification on GPG-signed packages to be turned on for all package repositories (package sources), or for individual repositories.

You can also use **yum** to your own repositories with **RPM** packages for download and installation on other machines. When possible, yum uses **parallel download** of multiple packages and metadata to speed up downloading.

**NOTE**

You must have superuser privileges in order to use **yum** to install, update or remove packages on your system. All examples here assume that you have already obtained superuser privileges by using either the **su** or **sudo** command.

## 2.3. USING YUM FOR PARTICULAR TASKS

The following sections describe how to use **yum** to achieve particular tasks.

### 2.3.1. Checking for updates and updating packages

**yum** enables you to check if your system has any updates waiting to be applied. You can list packages that need to be updated and update them as a whole, or you can update a selected individual package.

#### 2.3.1.1. Checking for updates

To see which installed packages on your system have updates available, use the following command:

```
yum check-update
```

The command shows the list of packages and their dependencies that have an update available. The output for each package consists of:

- the name of the package
- the CPU architecture the package was built for
- the version of the updated package to be installed
- the release of the updated package
- a build version, added as part of a z-stream update
- the repository in which the updated package is located.

##### 2.3.1.1.1. Updating packages

You can choose to update a single package, multiple packages, or all packages at once. If any dependencies of the package or packages you update have updates available themselves, then they are updated too.

###### 2.3.1.1.1.1. Updating a single package

To update a single package, run the following command as **root**:

```
yum update package_name
```

IMPORTANT

**yum** always **installs** a new kernel regardless of whether you are using the **yum update** or **yum install** command to apply kernel updates.

When using **RPM**, on the other hand, it is important to use the **rpm -i kernel** command which installs a new kernel instead of **rpm -u kernel** which **replaces** the current kernel.

##### 2.3.1.1.1.2. Updating a package group

To update a package group, type as **root**:

```
yum group update group_name
```

Here, replace *group_name* with a name of the package group you want to update. For more information on package groups, see Section 2.3.3, “Working with package groups”.

#### 2.3.1.1.2. Updating all packages and their dependencies

To update all packages and their dependencies, use the **yum update** command without any arguments:

```
yum update
```

#### 2.3.1.1.3. Updating security-related packages

If packages have security updates available, you can update only these packages to their latest versions. Type as **root**:

```
yum update --security
```

You can also update packages only to versions containing the latest security updates. Type as **root**:

```
yum update-minimal --security
```

## 2.3.2. Working with packages

**yum** enables you to perform a complete set of operations with software packages, including searching for packages, viewing information about them, installing and removing.

### 2.3.2.1. Searching packages

You can search all package names, descriptions and summaries by using the following command:

```
yum search term...
```

Replace *term* with a package name you want to search.

The **yum search** command is useful for searching for packages you do not know the name of, but for which you know a related term. Note that by default, **yum search** returns matches in package name and summary, which makes the search faster. Use the **yum search all** command for a more exhaustive but slower search.

#### 2.3.2.1.1. Filtering the Results

All of yum's list commands allow you to filter the results by appending one or more *glob expressions* as arguments. Global expressions are normal strings of characters which contain one or more of the wildcard characters **`*`** (which expands to match any character subset) and **?** (which expands to match any single character).

Be careful to escape the global expressions when passing them as arguments to a **yum** command, otherwise the Bash shell will interpret these expressions as *pathname expansions*, and potentially pass all files in the current directory that match the global expressions to **yum**. To make sure the global expressions are passed to **yum** as intended, use one of the following methods:

- escape the wildcard characters by preceding them with a backslash character
- double-quote or single-quote the entire global expression.

### 2.3.2.2. Listing packages

To list information on all installed **and** available packages type the following at a shell prompt:

```
yum list all
```

To list installed **and** available packages that match inserted global expressions use the following command:

```
yum list glob_expression...
```

To list all packages installed on your system use the **installed** keyword.

```
yum list installed glob_expression...
```

To list all packages in all enabled repositories that are available to install, use the command in the following form:

```
yum list available glob_expression...
```

#### 2.3.2.2.1. Listing repositories

To list the repository ID, name, and number of packages for each **enabled** repository on your system, use the following command:

```
yum repolist
```

To list more information about these repositories, use the **repoinfo** command. With this command, information including the file name, overall size, date of the last update, and base URL are displayed for each listed repository.

```
yum repoinfo
```

- To list both enabled and disabled repositories use the following command. A status column is added to the output list to show which of the repositories are enabled.

```
yum repolist all
```

By passing **`disabled`** as a first argument, you can reduce the command output to disabled repositories. For further specification you can pass the ID or name of repositories or related glob_expressions as arguments. Note that if there is an exact match between the repository ID or name and the inserted argument, this repository is listed even if it does not pass the **enabled** or **disabled** filter.

#### 2.3.2.2.2. Displaying package information

To display information about one or more packages, use the following command (global expressions are valid here as well):

```
yum info package_name...
```

Replace *package_name* with the name of the package.

#### 2.3.2.2.3. Installing packages

To install a single package and all of its non-installed dependencies, enter a command in the following form as **`root`**:

```
yum install package_name
```

You can also install multiple packages simultaneously by appending their names as arguments. To do so, type as **`root`**:

```
yum install package_name package_name...
```

If you are installing packages on a *multilib* system, such as an AMD64 or Intel 64 machine, you can specify the architecture of the package (as long as it is available in an enabled repository) by appending *.arch* to the package name:

```
yum install package_name.arch
```

You can use global expressions to quickly install multiple similarly named packages. Execute as **`root`**:

```
yum install glob_expression...
```

In addition to package names and global expressions, you can also provide file names to **`yum install`**. If you know the name of the binary you want to install, but not its package name, you can give **`yum install`** the path name. As **`root`**, type:

```
yum install /usr/sbin/named
```

Yum then searches through its package lists, finds the package which provides **`/usr/sbin/named`**, if any, and prompts you as to whether you want to install it.

As you can see in the above examples, the **`yum install`** command does not require strictly defined arguments. It can process various formats of package names and global expressions, which makes installation easier for users. On the other hand, it takes some time until **yum** parses the input correctly, especially if you specify a large number of packages. To optimize the package search, you can use the following commands to explicitly define how to parse the arguments:

```
yum install-n name
```

```
yum install-na name.architecture
```

```
yum install-nevra name-epoch:version-release.architecture
```

With **`install-n`**, **yum** interprets *name* as the exact name of the package. The **`install-na`** command tells **yum** that the subsequent argument contains the package name and architecture divided by the dot character. With **`install-nevra`**, **yum** will expect an argument in the form *name-epoch:version-release.architecture*. Similarly, you can use **`yum remove-n`**, **`yum remove-na`**, and **`yum remove-nevra`** when searching for packages to be removed.

### NOTE

If you know you want to install the package that contains the **`named`** binary, but you do not know in which **`bin/`** or **`sbin/`** directory the file is installed, use the **`yum provides`** command with a global expression.

**`yum provides "*/`*`file_name`*`"`** is a useful way to find the packages that contain *file_name*.

To install a previously-downloaded package from a local directory on your system, use the following command:

```
yum install path
```

Replace *path* with the path to the package you want to install.

Alternatively, you can use also the **`yum localinstall`** command to install a previously -nloaded package from a local directory.

#### 2.3.2.2.4. Removing packages

To uninstall a particular package, as well as any packages that depend on it, run the following command as **`root`**:

```
yum remove package_name...
```

To remove multiple packages at once by adding more package names to the command.

Similar to **`install`**, **`remove`** can take these arguments:

- package names
- global expressions
- file lists

- package provides

**WARNING**

**yum** is not able to remove a package without also removing packages which depend on it.

### 2.3.3. Working with package groups

A package group is a collection of packages that serve a common purpose, for instance **System Tools** or **Sound and Video**. Installing a package group pulls a set of dependent packages, saving time considerably. The `yum groups` command is a top-level command that covers all the operations that act on package groups in **yum**.

#### 2.3.3.1. Listing package groups

The `summary` option is used to view the number of:

- installed groups
- available groups
- available environment groups
- installed and available language groups

```
yum groups summary
```

To list all package groups from yum repositories add the `list` option. You can filter the command output by group names.

```
yum group list glob_expression...
```

Several optional arguments can be passed to this command, including `hidden` to list also groups not marked as user visible, and `ids` to list group IDs. You can add `language`, `environment`, `installed`, or `available` options to reduce the command output to a specific group type.

To list mandatory and optional packages contained in a particular group, use the following command:

```
yum group info glob_expression...
```

**NOTE**

A package group can be marked with @. When using `yum group list`, `info`, `install`, or `remove`, pass **@group_name** to specify a package group or an environmental group.

#### 2.3.3.2. Installing a package group

Each package group has a name and a group ID (*groupid*). To list the names of all package groups, and their group IDs, which are displayed in parentheses, type:

```
yum group list ids
```

You can install a package group by passing its full group name, without the groupid part, to the **`group install`** command. As **`root`**, type:

```
yum group install group_name
```

You can also install by groupid. As **`root`**, execute the following command:

```
yum group install groupid
```

You can pass the groupid or quoted group name to the **`install`** command if you prepend it with an @ symbol, which tells **`yum`** that you want to perform **`group install`**. As **`root`**, type:

```
yum install @group
```

Replace *group* with the groupid or quoted group name. The same logic appplies to environmental groups:

```
yum install @group
```

#### 2.3.3.3. Removing a package group

You can remove a package group using syntax similar to the **`install`** syntax, with use of either name of the package group or its id. As **`root`**, type:

```
yum group remove group_name
```

```
yum group remove groupid
```

Also, you can pass the groupid or quoted name to the **`remove`** command if you prepend it with an @ symbol, which tells **`yum`** that you want to perform **`group remove`**. As **`root`**, type:

```
yum remove @group
```

Replace *group* with the groupid or quoted group name. Similarly, you can replace an environmental group:

```
yum remove @group
```

## 2.4. WORKING WITH TRANSACTION HISTORY

The **`yum history`** command enables users to review information about a timeline of yum transactions, the dates and times they occurred, the number of packages affected, whether these transactions succeeded or were aborted, and if the RPM database was changed between transactions. Additionally, this command can be used to undo or redo certain transactions. All history data is stored in the **history DB in the `/var/lib/yum/history/`** directory.

### 2.4.1. Listing transactions

To display a list of the twenty most recent transactions, as **root**, either run **yum history** with no additional arguments, or type the following at a shell prompt:

```
yum history list
```

To examine a particular transaction or transactions in more detail, run the following command as **root**:

```
yum history info id...
```

The *id* argument here stands for the ID of the transaction. This argument is optional and when you omit it, **yum** automatically uses the last transaction.

### 2.4.2. Reverting and repeating transactions

Apart from reviewing the transaction history, the **yum history** command provides means to revert or repeat a selected transaction. To revert a transaction, type the following at a shell prompt as **root**:

```
yum history undo id
```

To repeat a particular transaction, as **root**, run the following command:

```
yum history redo id
```

Both commands also accept the **last** keyword to undo or repeat the latest transaction.

Note that both **yum history undo** and **yum history redo** commands only revert or repeat the steps that were performed during a transaction. If the transaction installed a new package, the **yum history undo** command will uninstall it, and if the transaction uninstalled a package the command will again install it. This command also attempts to downgrade all updated packages to their previous version, if these older packages are still available.

## 2.5. CONFIGURING YUM AND YUM REPOSITORIES

The configuration information for **yum** and related utilities is located in the **/etc/yum.conf** file. This file contains one mandatory **[main]** section, which enables you to set yum options that have global effect, and can also contain one or more **[*repository*]** sections, which allow you to set repository-specific options. However, it is recommended to define individual repositories in new or existing **.repo** files in the **/etc/yum.repos.d/** directory. The values you define in individual **[*repository*]** sections of the **/etc/yum.conf** file override values set in the **[main]** section.

This section shows how to:

- set global yum options by editing the **[main]** section of the **/etc/yum.conf** configuration file;
- set options for individual repositories by editing the **[*repository*]** sections in **/etc/yum.conf** and **.repo** files in the **/etc/yum.repos.d/** directory;
- add, enable, and disable yum repositories on the command line

### 2.5.1. Setting [main] options

The **/etc/yum.conf** configuration file contains exactly one **[main]** section, and while some of the key-value pairs in this section affect how yum operates, others affect how **yum** treats repositories.

You can add many additional options under the **[main]** section heading in **/etc/yum.conf**.

For a complete list of available **[main]** options, see the **[main] OPTIONS** section of the **yum.conf**(5) manual page.

### 2.5.2. Setting [repository] options

The **[*repository*]** sections, where *repository* is a unique repository ID such as **my_personal_repo** (spaces are not permitted), allow you to define individual yum repositories. To avoid conflicts, do not use names used by Red Hat repositories for custome repositories.

For a complete list of available **[repository]** options, see the **[repository] OPTIONS** section of the **yum.conf**(5) manual page.

### 2.5.3. Viewing the current configuration

To display the current values of global yum options (that is, the options specified in the **[main]** section of the **/etc/yum.conf** file), execute the **yum-config-manager** command with no command-line options:

```
yum-config-manager
```

### 2.5.4. Adding, enabling, and disabling a yum repository

Section 2.5.2, “Setting [repository] options” describes various options you can use to define a yum repository. This section explains how to add, enable, and disable a repository by using the **yum-config-manager** command.

#### 2.5.4.1. Adding a yum repository

To define a new repository, you can either add a **[*repository*]** section to the **/etc/yum.conf** file, or to a **.repo** file in the **/etc/yum.repos.d/** directory. All files with the **.repo** file extension in this directory are read by yum, and it is recommended to define your repositories here instead of in **/etc/yum.conf**.

> **WARNING**
>
> Obtaining and installing software packages from unverified or untrusted software sources other than Red Hat’s certificate-based **Content Delivery Network** (**CDN**) constitutes a potential security risk, and could lead to security, stability, compatibility, and maintainability issues.

**Yum** repositories commonly provide their own **.repo** file. To add such a repository to your system and enable it, run the following command as **root**:

```
yum-config-manager --add-repo repository_url
```

Here *repository_url* is a link to the **.repo** file.

#### 2.5.4.2. Enabling a yum repository

To enable a particular repository or repositories, type the following at a shell prompt as **root**:

```
yum-config-manager --enable repository...
```

Here *repository* is the unique repository ID (use **yum repolist all** to list available repository IDs).

**Disabling a yum repository**
To disable a yum repository, run the following command as **root**:

```
yum-config-manager --disable repository...
```

Here *repository* is the unique repository ID (use **yum repolist all** to list available repository IDs).

## 2.6. USING YUM PLUG-INS

**Yum** provides plug-ins that extend and enhance its operations. Certain plug-ins are installed by default. **Yum** always informs you which plug-ins, if any, are loaded and active whenever you call any **yum** command.

### 2.6.1. Enabling, configuring, and disabling yum plug-ins

To enable yum plug-ins, ensure that a line beginning with **plugins=** is present in the **[main]** section of **/etc/yum.conf**, and that its value is **1**:

```
plugins=1
```

You can disable all plug-ins by changing this line to **plugins=0**.

> **IMPORTANT**
>
> Disabling all plug-ins is not advised because certain plug-ins provide important yum services. In particular, the **product-id** and **subscription-manager** plug-ins provide support for the certificate-based **Content Delivery Network** (**CDN**). Disabling plug-ins globally is provided as a convenience option, and is generally only recommended when diagnosing a potential problem with **yum**.

Every installed plug-in has its own configuration file in the **/etc/dnf/plugins/** directory. You can set plug-in specific options in these files.

Similar to the **/etc/yum.conf** file, the plug-in configuration files always contain a **[main]** section where the **enabled=** option controls whether the plug-in is enabled when you run **yum** commands. If this option is missing, you can add it manually to the file.

If you disable all plug-ins by setting **enabled=0** in **/etc/yum.conf**, then all plug-ins are disabled regardless of whether they are enabled in their individual configuration files.

If you want to disable all yum plug-ins for a single **yum** command, use the **--noplugins** option.

If you want to disable one or more yum plug-ins for a single **yum** command, add the **--disableplugin=*plugin_name*** option to the command.

## 2.7. ADDITIONAL RESOURCES

The following sources of information provide additional resources regarding **YUM**.

### 2.7.1. Installed Documentation

- **yum**(8) — The manual page for the **yum** command-line utility provides a complete list of supported options and commands.
- **yum.conf**(5) — The manual page named **yum.conf** documents available yum configuration options.

### 2.7.2. Online Documentation

- Red Hat Customer Portal Labs — The Red Hat Customer Portal Labs includes a "Yum Repository Configuration Helper".

# CHAPTER 3. MANAGING SERVICES WITH SYSTEMD

## 3.1. INTRODUCTION TO SYSTEMD

**Systemd** is a system and service manager for Linux operating systems. It is designed to be backwards compatible with SysV init scripts, and provides a number of features such as parallel startup of system services at boot time, on-demand activation of daemons, or dependency-based service control logic. Starting with Red Hat Enterprise Linux 7, **systemd** replaced Upstart as the default init system.

**Systemd** introduces the concept of *systemd units*. These units are represented by unit configuration files located in one of the directories listed in the following table.

**Table 3.1. Systemd unit files locations**

| Directory | Description |
|---|---|
| `/usr/lib/systemd/system/` | Systemd unit files distributed with installed RPM packages. |
| `/run/systemd/system/` | Systemd unit files created at run time. This directory takes precedence over the directory with installed service unit files. |
| `/etc/systemd/system/` | Systemd unit files created by **`systemctl enable`** as well as unit files added for extending a service. This directory takes precedence over the directory with runtime unit files. |

The units encapsulate information about:

- System services
- Listening sockets
- Other objects that are relevant to the init system

For a complete list of available systemd unit types, see the follwoing table.

**Table 3.2. Available systemd unit types**

| Unit Type | File Extension | Description |
|---|---|---|
| Service unit | `.service` | A system service. |
| Target unit | `.target` | A group of systemd units. |
| Automount unit | `.automount` | A file system automount point. |
| Device unit | `.device` | A device file recognized by the kernel. |

| Unit Type | File Extension | Description |
|---|---|---|
| Mount unit | `.mount` | A file system mount point. |
| Path unit | `.path` | A file or directory in a file system. |
| Scope unit | `.scope` | An externally created process. |
| Slice unit | `.slice` | A group of hierarchically organized units that manage system processes. |
| Socket unit | `.socket` | An inter-process communication socket. |
| Swap unit | `.swap` | A swap device or a swap file. |
| Timer unit | `.timer` | A systemd timer. |

**Overriding the default systemd configuration using system.conf**
The default configuration of **systemd** is defined during the compilation and it can be found in the systemd configuration file at **`/etc/systemd/system.conf`**. Use this file if you want to deviate from those defaults and override selected default values for systemd units globally.

For example, to override the default value of the timeout limit, which is set to 90 seconds, use the **`DefaultTimeoutStartSec`** parameter to input the required value in seconds.

```
DefaultTimeoutStartSec=required value
```

For further information, see Example 3.20, “Changing the timeout limit”.

### 3.1.1. Main features

The systemd system and service manager provides the following main features:

- **Socket-based activation** — At boot time, **systemd** creates listening sockets for all system services that support this type of activation, and passes the sockets to these services as soon as they are started. This not only allows **systemd** to start services in parallel, but also makes it possible to restart a service without losing any message sent to it while it is unavailable: the corresponding socket remains accessible and all messages are queued. **Systemd** uses *socket units* for socket-based activation.

- **Bus-based activation** — System services that use D-Bus for inter-process communication can be started on-demand the first time a client application attempts to communicate with them. **Systemd** uses *D-Bus service files* for bus-based activation.

- **Device-based activation** — System services that support device-based activation can be started on-demand when a particular type of hardware is plugged in or becomes available. **Systemd** uses *device units* for device-based activation.

- **Path-based activation** — System services that support path-based activation can be started on-demand when a particular file or directory changes its state. **Systemd** uses *path units* for path-based activation.

- **Mount and automount point management** — **Systemd** monitors and manages mount and automount points. **Systemd** uses *mount units* for mount points and *automount units* for automount points.

- **Aggressive parallelization** — Because of the use of socket-based activation, **systemd** can start system services in parallel as soon as all listening sockets are in place. In combination with system services that support on-demand activation, parallel activation significantly reduces the time required to boot the system.

- **Transactional unit activation logic** — Before activating or deactivating a unit, **systemd** calculates its dependencies, creates a temporary transaction, and verifies that this transaction is consistent. If a transaction is inconsistent, **systemd** automatically attempts to correct it and remove non-essential jobs from it before reporting an error.

- **Backwards compatibility with SysV init** — **Systemd** supports SysV init scripts as described in the *Linux Standard Base Core Specification*, which eases the upgrade path to systemd service units.

### 3.1.2. Compatibility changes

The systemd system and service manager is designed to be mostly compatible with SysV init and Upstart. The following are the most notable compatibility changes with regards to Red Hat Enterprise Linux 6 system that used SysV init:

- **Systemd** has only limited support for runlevels. It provides a number of target units that can be directly mapped to these runlevels and for compatibility reasons, it is also distributed with the earlier **`runlevel`** command. Not all systemd targets can be directly mapped to runlevels, however, and as a consequence, this command might return **`N`** to indicate an unknown runlevel. It is recommended that you avoid using the **`runlevel`** command if possible.
For more information about systemd targets and their comparison with runlevels, see Section 3.3, “Working with systemd targets”.

- The **`systemctl`** utility does not support custom commands. In addition to standard commands such as **`start`**, **`stop`**, and **`status`**, authors of SysV init scripts could implement support for any number of arbitrary commands in order to provide additional functionality. For example, the init script for **`iptables`** could be executed with the **`panic`** command, which immediately enabled panic mode and reconfigured the system to start dropping all incoming and outgoing packets. This is not supported in **systemd** and the **`systemctl`** only accepts documented commands.
For more information about the **`systemctl`** utility and its comparison with the earlier **`service`** utility, see Table 3.3, “Comparison of the service utility with systemctl”.

- The **`systemctl`** utility does not communicate with services that have not been started by **systemd**. When **systemd** starts a system service, it stores the ID of its main process in order to keep track of it. The **`systemctl`** utility then uses this PID to query and manage the service. Consequently, if a user starts a particular daemon directly on the command line, **`systemctl`** is unable to determine its current status or stop it.

- **Systemd** stops only running services. Previously, when the shutdown sequence was initiated, Red Hat Enterprise Linux 6 and earlier releases of the system used symbolic links located in the **`/etc/rc0.d/`** directory to stop all available system services regardless of their status. With **systemd** , only running services are stopped on shutdown.

- System services are unable to read from the standard input stream. When **systemd** starts a service, it connects its standard input to **/dev/null** to prevent any interaction with the user.
- System services do not inherit any context (such as the **HOME** and **PATH** environment variables) from the invoking user and their session. Each service runs in a clean execution context.
- When loading a SysV init script, **systemd** reads dependency information encoded in the Linux Standard Base (LSB) header and interprets it at run time.
- All operations on service units are subject to a default timeout of 5 minutes to prevent a malfunctioning service from freezing the system. This value is hardcoded for services that are generated from initscripts and cannot be changed. However, individual configuration files can be used to specify a longer timeout value per service, see Example 3.20, “Changing the timeout limit”.

For a detailed list of compatibility changes introduced with **systemd**, see the Migration Planning Guide for Red Hat Enterprise Linux 7.

## 3.2. MANAGING SYSTEM SERVICES

Previous versions of Red Hat Enterprise Linux, which were distributed with SysV init or Upstart, used *init scripts* located in the **/etc/rc.d/init.d/** directory. These init scripts were typically written in Bash, and allowed the system administrator to control the state of services and daemons in their system. Starting with Red Hat Enterprise Linux 7, these init scripts have been replaced with *service units*.

Service units end with the **.service** file extension and serve a similar purpose as init scripts. To view, start, stop, restart, enable, or disable system services, use the **systemctl** command as described in Comparison of the service utility with systemctl, Comparison of the chkconfig utility with systemctl, and further in this section. The **service** and **chkconfig** commands are still available in the system and work as expected, but are only included for compatibility reasons and should be avoided.

**Table 3.3. Comparison of the service utility with systemctl**

| service | systemctl | Description |
|---|---|---|
| `service name start` | `systemctl start name.service` | Starts a service. |
| `service name stop` | `systemctl stop name.service` | Stops a service. |
| `service name restart` | `systemctl restart name.service` | Restarts a service. |
| `service name condrestart` | `systemctl try-restart name.service` | Restarts a service only if it is running. |
| `service name reload` | `systemctl reload name.service` | Reloads configuration. |

| service | systemctl | Description |
|---|---|---|
| `service name status` | `systemctl status name.service`<br><br>`systemctl is-active name.service` | Checks if a service is running. |
| `service --status-all` | `systemctl list-units --type service --all` | Displays the status of all services. |

**Table 3.4. Comparison of the chkconfig utility with systemctl**

| chkconfig | systemctl | Description |
|---|---|---|
| `chkconfig name on` | `systemctl enable name.service` | Enables a service. |
| `chkconfig name off` | `systemctl disable name.service` | Disables a service. |
| `chkconfig --list name` | `systemctl status name.service`<br><br>`systemctl is-enabled name.service` | Checks if a service is enabled. |
| `chkconfig --list` | `systemctl list-unit-files --type service` | Lists all services and checks if they are enabled. |
| `chkconfig --list` | `systemctl list-dependencies --after` | Lists services that are ordered to start before the specified unit. |
| `chkconfig --list` | `systemctl list-dependencies --before` | Lists services that are ordered to start after the specified unit. |

### Specifying service units

For clarity, all command examples in the rest of this section use full unit names with the **`.service`** file extension, for example:

```
~]# systemctl stop nfs-server.service
```

However, the file extension can be omitted, in which case the **`systemctl`** utility assumes the argument is a service unit. The following command is equivalent to the one above:

```
~]# systemctl stop nfs-server
```

Additionally, some units have alias names. Those names can have shorter names than units, which can be used instead of the actual unit names. To find all aliases that can be used for a particular unit, use:

```
~]# systemctl show nfs-server.service -p Names
```

**Behavior of systemctl in a chroot environment**

If you change the root directory using the **chroot** command, most **systemctl** commands refuse to perform any action. The reason for this is that the **systemd** process and the user that used the **chroot** command do not have the same view of the filesystem. This happens, for example, when **systemctl** is invoked from a **kickstart** file.

The exception to this are unit file commands such as the **systemctl enable** and **systemctl disable** commands. These commands do not need a running system and do not affect running processes, but they do affect unit files. Therefore, you can run these commands even in **chroot** environment. For example, to enable the **httpd** service on a system under the **/srv/website1/** directory:

```
~]# chroot /srv/website1
~]# systemctl enable httpd.service
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service,
pointing to /usr/lib/systemd/system/httpd.service.
```

### 3.2.1. Listing services

To list all currently loaded service units, type the following at a shell prompt:

```
systemctl list-units --type service
```

For each service unit file, this command displays its full name (**UNIT**) followed by a note whether the unit file has been loaded (**LOAD**), its high-level (**ACTIVE**) and low-level (**SUB**) unit file activation state, and a short description (**DESCRIPTION**).

By default, the **systemctl list-units** command displays only active units. If you want to list all loaded units regardless of their state, run this command with the **--all** or **-a** command line option:

```
systemctl list-units --type service --all
```

You can also list all available service units to see if they are enabled. To do so, type:

```
systemctl list-unit-files --type service
```

For each service unit, this command displays its full name (**UNIT FILE**) followed by information whether the service unit is enabled or not (**STATE**). For information on how to determine the status of individual service units, see Displaying service status.

**Example 3.1. Listing services**

To list all currently loaded service units, run the following command:

```
~]$ systemctl list-units --type service
UNIT                                 LOAD   ACTIVE SUB     DESCRIPTION
abrt-ccpp.service                    loaded active exited  Install ABRT
```

```
coredump hook
abrt-oops.service               loaded active running ABRT kernel log
watcher
abrt-vmcore.service             loaded active exited  Harvest vmcores
for ABRT
abrt-xorg.service               loaded active running ABRT Xorg log
watcher
abrtd.service                   loaded active running ABRT Automated Bug
Reporting Tool
…
systemd-vconsole-setup.service loaded active exited  Setup Virtual
Console
tog-pegasus.service             loaded active running OpenPegasus CIM
Server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of
SUB.
SUB    = The low-level unit activation state, values depend on unit
type.

46 loaded units listed. Pass --all to see loaded but inactive units,
too.
To show all installed unit files use 'systemctl list-unit-files'
```

To list all installed service unit files to determine if they are enabled, type:

```
~]$ systemctl list-unit-files --type service
UNIT FILE                                    STATE
abrt-ccpp.service                            enabled
abrt-oops.service                            enabled
abrt-vmcore.service                          enabled
abrt-xorg.service                            enabled
abrtd.service                                enabled
…
wpa_supplicant.service                       disabled
ypbind.service                               disabled

208 unit files listed.
```

### 3.2.2. Displaying service status

To display detailed information about a service unit that corresponds to a system service, type the following at a shell prompt:

```
systemctl status name.service
```

Replace *name* with the name of the service unit you want to inspect (for example, **gdm**). This command displays the name of the selected service unit followed by its short description, one or more fields described in Table 3.5, "Available service unit information", and if it is executed by the **root** user, also the most recent log entries.

**Table 3.5. Available service unit information**

| Field | Description |
|---|---|
| **`Loaded`** | Information whether the service unit has been loaded, the absolute path to the unit file, and a note whether the unit is enabled. |
| **`Active`** | Information whether the service unit is running followed by a time stamp. |
| **`Main PID`** | The PID of the corresponding system service followed by its name. |
| **`Status`** | Additional information about the corresponding system service. |
| **`Process`** | Additional information about related processes. |
| **`CGroup`** | Additional information about related Control Groups (cgroups). |

To only verify that a particular service unit is running, run the following command:

```
systemctl is-active name.service
```

Similarly, to determine whether a particular service unit is enabled, type:

```
systemctl is-enabled name.service
```

Note that both **`systemctl is-active`** and **`systemctl is-enabled`** return an exit status of **`0`** if the specified service unit is running or enabled. For information on how to list all currently loaded service units, see Listing services.

**Example 3.2. Displaying service status**

The service unit for the GNOME Display Manager is named **`gdm.service`**. To determine the current status of this service unit, type the following at a shell prompt:

```
~]# systemctl status gdm.service
gdm.service - GNOME Display Manager
   Loaded: loaded (/usr/lib/systemd/system/gdm.service; enabled)
   Active: active (running) since Thu 2013-10-17 17:31:23 CEST; 5min ago
 Main PID: 1029 (gdm)
   CGroup: /system.slice/gdm.service
           ├─1029 /usr/sbin/gdm
           ├─1037 /usr/libexec/gdm-simple-slave --display-id /org/gno…
           └─1047 /usr/bin/Xorg :0 -background none -verbose -auth /r…

Oct 17 17:31:23 localhost systemd[1]: Started GNOME Display Manager.
```

**Example 3.3. Displaying services ordered to start before a service**

To determine what services are ordered to start before the specified service, type the following at a shell prompt:

```
~]# systemctl list-dependencies --after gdm.service
gdm.service
├─dbus.socket
├─getty@tty1.service
├─livesys.service
├─plymouth-quit.service
├─system.slice
├─systemd-journald.socket
├─systemd-user-sessions.service
└─basic.target
[output truncated]
```

**Example 3.4. Displaying services ordered to start after a service**

To determine what services are ordered to start after the specified service, type the following at a shell prompt:

```
~]# systemctl list-dependencies --before gdm.service
gdm.service
├─dracut-shutdown.service
├─graphical.target
│ ├─systemd-readahead-done.service
│ ├─systemd-readahead-done.timer
│ └─systemd-update-utmp-runlevel.service
└─shutdown.target
  ├─systemd-reboot.service
  └─final.target
    └─systemd-reboot.service
```

### 3.2.3. Starting a service

To start a service unit that corresponds to a system service, type the following at a shell prompt as **`root`**:

```
systemctl start name.service
```

Replace *name* with the name of the service unit you want to start (for example, **`gdm`**). This command starts the selected service unit in the current session. For information on how to enable a service unit to be started at boot time, see Enabling a service. For information on how to determine the status of a certain service unit, see Displaying service status.

**Example 3.5. Starting a service**

The service unit for the Apache HTTP Server is named **`httpd.service`**. To activate this service unit and start the **`httpd`** daemon in the current session, run the following command as **`root`**:

```
~]# systemctl start httpd.service
```

```
-
```

### 3.2.4. Stopping a service

To stop a service unit that corresponds to a system service, type the following at a shell prompt as **root**:

```
systemctl stop name.service
```

Replace *name* with the name of the service unit you want to stop (for example, **bluetooth**). This command stops the selected service unit in the current session. For information on how to disable a service unit and prevent it from being started at boot time, see Disabling a service. For information on how to determine the status of a certain service unit, see Displaying service status.

**Example 3.6. Stopping a service**

The service unit for the **bluetoothd** daemon is named **bluetooth.service**. To deactivate this service unit and stop the **bluetoothd** daemon in the current session, run the following command as **root**:

```
~]# systemctl stop bluetooth.service
```

### 3.2.5. Restarting a service

To restart a service unit that corresponds to a system service, type the following at a shell prompt as **root**:

```
systemctl restart name.service
```

Replace *name* with the name of the service unit you want to restart (for example, **httpd**). This command stops the selected service unit in the current session and immediately starts it again. Importantly, if the selected service unit is not running, this command starts it too. To tell **systemd** to restart a service unit only if the corresponding service is already running, run the following command as **root**:

```
systemctl try-restart name.service
```

Certain system services also allow you to reload their configuration without interrupting their execution. To do so, type as **root**:

```
systemctl reload name.service
```

Note that system services that do not support this feature ignore this command altogether. For convenience, the **systemctl** command also supports the **reload-or-restart** and **reload-or-try-restart** commands that restart such services instead. For information on how to determine the status of a certain service unit, see Displaying service status.

**Example 3.7. Restarting a service**

In order to prevent users from encountering unnecessary error messages or partially rendered web pages, the Apache HTTP Server allows you to edit and reload its configuration without the need to restart it and interrupt actively processed requests. To do so, type the following at a shell prompt as **root**:

```
~]# systemctl reload httpd.service
```

### 3.2.6. Enabling a service

To configure a service unit that corresponds to a system service to be automatically started at boot time, type the following at a shell prompt as **root**:

```
systemctl enable name.service
```

Replace *name* with the name of the service unit you want to enable (for example, **httpd**). This command reads the **[Install]** section of the selected service unit and creates appropriate symbolic links to the **/usr/lib/systemd/system/*name*.service** file in the **/etc/systemd/system/** directory and its subdirectories. This command does not, however, rewrite links that already exist. If you want to ensure that the symbolic links are re-created, use the following command as **root**:

```
systemctl reenable name.service
```

This command disables the selected service unit and immediately enables it again. For information on how to determine whether a certain service unit is enabled to start at boot time, see Displaying service status. For information on how to start a service in the current session, see Starting a service.

**Example 3.8. Enabling a service**

To configure the Apache HTTP Server to start automatically at boot time, run the following command as **root**:

```
~]# systemctl enable httpd.service
Created symlink from /etc/systemd/system/multi-
user.target.wants/httpd.service to
/usr/lib/systemd/system/httpd.service.
```

### 3.2.7. Disabling a service

To prevent a service unit that corresponds to a system service from being automatically started at boot time, type the following at a shell prompt as **root**:

```
systemctl disable name.service
```

Replace *name* with the name of the service unit you want to disable (for example, **bluetooth**). This command reads the **[Install]** section of the selected service unit and removes appropriate symbolic links to the **/usr/lib/systemd/system/*name*.service** file from the **/etc/systemd/system/** directory and its subdirectories. In addition, you can mask any service unit to prevent it from being started manually or by another service. To do so, run the following command as **root**:

```
systemctl mask name.service
```

This command replaces the **/etc/systemd/system/*name*.service** file with a symbolic link to **/dev/null**, rendering the actual unit file inaccessible to **systemd**. To revert this action and unmask a service unit, type as **root**:

```
systemctl unmask name.service
```

For information on how to determine whether a certain service unit is enabled to start at boot time, see Displaying service status. For information on how to stop a service in the current session, see Stopping a service.

**Example 3.9. Disabling a service**

Example 3.6, "Stopping a service" illustrates how to stop the **bluetooth.service** unit in the current session. To prevent this service unit from starting at boot time, type the following at a shell prompt as **root**:

```
~]# systemctl disable bluetooth.service
Removed symlink
/etc/systemd/system/bluetooth.target.wants/bluetooth.service.
Removed symlink /etc/systemd/system/dbus-org.bluez.service.
```

### 3.2.8. Starting a conflicting service

In **systemd**, positive and negative dependencies between services exist. Starting particular service may require starting one or more other services (positive dependency) or stopping one or more services (negative dependency).

When you attempt to start a new service, **systemd** resolves all dependencies automatically. Note that this is done without explicit notification to the user. If you are already running a service, and you attempt to start another service with a negative dependency, the first service is automatically stopped.

For example, if you are running the **postfix** service, and you try to start the **sendmail** service, **systemd** first automatically stops **postfix**, because these two services are conflicting and cannot run on the same port.

## 3.3. WORKING WITH SYSTEMD TARGETS

Previous versions of Red Hat Enterprise Linux, which were distributed with SysV init or Upstart, implemented a predefined set of *runlevels* that represented specific modes of operation. These runlevels were numbered from 0 to 6 and were defined by a selection of system services to be run when a particular runlevel was enabled by the system administrator. Starting with Red Hat Enterprise Linux 7, the concept of runlevels has been replaced with *systemd targets*.

Systemd targets are represented by *target units*. Target units end with the **.target** file extension and their only purpose is to group together other systemd units through a chain of dependencies. For example, the **graphical.target** unit, which is used to start a graphical session, starts system services such as the GNOME Display Manager (**gdm.service**) or Accounts Service (**accounts-daemon.service**) and also activates the **multi-user.target** unit. Similarly, the **multi-user.target** unit starts other essential system services such as NetworkManager (**NetworkManager.service**) or D-Bus (**dbus.service**) and activates another target unit named **basic.target**.

Red Hat Enterprise Linux 7 was distributed with a number of predefined targets that are more or less similar to the standard set of runlevels from the previous releases of this system. For compatibility reasons, it also provides aliases for these targets that directly map them to SysV runlevels. Table 3.6, "Comparison of SysV runlevels with systemd targets" provides a complete list of SysV runlevels and their corresponding systemd targets.

**Table 3.6. Comparison of SysV runlevels with systemd targets**

| Runlevel | Target Units | Description |
|---|---|---|
| `0` | `runlevel0.target`, `poweroff.target` | Shut down and power off the system. |
| `1` | `runlevel1.target`, `rescue.target` | Set up a rescue shell. |
| `2` | `runlevel2.target`, `multi-user.target` | Set up a non-graphical multi-user system. |
| `3` | `runlevel3.target`, `multi-user.target` | Set up a non-graphical multi-user system. |
| `4` | `runlevel4.target`, `multi-user.target` | Set up a non-graphical multi-user system. |
| `5` | `runlevel5.target`, `graphical.target` | Set up a graphical multi-user system. |
| `6` | `runlevel6.target`, `reboot.target` | Shut down and reboot the system. |

To view, change, or configure systemd targets, use the **`systemctl`** utility as described in Table 3.7, "Comparison of SysV init commands with systemctl" and in the sections below. The **`runlevel`** and **`telinit`** commands are still available in the system and work as expected, but are only included for compatibility reasons and should be avoided.

**Table 3.7. Comparison of SysV init commands with systemctl**

| Old Command | New Command | Description |
|---|---|---|
| `runlevel` | `systemctl list-units --type target` | Lists currently loaded target units. |
| `telinit` *`runlevel`* | `systemctl isolate` *`name`*`.target` | Changes the current target. |

### 3.3.1. Viewing the default target

To determine which target unit is used by default, run the following command:

```
systemctl get-default
```

This command resolves the symbolic link located at **/etc/systemd/system/default.target** and displays the result.

**Example 3.10. Viewing the default target**

To display the default target unit, type:

```
~]$ systemctl get-default
graphical.target
```

### 3.3.2. Viewing the current target

To list all currently loaded target units, type the following command at a shell prompt:

```
systemctl list-units --type target
```

For each target unit, this commands displays its full name (**UNIT**) followed by a note whether the unit has been loaded (**LOAD**), its high-level (**ACTIVE**) and low-level (**SUB**) unit activation state, and a short description (**DESCRIPTION**).

By default, the **systemctl list-units** command displays only active units. If you want to list all loaded units regardless of their state, run this command with the **--all** or **-a** command line option:

```
systemctl list-units --type target --all
```

**Example 3.11. Viewing the current target**

To list all currently loaded target units, run:

```
~]$ systemctl list-units --type target
UNIT                  LOAD   ACTIVE SUB    DESCRIPTION
basic.target          loaded active active Basic System
cryptsetup.target     loaded active active Encrypted Volumes
getty.target          loaded active active Login Prompts
graphical.target      loaded active active Graphical Interface
local-fs-pre.target   loaded active active Local File Systems (Pre)
local-fs.target       loaded active active Local File Systems
multi-user.target     loaded active active Multi-User System
network.target        loaded active active Network
paths.target          loaded active active Paths
remote-fs.target      loaded active active Remote File Systems
sockets.target        loaded active active Sockets
sound.target          loaded active active Sound Card
spice-vdagentd.target loaded active active Agent daemon for Spice guests
swap.target           loaded active active Swap
sysinit.target        loaded active active System Initialization
time-sync.target      loaded active active System Time Synchronized
timers.target         loaded active active Timers

LOAD   = Reflects whether the unit definition was properly loaded.
```

```
ACTIVE = The high-level unit activation state, i.e. generalization of
SUB.
SUB    = The low-level unit activation state, values depend on unit
type.

17 loaded units listed. Pass --all to see loaded but inactive units,
too.
To show all installed unit files use 'systemctl list-unit-files'.
```

### 3.3.3. Changing the default target

To configure the system to use a different target unit by default, type the following at a shell prompt as **root**:

```
systemctl set-default name.target
```

Replace *name* with the name of the target unit you want to use by default (for example, **multi-user**). This command replaces the **/etc/systemd/system/default.target** file with a symbolic link to **/usr/lib/systemd/system/*name*.target**, where *name* is the name of the target unit you want to use.

**Example 3.12. Changing the default target**

To configure the system to use the **multi-user.target** unit by default, run the following command as **root**:

```
~]# systemctl set-default multi-user.target
rm '/etc/systemd/system/default.target'
ln -s '/usr/lib/systemd/system/multi-user.target'
'/etc/systemd/system/default.target'
```

### 3.3.4. Changing the current target

To change to a different target unit in the current session, type the following at a shell prompt as **root**:

```
systemctl isolate name.target
```

Replace *name* with the name of the target unit you want to use (for example, **multi-user**). This command starts the target unit named *name* and all dependent units, and immediately stops all others.

**Example 3.13. Changing the current target**

To turn off the graphical user interface and change to the **multi-user.target** unit in the current session, run the following command as **root**:

```
~]# systemctl isolate multi-user.target
```

### 3.3.5. Changing to rescue mode

*Rescue mode* provides a convenient single-user environment and allows you to repair your system in situations when it is unable to complete a regular booting process. In rescue mode, the system attempts to mount all local file systems and start some important system services, but it does not activate network interfaces or allow more users to be logged into the system at the same time. Rescue mode requires the root password.

To change the current target and enter rescue mode in the current session, type the following at a shell prompt as **root**:

```
systemctl rescue
```

This command is similar to **systemctl isolate rescue.target**, but it also sends an informative message to all users that are currently logged into the system. To prevent **systemd** from sending this message, run this command with the **--no-wall** command line option:

```
systemctl --no-wall rescue
```

For information on how to enter emergency mode, see Section 3.3.6, “Changing to emergency mode”.

**Example 3.14. Changing to rescue mode**

To enter rescue mode in the current session, run the following command as **root**:

```
~]# systemctl rescue

Broadcast message from root@localhost on pts/0 (Fri 2013-10-25 18:23:15
CEST):

The system is going down to rescue mode NOW!
```

### 3.3.6. Changing to emergency mode

*Emergency mode* provides the most minimal environment possible and allows you to repair your system even in situations when the system is unable to enter rescue mode. In emergency mode, the system mounts the root file system only for reading, does not attempt to mount any other local file systems, does not activate network interfaces, and only starts a few essential services. Emergency mode requires the root password.

To change the current target and enter emergency mode, type the following at a shell prompt as **root**:

```
systemctl emergency
```

This command is similar to **systemctl isolate emergency.target**, but it also sends an informative message to all users that are currently logged into the system. To prevent **systemd** from sending this message, run this command with the **--no-wall** command line option:

```
systemctl --no-wall emergency
```

For information on how to enter rescue mode, see Section 3.3.5, “Changing to rescue mode”.

**Example 3.15. Changing to emergency mode**

> To enter emergency mode without sending a message to all users that are currently logged into the system, run the following command as **root**:
>
> ```
> ~]# systemctl --no-wall emergency
> ```

## 3.4. SHUTTING DOWN, SUSPENDING, AND HIBERNATING THE SYSTEM

In Red Hat Enterprise Linux 7, the **systemctl** utility replaced a number of power management commands used in previous versions of Red Hat Enterprise Linux. The commands listed in Table 3.8, "Comparison of power management commands with systemctl" are still available in the system for compatibility reasons, but it is advised that you use **systemctl** when possible.

**Table 3.8. Comparison of power management commands with systemctl**

| Old Command | New Command | Description |
|---|---|---|
| **halt** | **systemctl halt** | Halts the system. |
| **poweroff** | **systemctl poweroff** | Powers off the system. |
| **reboot** | **systemctl reboot** | Restarts the system. |
| **pm-suspend** | **systemctl suspend** | Suspends the system. |
| **pm-hibernate** | **systemctl hibernate** | Hibernates the system. |
| **pm-suspend-hybrid** | **systemctl hybrid-sleep** | Hibernates and suspends the system. |

### 3.4.1. Shutting down the system

The **systemctl** utility provides commands for shutting down the system, however the traditional **shutdown** command is also supported. Although the **shutdown** command will call the **systemctl** utility to perform the shutdown, it has an advantage in that it also supports a time argument. This is particularly useful for scheduled maintenance and to allow more time for users to react to the warning that a system shutdown has been scheduled. The option to cancel the shutdown can also be an advantage.

#### Using systemctl commands

To shut down the system and power off the machine, type the following at a shell prompt as **root**:

```
systemctl poweroff
```

To shut down and halt the system without powering off the machine, run the following command as **root**:

```
systemctl halt
```

By default, running either of these commands causes **systemd** to send an informative message to all users that are currently logged into the system. To prevent **systemd** from sending this message, run the selected command with the **`--no-wall`** command line option, for example:

```
systemctl --no-wall poweroff
```

### Using the shutdown command

To shut down the system and power off the machine at a certain time, use a command in the following format as **`root`**:

```
shutdown --poweroff hh:mm
```

Where *hh:mm* is the time in 24 hour clock format. The **`/run/nologin`** file is created 5 minutes before system shutdown to prevent new logins. When a time argument is used, an optional message, the *wall message*, can be appended to the command.

To shut down and halt the system after a delay, without powering off the machine, use a command in the following format as **`root`**:

```
shutdown --halt +m
```

Where *+m* is the delay time in minutes. The **`now`** keyword is an alias for **`+0`**.

A pending shutdown can be canceled by the **`root`** user as follows:

```
shutdown -c
```

See the **`shutdown(8)`** manual page for further command options.

## 3.4.2. Restarting the system

To restart the system, run the following command as **`root`**:

```
systemctl reboot
```

By default, this command causes **systemd** to send an informative message to all users that are currently logged into the system. To prevent **systemd** from sending this message, run this command with the **`--no-wall`** command line option:

```
systemctl --no-wall reboot
```

## 3.4.3. Suspending the system

To suspend the system, type the following at a shell prompt as **`root`**:

```
systemctl suspend
```

This command saves the system state in RAM and with the exception of the RAM module, powers off most of the devices in the machine. When you turn the machine back on, the system then restores its state from RAM without having to boot again. Because the system state is saved in RAM and not on the hard disk, restoring the system from suspend mode is significantly faster than restoring it from hibernation, but as a consequence, a suspended system state is also vulnerable to power outages.

For information on how to hibernate the system, see Section 3.4.4, “Hibernating the system”.

### 3.4.4. Hibernating the system

To hibernate the system, type the following at a shell prompt as **root**:

```
systemctl hibernate
```

This command saves the system state on the hard disk drive and powers off the machine. When you turn the machine back on, the system then restores its state from the saved data without having to boot again. Because the system state is saved on the hard disk and not in RAM, the machine does not have to maintain electrical power to the RAM module, but as a consequence, restoring the system from hibernation is significantly slower than restoring it from suspend mode.

To hibernate and suspend the system, run the following command as **root**:

```
systemctl hybrid-sleep
```

For information on how to suspend the system, see Section 3.4.3, “Suspending the system”.

## 3.5. WORKING WITH SYSTEMD UNIT FILES

A unit file contains configuration directives that describe the unit and define its behavior. Several **systemctl** commands work with unit files in the background. To make finer adjustments, system administrator must edit or create unit files manually. Table 3.1, “Systemd unit files locations” lists three main directories where unit files are stored on the system, the **/etc/systemd/system/** directory is reserved for unit files created or customized by the system administrator.

Unit file names take the following form:

```
unit_name.type_extension
```

Here, *unit_name* stands for the name of the unit and *type_extension* identifies the unit type, see Table 3.2, “Available systemd unit types” for a complete list of unit types. For example, there usually is **sshd.service** as well as **sshd.socket** unit present on your system.

Unit files can be supplemented with a directory for additional configuration files. For example, to add custom configuration options to **sshd.service**, create the **sshd.service.d/custom.conf** file and insert additional directives there. For more information on configuration directories, see Section 3.5.4, “Modifying existing unit files”.

Also, the **sshd.service.wants/** and **sshd.service.requires/** directories can be created. These directories contain symbolic links to unit files that are dependencies of the **sshd** service. The symbolic links are automatically created either during installation according to [Install] unit file options (see Table 3.11, “Important [Install] section options”) or at runtime based on [Unit] options (see Table 3.9, “Important [Unit] section options”). It is also possible to create these directories and symbolic links manually.

Many unit file options can be set using the so called **unit specifiers** – wildcard strings that are dynamically replaced with unit parameters when the unit file is loaded. This enables creation of generic unit files that serve as templates for generating instantiated units. See Section 3.5.5, “Working with instantiated units” for details.

### 3.5.1. Understanding the unit file structure

Unit files typically consist of three sections:

- [Unit] — contains generic options that are not dependent on the type of the unit. These options provide unit description, specify the unit's behavior, and set dependencies to other units. For a list of most frequently used [Unit] options, see Table 3.9, "Important [Unit] section options".

- [*unit type*] — if a unit has type-specific directives, these are grouped under a section named after the unit type. For example, service unit files contain the [Service] section, see Table 3.10, "Important [Service] section options" for most frequently used [Service] options.

- [Install] — contains information about unit installation used by **`systemctl enable`** and **`disable`** commands, see Table 3.11, "Important [Install] section options" for a list of [Install] options.

**Table 3.9. Important [Unit] section options**

| Option[a] section, see the `systemd.unit(5)` manual page.] | Description |
|---|---|
| **`Description`** | A meaningful description of the unit. This text is displayed for example in the output of the **`systemctl status`** command. |
| **`Documentation`** | Provides a list of URIs referencing documentation for the unit. |
| **`After`**[b] | Defines the order in which units are started. The unit starts only after the units specified in **`After`** are active. Unlike **`Requires`**, **`After`** does not explicitly activate the specified units. The **`Before`** option has the opposite functionality to **`After`**. |
| **`Requires`** | Configures dependencies on other units. The units listed in **`Requires`** are activated together with the unit. If any of the required units fail to start, the unit is not activated. |
| **`Wants`** | Configures weaker dependencies than **`Requires`**. If any of the listed units does not start successfully, it has no impact on the unit activation. This is the recommended way to establish custom unit dependencies. |
| **`Conflicts`** | Configures negative dependencies, an opposite to **`Requires`**. |

[a] For a complete list of options configurable in the [Unit

[b] In most cases, it is sufficient to set only the ordering dependencies with **`After`** and **`Before`** unit file options. If you also set a requirement dependency with **`Wants`** (recommended) or **`Requires`**, the ordering dependency still needs to be specified. That is because ordering and requirement dependencies work independently from each other.

**Table 3.10. Important [Service] section options**

| Option[a] section, see the `systemd.service(5)` manual page.] | Description |
|---|---|
| **`Type`** | Configures the unit process startup type that affects the functionality of **`ExecStart`** and related options. One of:<br><br>* **`simple`** – The default value. The process started with **`ExecStart`** is the main process of the service.<br><br>* **`forking`** – The process started with **`ExecStart`** spawns a child process that becomes the main process of the service. The parent process exits when the startup is complete.<br><br>* **`oneshot`** – This type is similar to **`simple`**, but the process exits before starting consequent units.<br><br>* **`dbus`** – This type is similar to **`simple`**, but consequent units are started only after the main process gains a D-Bus name.<br><br>* **`notify`** – This type is similar to **`simple`**, but consequent units are started only after a notification message is sent via the sd_notify() function.<br><br>* **`idle`** – similar to **`simple`**, the actual execution of the service binary is delayed until all jobs are finished, which avoids mixing the status output with shell output of services. |
| **`ExecStart`** | Specifies commands or scripts to be executed when the unit is started. **`ExecStartPre`** and **`ExecStartPost`** specify custom commands to be executed before and after **`ExecStart`**. **`Type=oneshot`** enables specifying multiple custom commands that are then executed sequentially. |
| **`ExecStop`** | Specifies commands or scripts to be executed when the unit is stopped. |
| **`ExecReload`** | Specifies commands or scripts to be executed when the unit is reloaded. |
| **`Restart`** | With this option enabled, the service is restarted after its process exits, with the exception of a clean stop by the **`systemctl`** command. |
| **`RemainAfterExit`** | If set to True, the service is considered active even when all its processes exited. Default value is False. This option is especially useful if **`Type=oneshot`** is configured. |

| Option[a] section, see the `systemd.service(5)` manual page.] | Description |
|---|---|
| [a] For a complete list of options configurable in the [Service | |

### Table 3.11. Important [Install] section options

| Option[a] section, see the `systemd.unit(5)` manual page.] | Description |
|---|---|
| **`Alias`** | Provides a space-separated list of additional names for the unit. Most **`systemctl`** commands, excluding **`systemctl enable`**, can use aliases instead of the actual unit name. |
| **`RequiredBy`** | A list of units that depend on the unit. When this unit is enabled, the units listed in **`RequiredBy`** gain a **`Require`** dependency on the unit. |
| **`WantedBy`** | A list of units that weakly depend on the unit. When this unit is enabled, the units listed in **`WantedBy`** gain a **`Want`** dependency on the unit. |
| **`Also`** | Specifies a list of units to be installed or uninstalled along with the unit. |
| **`DefaultInstance`** | Limited to instantiated units, this option specifies the default instance for which the unit is enabled. See Section 3.5.5, "Working with instantiated units" |
| [a] For a complete list of options configurable in the [Install | |

A whole range of options that can be used to fine tune the unit configuration. The below example shows a service unit installed on the system. Moreover, unit file options can be defined in a way that enables dynamic creation of units as described in Working with instantiated units.

**Example 3.16. postfix.service unit file**

What follows is the content of the **`/usr/lib/systemd/system/postfix.service`** unit file as currently provided by the **postfix** package:

```
[Unit]
Description=Postfix Mail Transport Agent
After=syslog.target network.target
Conflicts=sendmail.service exim.service

[Service]
Type=forking
PIDFile=/var/spool/postfix/pid/master.pid
```

```
EnvironmentFile=-/etc/sysconfig/network
ExecStartPre=-/usr/libexec/postfix/aliasesdb
ExecStartPre=-/usr/libexec/postfix/chroot-update
ExecStart=/usr/sbin/postfix start
ExecReload=/usr/sbin/postfix reload
ExecStop=/usr/sbin/postfix stop

[Install]
WantedBy=multi-user.target
```

The [Unit] section describes the service, specifies the ordering dependencies, as well as conflicting units. In [Service], a sequence of custom scripts is specified to be executed during unit activation, on stop, and on reload. **EnvironmentFile** points to the location where environment variables for the service are defined, **PIDFile** specifies a stable PID for the main process of the service. Finally, the [Install] section lists units that depend on the service.

### 3.5.2. Creating custom unit files

There are several use cases for creating unit files from scratch: you could run a custom daemon, create a second instance of some existing service (as in Creating a second instance of the sshd service), or import a SysV init script (more in Converting SysV init scripts to unit files). On the other hand, if you intend just to modify or extend the behavior of an existing unit, use the instructions from Modifying existing unit files. The following procedure describes the general process of creating a custom service:

1. Prepare the executable file with the custom service. This can be a custom-created script, or an executable delivered by a software provider. If required, prepare a PID file to hold a constant PID for the main process of the custom service. It is also possible to include environment files to store shell variables for the service. Make sure the source script is executable (by executing the **chmod a+x**) and is not interactive.

2. Create a unit file in the **/etc/systemd/system/** directory and make sure it has correct file permissions. Execute as **root**:

   ```
   touch /etc/systemd/system/name.service

   chmod 664 /etc/systemd/system/name.service
   ```

   Replace *name* with a name of the service to be created. Note that file does not need to be executable.

3. Open the ***name*.service** file created in the previous step, and add the service configuration options. There is a variety of options that can be used depending on the type of service you wish to create, see Section 3.5.1, "Understanding the unit file structure". The following is an example unit configuration for a network-related service:

   ```
   [Unit]
   Description=service_description
   After=network.target

   [Service]
   ExecStart=path_to_executable
   Type=forking
   PIDFile=path_to_pidfile
   ```

```
[Install]
WantedBy=default.target
```

Where:

- *service_description* is an informative description that is displayed in journal log files and in the output of the **`systemctl status`** command.
- the **`After`** setting ensures that the service is started only after the network is running. Add a space-separated list of other relevant services or targets.
- *path_to_executable* stands for the path to the actual service executable.
- **`Type=forking`** is used for daemons that make the fork system call. The main process of the service is created with the PID specified in *path_to_pidfile*. Find other startup types in Table 3.10, "Important [Service] section options".
- **`WantedBy`** states the target or targets that the service should be started under. Think of these targets as of a replacement of the older concept of runlevels, see Section 3.3, "Working with systemd targets" for details.

4. Notify **systemd** that a new ***`name`*`.service`** file exists by executing the following command as **`root`**:

```
systemctl daemon-reload

systemctl start name.service
```

> **WARNING**
>
> Always run the **`systemctl daemon-reload`** command after creating new unit files or modifying existing unit files. Otherwise, the **`systemctl start`** or **`systemctl enable`** commands could fail due to a mismatch between states of **systemd** and actual service unit files on disk. Note, that on systems with a large number of units this can take a long time, as the state of each unit has to be serialized and subsequently deserialized during the reload.

**Example 3.17. Creating the emacs.service file**

When using the **Emacs** text editor, it is often faster and more convenient to have it running in the background instead of starting a new instance of the program whenever editing a file. The following steps show how to create a unit file for Emacs, so that it can be handled like a service.

1. Create a unit file in the **`/etc/systemd/system/`** directory and make sure it has the correct file permissions. Execute as **`root`**:

```
~]# touch /etc/systemd/system/emacs.service
```

```
~]# chmod 664 /etc/systemd/system/emacs.service
```

2. Add the following content to the file:

```
[Unit]
Description=Emacs: the extensible, self-documenting text editor

[Service]
Type=forking
ExecStart=/usr/bin/emacs --daemon
ExecStop=/usr/bin/emacsclient --eval "(kill-emacs)"
Environment=SSH_AUTH_SOCK=%t/keyring/ssh
Restart=always

[Install]
WantedBy=default.target
```

With the above configuration, the **/usr/bin/emacs** executable is started in daemon mode on service start. The SSH_AUTH_SOCK environment variable is set using the "%t" unit specifier that stands for the runtime directory. The service also restarts the emacs process if it exits unexpectedly.

3. Execute the following commands to reload the configuration and start the custom service:

```
~]# systemctl daemon-reload
```

```
~]# systemctl start emacs.service
```

As the editor is now registered as a systemd service, you can use all standard **systemctl** commands. For example, run **systemctl status emacs** to display the editor's status or **systemctl enable emacs** to make the editor start automatically on system boot.

**Example 3.18. Creating a second instance of the sshd service**

System Administrators often need to configure and run multiple instances of a service. This is done by creating copies of the original service configuration files and modifying certain parameters to avoid conflicts with the primary instance of the service. The following procedure shows how to create a second instance of the **sshd** service:

1. Create a copy of the **sshd_config** file that will be used by the second daemon:

```
~]# cp /etc/ssh/sshd{,-second}_config
```

2. Edit the **sshd-second_config** file created in the previous step to assign a different port number and PID file to the second daemon:

```
Port 22220
PidFile /var/run/sshd-second.pid
```

See the **sshd_config**(5) manual page for more information on **Port** and **PidFile** options. Make sure the port you choose is not in use by any other service. The PID file does not have to exist before running the service, it is generated automatically on service start.

3. Create a copy of the systemd unit file for the **sshd** service:

```
~]# cp /usr/lib/systemd/system/sshd.service
/etc/systemd/system/sshd-second.service
```

4. Alter the **sshd-second.service** created in the previous step as follows:

   a. Modify the **Description** option:

   ```
   Description=OpenSSH server second instance daemon
   ```

   b. Add sshd.service to services specified in the **After** option, so that the second instance starts only after the first one has already started:

   ```
   After=syslog.target network.target auditd.service sshd.service
   ```

   c. The first instance of sshd includes key generation, therefore remove the **ExecStartPre=/usr/sbin/sshd-keygen** line.

   d. Add the **-f /etc/ssh/sshd-second_config** parameter to the **sshd** command, so that the alternative configuration file is used:

   ```
   ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config
   $OPTIONS
   ```

   e. After the above modifications, the sshd-second.service should look as follows:

   ```
   [Unit]
   Description=OpenSSH server second instance daemon
   After=syslog.target network.target auditd.service sshd.service

   [Service]
   EnvironmentFile=/etc/sysconfig/sshd
   ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config
   $OPTIONS
   ExecReload=/bin/kill -HUP $MAINPID
   KillMode=process
   Restart=on-failure
   RestartSec=42s

   [Install]
   WantedBy=multi-user.target
   ```

5. If using SELinux, add the port for the second instance of sshd to SSH ports, otherwise the second instance of sshd will be rejected to bind to the port:

```
~]# semanage port -a -t ssh_port_t -p tcp 22220
```

6. Enable sshd-second.service, so that it starts automatically upon boot:

```
~]# systemctl enable sshd-second.service
```

> Verify if the sshd-second.service is running by using the **systemctl status** command. Also, verify if the port is enabled correctly by connecting to the service:
>
> ```
> ~]$ ssh -p 22220 user@server
> ```
>
> If the firewall is in use, make sure that it is configured appropriately in order to allow connections to the second instance of sshd.

**To learn how to properly choose a target for ordering and dependencies of your custom unit files, see the following articles**

- How to write a service unit file which enforces that particular services have to be started
- How to decide what dependencies a systemd service unit definition should have

Additional information with some real-world examples of cases triggered by the ordering and dependencies in a unit file is available in Red Hat Knowledgebase article Is there any useful information about writing unit files?

If you want to set limits for services started by **systemd**, see the Red Hat Knowledgebase article How to set limits for services in RHEL 7 and systemd. These limits need to be set in the service's unit file. Note that **systemd** ignores limits set in the **/etc/security/limits.conf** and **/etc/security/limits.d/*.conf** configuration files. The limits defined in these files are set by PAM when starting a login session, but daemons started by **systemd** do not use PAM login sessions.

### 3.5.3. Converting SysV init scripts to unit files

Before taking time to convert a SysV init script to a unit file, make sure that the conversion was not already done elsewhere. All core services installed on Red Hat Enterprise Linux come with default unit files, and the same applies for many third-party software packages.

Converting an init script to a unit file requires analyzing the script and extracting the necessary information from it. Based on this data you can create a unit file. As init scripts can vary greatly depending on the type of the service, you might need to employ more configuration options for translation than outlined in this chapter. Note that some levels of customization that were available with init scripts are no longer supported by systemd units.

The majority of information needed for conversion is provided in the script's header. The following example shows the opening section of the init script used to start the **postfix** service on Red Hat Enterprise Linux 6:

```
!/bin/bash # postfix Postfix Mail Transfer Agent # chkconfig: 2345 80 30 #
description: Postfix is a Mail Transport Agent, which is the program that
moves mail from one machine to another. # processname: master # pidfile:
/var/spool/postfix/pid/master.pid # config: /etc/postfix/main.cf # config:
/etc/postfix/master.cf  BEGIN INIT INFO # Provides: postfix MTA #
Required-Start: $local_fs $network $remote_fs # Required-Stop: $local_fs
$network $remote_fs # Default-Start: 2 3 4 5 # Default-Stop: 0 1 6 # Short-
Description: start and stop postfix # Description: Postfix is a Mail
Transport Agent, which is the program that moves mail from one machine to
another. # END INIT INFO
```

In the above example, only lines starting with **# chkconfig** and **# description** are mandatory, so you might not find the rest in different init files. The text enclosed between the **# BEGIN INIT INFO and # END INIT INFO lines is called Linux Standard Base (LSB) header. If specified, LSB headers contain directives defining the service description, dependencies, and default runlevels. What follows is an overview of analytic tasks aiming to collect the data needed for a new unit file. The postfix init script is used as an example, see the resulting postfix unit file in Example 3.16, "postfix.service unit file".**

### Finding the service description

**Find descriptive information about the script on the line starting with #description. Use this description together with the service name in the `Description` option in the [Unit] section of the unit file. The LSB header might contain similar data on the #Short-Description and #Description lines.**

### Finding service dependencies

**The LSB header might contain several directives that form dependencies between services. Most of them are translatable to systemd unit options, see Table 3.12, "Dependency options from the LSB header"**

**Table 3.12. Dependency options from the LSB header**

| LSB Option | Description | Unit File Equivalent |
|---|---|---|
| `Provides` | Specifies the boot facility name of the service, that can be referenced in other init scripts (with the "$" prefix). This is no longer needed as unit files refer to other units by their file names. | – |
| `Required-Start` | Contains boot facility names of required services. This is translated as an ordering dependency, boot facility names are replaced with unit file names of corresponding services or targets they belong to. For example, in case of `postfix`, the Required-Start dependency on $network was translated to the After dependency on network.target. | `After`, `Before` |
| `Should-Start` | Constitutes weaker dependencies than Required-Start. Failed Should-Start dependencies do not affect the service startup. | `After`, `Before` |
| `Required-Stop`, `Should-Stop` | Constitute negative dependencies. | `Conflicts` |

### Finding default targets of the service

**The line starting with #chkconfig contains three numerical values. The most important is the first number that represents the default runlevels in which the service is started. Use Table 3.6, "Comparison of SysV runlevels with systemd targets" to map these runlevels to equivalent**

systemd targets. Then list these targets in the `WantedBy` option in the [Install] section of the unit file. For example, `postfix` was previously started in runlevels 2, 3, 4, and 5, which translates to multi-user.target and graphical.target. Note that the graphical.target depends on multiuser.target, therefore it is not necessary to specify both, as in Example 3.16, “postfix.service unit file”. You might find information on default and forbidden runlevels also at #Default-Start and #Default-Stop lines in the LSB header.

The other two values specified on the #chkconfig line represent startup and shutdown priorities of the init script. These values are interpreted by systemd if it loads the init script, but there is no unit file equivalent.

#### Finding files used by the service

Init scripts require loading a function library from a dedicated directory and allow importing configuration, environment, and PID files. Environment variables are specified on the line starting with #config in the init script header, which translates to the `EnvironmentFile` unit file option. The PID file specified on the #pidfile init script line is imported to the unit file with the `PIDFile` option.

The key information that is not included in the init script header is the path to the service executable, and potentially some other files required by the service. In previous versions of Red Hat Enterprise Linux, init scripts used a Bash case statement to define the behavior of the service on default actions, such as start, stop, or restart, as well as custom-defined actions. The following excerpt from the `postfix` init script shows the block of code to be executed at service start.

```
conf_check() {
    [ -x /usr/sbin/postfix ] || exit 5
    [ -d /etc/postfix ] || exit 6
    [ -d /var/spool/postfix ] || exit 5
}

make_aliasesdb() {
 if [ "$(/usr/sbin/postconf -h alias_database)" == "hash:/etc/aliases" ]
 then
  # /etc/aliases.db might be used by other MTA, make sure nothing
  # has touched it since our last newaliases call
  [ /etc/aliases -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -ot /etc/aliases.db ] || return
  /usr/bin/newaliases
  touch -r /etc/aliases.db "$ALIASESDB_STAMP"
 else
  /usr/bin/newaliases
 fi
}

start() {
 [ "$EUID" != "0" ] && exit 4
 # Check that networking is up.
 [ ${NETWORKING} = "no" ] && exit 1
 conf_check
 # Start daemons.
 echo -n $"Starting postfix: "
 make_aliasesdb >/dev/null 2>&1
 [ -x $CHROOT_UPDATE ] && $CHROOT_UPDATE
 /usr/sbin/postfix start 2>/dev/null 1>&2 && success || failure $"$prog
```

```
start"
 RETVAL=$?
 [ $RETVAL -eq 0 ] && touch $lockfile
        echo
 return $RETVAL
}
```

The extensibility of the init script allowed specifying two custom functions, `conf_check()` and `make_aliasesdb()`, that are called from the `start()` function block. On closer look, several external files and directories are mentioned in the above code: the main service executable `/usr/sbin/postfix`, the `/etc/postfix/` and `/var/spool/postfix/` configuration directories, as well as the `/usr/sbin/postconf/` directory.

Systemd supports only the predefined actions, but enables executing custom executables with `ExecStart`, `ExecStartPre`, `ExecStartPost`, `ExecStop`, and `ExecReload` options. The `/usr/sbin/postfix` together with supporting scripts are executed on service start. Consult the `postfix` unit file at Example 3.16, “postfix.service unit file”.

Converting complex init scripts requires understanding the purpose of every statement in the script. Some of the statements are specific to the operating system version, therefore you do not need to translate them. On the other hand, some adjustments might be needed in the new environment, both in unit file as well as in the service executable and supporting files.

### 3.5.4. Modifying existing unit files

Services installed on the system come with default unit files that are stored in the `/usr/lib/systemd/system/` directory. System Administrators should not modify these files directly, therefore any customization must be confined to configuration files in the `/etc/systemd/system/` directory. Depending on the extent of the required changes, pick one of the following approaches:

- Create a directory for supplementary configuration files at `/etc/systemd/system/unit.d/`. This method is recommended for most use cases. It enables extending the default configuration with additional functionality, while still referring to the original unit file. Changes to the default unit introduced with a package upgrade are therefore applied automatically. See Extending the default unit configuration for more information.

- Create a copy of the original unit file `/usr/lib/systemd/system/` in `/etc/systemd/system/` and make changes there. The copy overrides the original file, therefore changes introduced with the package update are not applied. This method is useful for making significant unit changes that should persist regardless of package updates. See Overriding the default unit configuration for details.

In order to return to the default configuration of the unit, just delete custom-created configuration files in `/etc/systemd/system/`. To apply changes to unit files without rebooting the system, execute:

```
systemctl daemon-reload
```

The `daemon-reload` option reloads all unit files and recreates the entire dependency tree, which is needed to immediately apply any change to a unit file. As an alternative, you can achieve the same result with the following command, which must be executed under the `root` user:

```
init q
```

Also, if the modified unit file belongs to a running service, this service must be restarted to accept new settings:

```
systemctl restart name.service
```

**IMPORTANT**

To modify properties, such as dependencies or timeouts, of a service that is handled by a SysV initscript, do not modify the initscript itself. Instead, create a `systemd` drop-in configuration file for the service as described in Extending the default unit configuration and Overriding the default unit configuration. Then manage this service in the same way as a normal `systemd` service.

For example, to extend the configuration of the `network` service, do not modify the `/etc/rc.d/init.d/network` initscript file. Instead, create new directory `/etc/systemd/system/network.service.d/` and a `systemd` drop-in file `/etc/systemd/system/network.service.d/my_config.conf`. Then, put the modified values into the drop-in file. Note: `systemd` knows the `network` service as `network.service`, which is why the created directory must be called `network.service.d`

Extending the default unit configuration
To extend the default unit file with additional configuration options, first create a configuration directory in `/etc/systemd/system/`. If extending a service unit, execute the following command as `root`:

```
mkdir /etc/systemd/system/name.service.d/
```

Replace *name* with the name of the service you want to extend. The above syntax applies to all unit types.

Create a configuration file in the directory made in the previous step. Note that the file name must end with the .conf suffix. Type:

```
touch /etc/systemd/system/name.service.d/config_name.conf
```

Replace *config_name* with the name of the configuration file. This file adheres to the normal unit file structure, therefore all directives must be specified under appropriate sections, see Section 3.5.1, “Understanding the unit file structure”.

For example, to add a custom dependency, create a configuration file with the following content:

```
[Unit]
Requires=new_dependency
After=new_dependency
```

Where *new_dependency* stands for the unit to be marked as a dependency. Another example is a configuration file that restarts the service after its main process exited, with a delay of 30 seconds:

```
[Service]
Restart=always
RestartSec=30
```

It is recommended to create small configuration files focused only on one task. Such files can be easily moved or linked to configuration directories of other services.

To apply changes made to the unit, execute as `root`:

```
systemctl daemon-reload
systemctl restart name.service
```

**Example 3.19. Extending the httpd.service configuration**

To modify the httpd.service unit so that a custom shell script is automatically executed when starting the Apache service, perform the following steps. First, create a directory and a custom configuration file:

```
~]# mkdir /etc/systemd/system/httpd.service.d/
```

```
~]# touch /etc/systemd/system/httpd.service.d/custom_script.conf
```

Provided that the script you want to start automatically with Apache is located at `/usr/local/bin/custom.sh`, insert the following text to the `custom_script.conf` file:

```
[Service]
ExecStartPost=/usr/local/bin/custom.sh
```

To apply the unit changes, execute:

```
~]# systemctl daemon-reload
```

```
~]# systemctl restart httpd.service
```

### NOTE

The configuration files from configuration directories in `/etc/systemd/system/` take precedence over unit files in `/usr/lib/systemd/system/`. Therefore, if the configuration files contain an option that can be specified only once, such as `Description` or `ExecStart`, the default value of this option is overridden. Note that in the output of the `systemd-delta` command, described in Monitoring overriden units, such units are always marked as [EXTENDED], even though in sum, certain options are actually overridden.

**Overriding the default unit configuration**
To make changes that will persist after updating the package that provides the unit file, first copy the file to the `/etc/systemd/system/` directory. To do so, execute the following command as `root`:

```
cp /usr/lib/systemd/system/name.service /etc/systemd/system/name.service
```

Where *name* stands for the name of the service unit you wish to modify. The above syntax applies to all unit types.

Open the copied file with a text editor, and make the desired changes. To apply the unit changes, execute as `root`:

```
systemctl daemon-reload
systemctl restart name.service
```

**Example 3.20. Changing the timeout limit**

You can specify a timeout value per service to prevent a malfunctioning service from freezing the system. Otherwise, timeout is set by default to 90 seconds for normal services and to 300 seconds for SysV-compatible services.

For example, to extend timeout limit for the `httpd` service:

1. Copy the `httpd` unit file to the `/etc/systemd/system/` directory:

   ```
   cp /usr/lib/systemd/system/httpd.service
   /etc/systemd/system/httpd.service
   ```

2. Open file `/etc/systemd/system/httpd.service` and specify the `TimeoutStartUSec` value in the `[Service]` section:

   ```
   …
   [Service]
   …
   PrivateTmp=true
   TimeoutStartSec=10

   [Install]
   WantedBy=multi-user.target
   …
   ```

3. Reload the `systemd` daemon:

   ```
   systemctl daemon-reload
   ```

4. Optional. Verify the new timeout value:

   ```
   systemctl show httpd -p TimeoutStartUSec
   ```

**NOTE**

To change the timeout limit globally, input the `DefaultTimeoutStartSec` in the `/etc/systemd/system.conf` file.

**Monitoring overriden units**
To display an overview of overridden or modified unit files, use the following command:

```
systemd-delta
```

For example, the output of the above command can look as follows:

```
[EQUIVALENT] /etc/systemd/system/default.target →
/usr/lib/systemd/system/default.target
[OVERRIDDEN] /etc/systemd/system/autofs.service →
/usr/lib/systemd/system/autofs.service

--- /usr/lib/systemd/system/autofs.service      2014-10-16
21:30:39.000000000 -0400
+ /etc/systemd/system/autofs.service  2014-11-21 10:00:58.513568275 -0500
@@ -8,7 +8,8 @@
 EnvironmentFile=-/etc/sysconfig/autofs
 ExecStart=/usr/sbin/automount $OPTIONS --pid-file /run/autofs.pid
 ExecReload=/usr/bin/kill -HUP $MAINPID
-TimeoutSec=180
+TimeoutSec=240
+Restart=Always

 [Install]
 WantedBy=multi-user.target

[MASKED]     /etc/systemd/system/cups.service →
/usr/lib/systemd/system/cups.service
[EXTENDED]   /usr/lib/systemd/system/sssd.service →
/etc/systemd/system/sssd.service.d/journal.conf

4 overridden configuration files found.
```

Table 3.13, “systemd-delta difference types” lists override types that can appear in the output of `systemd-delta`. Note that if a file is overridden, `systemd-delta` by default displays a summary of changes similar to the output of the `diff` command.

**Table 3.13. systemd-delta difference types**

| Type | Description |
|---|---|
| [MASKED] | Masked unit files, see Section 3.2.7, “Disabling a service” for description of unit masking. |
| [EQUIVALENT] | Unmodified copies that override the original files but do not differ in content, typically symbolic links. |
| [REDIRECTED] | Files that are redirected to another file. |
| [OVERRIDEN] | Overridden and changed files. |
| [EXTENDED] | Files that are extended with .conf files in the **/etc/systemd/system/*unit*.d/** directory. |

| Type | Description |
|---|---|
| [UNCHANGED] | Unmodified files are displayed only when the **`--type=unchanged`** option is used. |

It is good practice to run `systemd-delta` after system update to check if there are any updates to the default units that are currently overridden by custom configuration. It is also possible to limit the output only to a certain difference type. For example, to view just the overridden units, execute:

```
systemd-delta --type=overridden
```

If you want to edit a unit file and automatically create a drop-in file with the submitted changes, use the following command:

```
~]# systemctl edit unit_name.type_extension
```

To dump the unit configuration applying all overrides and drop-ins, use this command:

```
~]# systemctl cat unit_name.type_extension
```

Replace the *unit_name.type_extension* by the name of the required unit and its type, for example `tuned.service`.

### 3.5.5. Working with instantiated units

It is possible to instantiate multiple units from a single template configuration file at runtime. The "@" character is used to mark the template and to associate units with it. Instantiated units can be started from another unit file (using `Requires` or `Wants` options), or with the `systemctl start` command. Instantiated service units are named the following way:

```
template_name@instance_name.service
```

Where *template_name* stands for the name of the template configuration file. Replace *instance_name* with the name for the unit instance. Several instances can point to the same template file with configuration options common for all instances of the unit. Template unit name has the form of:

```
unit_name@.service
```

For example, the following `Wants` setting in a unit file:

```
Wants=getty@ttyA.service getty@ttyB.service
```

first makes systemd search for given service units. If no such units are found, the part between "@" and the type suffix is ignored and systemd searches for the `getty@.service` file, reads the configuration from it, and starts the services.

Wildcard characters, called unit specifiers, can be used in any unit configuration file. Unit specifiers substitute certain unit parameters and are interpreted at runtime. Table 3.14, "Important unit specifiers" lists unit specifiers that are particularly useful for template units.

**Table 3.14. Important unit specifiers**

| Unit Specifier | Meaning | Description |
|---|---|---|
| **%n** | Full unit name | Stands for the full unit name including the type suffix. **%N** has the same meaning but also replaces the forbidden characters with ASCII codes. |
| **%p** | Prefix name | Stands for a unit name with type suffix removed. For instantiated units %p stands for the part of the unit name before the "@" character. |
| **%i** | Instance name | Is the part of the instantiated unit name between the "@" character and the type suffix. **%I** has the same meaning but also replaces the forbidden characters for ASCII codes. |
| **%H** | Host name | Stands for the hostname of the running system at the point in time the unit configuration is loaded. |
| **%t** | Runtime directory | Represents the runtime directory, which is either **/run** for the **root** user, or the value of the XDG_RUNTIME_DIR variable for unprivileged users. |

For a complete list of unit specifiers, see the `systemd.unit(5)` manual page.

For example, the `getty@.service` template contains the following directives:

```
[Unit]
Description=Getty on %I
…
[Service]
ExecStart=-/sbin/agetty --noclear %I $TERM
…
```

When the getty@ttyA.service and getty@ttyB.service are instantiated form the above template, `Description=` is resolved as Getty on ttyA and Getty on ttyB.

## 3.6. ADDITIONAL RESOURCES

For more information on systemd and its usage on Red Hat Enterprise Linux, see the resources listed below.

### 3.6.1. Installed Documentation

- `systemctl`(1) — The manual page for the `systemctl` command line utility provides a complete list of supported options and commands.
- `systemd`(1) — The manual page for the `systemd` system and service manager provides more information about its concepts and documents available command line options and environment variables, supported configuration files and directories, recognized signals, and available kernel options.
- `systemd-delta`(1) — The manual page for the `systemd-delta` utility that allows to find extended and overridden configuration files.
- `systemd.directives(7)` — The manual page named `systemd.directives` provides detailed information about systemd directives.
- `systemd.unit`(5) — The manual page named `systemd.unit` provides detailed information about systemd unit files and documents all available configuration options.
- `systemd.service`(5) — The manual page named `systemd.service` documents the format of service unit files.
- `systemd.target`(5) — The manual page named `systemd.target` documents the format of target unit files.
- `systemd.kill`(5) — The manual page named `systemd.kill` documents the configuration of the process killing procedure.

### 3.6.2. Online Documentation

- systemd Home Page — The project home page provides more information about systemd.

# CHAPTER 4. MANAGING USER AND GROUP ACCOUNTS AND SETTING PERMISSIONS ON FILES

The control of users and groups is a core element of Red Hat Enterprise Linux system administration. This section explains how to add, manage, and delete users and groups in the graphical user interface and on the command line, and covers advanced topics, such as creating group directories.

## 4.1. INTRODUCTION TO USERS AND GROUPS

While users can be either people (meaning accounts tied to physical users) or accounts that exist for specific applications to use, groups are logical expressions of organization, tying users together for a common purpose. Users within a group share the same permissions to read, write, or execute files owned by that group.

Each user is associated with a unique numerical identification number called a *user ID* (UID). Likewise, each group is associated with a *group ID* (GID). A user who creates a file is also the owner and group owner of that file. The file is assigned separate read, write, and execute permissions for the owner, the group, and everyone else. The file owner can be changed only by `root`, and access permissions can be changed by both the `root` user and file owner.

## 4.2. RESERVED USER AND GROUP IDS

Red Hat Enterprise Linux reserves user and group IDs below 1000 for system users and groups. By default, the User Manager does not display the system users. Reserved user and group IDs are documented in the `setup` package. To view the documentation, use this command:

```
cat /usr/share/doc/setup*/uidgid
```

The recommended practice is to assign IDs starting at 5,000 that were not already reserved, as the reserved range can increase in the future. To make the IDs assigned to new users by default start at 5,000, change the `UID_MIN` and `GID_MIN` directives in the `/etc/login.defs` file:

```
[file contents truncated]
UID_MIN                  5000
[file contents truncated]
GID_MIN                  5000
[file contents truncated]
```

**NOTE**

For users created before you changed `UID_MIN` and `GID_MIN` directives, UIDs will still start at the default 1000.

Even with new user and group IDs beginning with 5,000, it is recommended not to raise IDs reserved by the system above 1000 to avoid conflict with systems that retain the 1000 limit.

## 4.3. USER PRIVATE GROUPS

Red Hat Enterprise Linux uses a *user private group* (*UPG*) scheme, which makes UNIX groups easier to manage. A user private group is created whenever a new user is added to the system. It has the same name as the user for which it was created and that user is the only member of the user private group.

User private groups make it safe to set default permissions for a newly created file or directory, allowing both the user and the group of that user to make modifications to the file or directory.

The setting which determines what permissions are applied to a newly created file or directory is called a *umask* and is configured in the `/etc/bashrc` file. Traditionally on UNIX-based systems, the `umask` is set to `022`, which allows only the user who created the file or directory to make modifications. Under this scheme, all other users, including members of the creator's group, are not allowed to make any modifications. However, under the UPG scheme, this "group protection" is not necessary since every user has their own private group.

A list of all groups is stored in the `/etc/group` configuration file.

## 4.4. SHADOW PASSWORDS

In environments with multiple users, it is very important to use *shadow passwords* provided by the `shadow-utils` package to enhance the security of system authentication files. For this reason, the installation program enables shadow passwords by default.

The advantages of shadow passwords over the traditional way of storing passwords on UNIX-based systems are:

- Shadow passwords improve system security by moving encrypted password hashes from the world-readable `/etc/passwd` file to `/etc/shadow`, which is readable only by the `root` user.
- Shadow passwords store information about password aging.
- Shadow passwords allow to enforce some of the security policies set in the `/etc/login.defs` file.

Most utilities provided by the `shadow-utils` package work properly whether or not shadow passwords are enabled. However, since password aging information is stored exclusively in the `/etc/shadow` file, some utilities and commands do not work without first enabling shadow passwords:

- The `chage` utility for setting password aging parameters.
- The `gpasswd` utility for administrating the `/etc/group` file.
- The `usermod` command with the `-e, --expiredate` or `-f, --inactive` option.
- The `useradd` command with the `-e, --expiredate` or `-f, --inactive` option.

## 4.5. MANAGING USERS IN A GRAPHICAL ENVIRONMENT

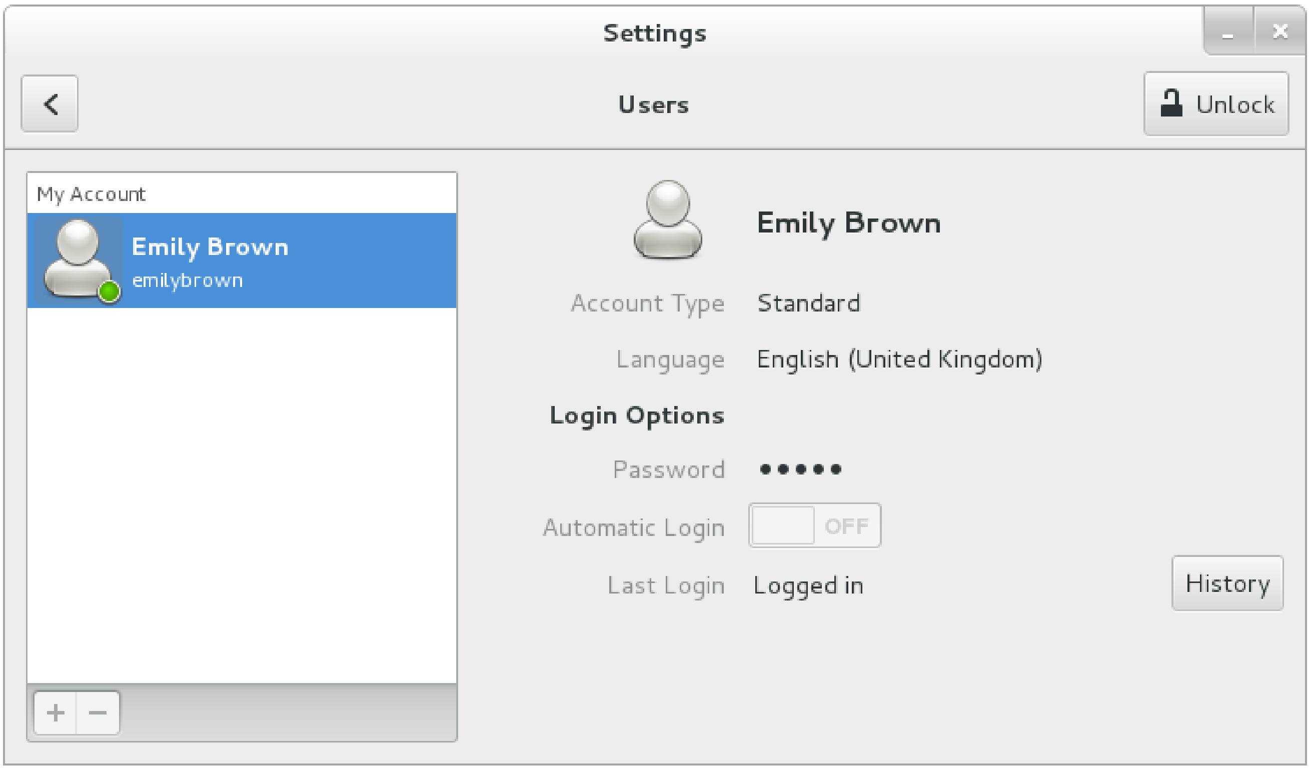
The Users utility allows you to view, modify, add, and delete local users in the graphical user interface.

### 4.5.1. Using the Users Settings tool

Press the `Super` key to enter the Activities Overview, type `Users` and then press `Enter`. The Users settings tool appears. The `Super` key appears in a variety of options, depending on the keyboard and other hardware, but often as either the Windows or Command key, and typically to the left of the Space bar. Alternatively, you can open the Users utility from the Settings menu after clicking your user name in the top right corner of the screen.

To make changes to the user accounts, first select the `Unlock` button and authenticate yourself as indicated by the dialog box that appears. Note that unless you have superuser privileges, the application will prompt you to authenticate as `root`. To add and remove users, select the + and - button respectively. To add a user to the administrative group `wheel`, change the Account Type from `Standard` to `Administrator`. To edit a user's language setting, select the language and a drop-down menu appears.

**Figure 4.1. The Users Settings Tool**




When a new user is created, the account is disabled until a password is set. The Password drop-down menu, shown in Figure 4.2, "The Password Menu", contains the options to set a password by the administrator immediately, choose a password by the user at the first login, or create a guest account with no password required to log in. You can also disable or enable an account from this menu.

**Figure 4.2. The Password Menu**


## 4.6. MANAGING USERS USING COMMAND-LINE TOOLS

**Apart from the Users settings tool described in Section 4.5.1, “Using the Users Settings tool”, which is designed for basic managing of users, you can use command line tools for managing users and groups that are listed in the table below.**

**Table 4.1. Command line utilities for managing users and groups**

| Utilities | Description |
|---|---|
| `id` | Displays user and group IDs. |
| `useradd`, `usermod`, `userdel` | Standard utilities for adding, modifying, and deleting user accounts. |
| `groupadd`, `groupmod`, `groupdel` | Standard utilities for adding, modifying, and deleting groups. |
| `gpasswd` | Utility primarily used for modification of group password in the `/etc/gshadow` file which is used by the `newgrp` command. |
| `pwck`, `grpck` | Utilities that can be used for verification of the password, group, and associated shadow files. |

| Utilities | Description |
| --- | --- |
| **pwconv**, **pwunconv** | Utilities that can be used for the conversion of passwords to shadow passwords, or back from shadow passwords to standard passwords. |
| **grpconv**, **grpunconv** | Similar to the previous, these utilities can be used for conversion of shadowed information for group accounts. |

### 4.6.1. Adding a new user

**To add a new user to the system, type the following at a shell prompt as `root`:**

```
useradd options username
```

**Here *options* are command-line options as described in [Table 4.2, “Common useradd command-line options”](#).**

**By default, the `useradd` command creates a locked user account. To unlock the account, run the following command as `root` to assign a password:**

```
passwd username
```

**Table 4.2. Common useradd command-line options**

| Option | |
| --- | --- |
| **-c** '*comment*' | *comment* can be replaced with any string. This option is generally used to specify the full name of a user. |
| **-d** *home_directory* | Home directory to be used instead of default **/home/*username*/**. |
| **-e** *date* | Date for the account to be disabled in the format YYYY-MM-DD. |
| **-f** *days* | Number of days after the password expires until the account is disabled. If **0** is specified, the account is disabled immediately after the password expires. If **-1** is specified, the account is not disabled after the password expires. |
| **-g** *group_name* | Group name or group number for the user’s default (primary) group. The group must exist prior to being specified here. |

| Option | |
|---|---|
| `-G` *group_list* | List of additional (supplementary, other than default) group names or group numbers, separated by commas, of which the user is a member. The groups must exist prior to being specified here. |
| `-m` | Create the home directory if it does not exist. |
| `-M` | Do not create the home directory. |
| `-N` | Do not create a user private group for the user. |
| `-p` *password* | The password encrypted with `crypt`. |
| `-r` | Create a system account with a UID less than 1000 and without a home directory. |
| `-s` | User's login shell, which defaults to `/bin/bash`. |
| `-u` *uid* | User ID for the user, which must be unique and greater than 999. |

**IMPORTANT**

**The default range of IDs for system and normal users has been changed in Red Hat Enterprise Linux 7 from earlier releases. Before Red Hat Enterprise Linux 7, UID 1-499 was used for system users and values above for normal users. The default range for system users is now 1-999. This change might cause problems when migrating to Red Hat Enterprise Linux 8 with existing users having UIDs and GIDs between 500 and 999. The default ranges of UID and GID can be changed in the `/etc/login.defs` file.**

The following steps illustrate what happens if the command `useradd juan` is issued on a system that has shadow passwords enabled:

1. A new line for `juan` is created in `/etc/passwd`:

   ```
   juan:x:1001:1001::/home/juan:/bin/bash
   ```

   The line has the following characteristics:

   - It begins with the user name `juan`.
   - There is an `x` for the password field indicating that the system is using shadow passwords.
   - A UID greater than 999 is created. Under Red Hat Enterprise Linux 7, UIDs below 1000 are reserved for system use and should not be assigned to users.

- A GID greater than 999 is created. Under Red Hat Enterprise Linux 7, GIDs below 1000 are reserved for system use and should not be assigned to users.
- The optional *GECOS* information is left blank. The GECOS field can be used to provide additional information about the user, such as their full name or phone number.
- The home directory for `juan` is set to `/home/juan/`.
- The default shell is set to `/bin/bash`.

2. A new line for `juan` is created in `/etc/shadow`:

   ```
   juan:!!:14798:0:99999:7:::
   ```

   The line has the following characteristics:

   - It begins with the user name `juan`.
   - Two exclamation marks (`!!`) appear in the password field of the `/etc/shadow` file, which locks the account.

     **NOTE**

     If an encrypted password is passed using the `-p` flag, it is placed in the `/etc/shadow` file on the new line for the user.

   - The password is set to never expire.

3. A new line for a group named `juan` is created in `/etc/group`:

   ```
   juan:x:1001:
   ```

   A group with the same name as a user is called a *user private group*. For more information on user private groups, see Section 4.3, “User private groups”.

   The line created in `/etc/group` has the following characteristics:

   - It begins with the group name `juan`.
   - An `x` appears in the password field indicating that the system is using shadow group passwords.
   - The GID matches the one listed for `juan`'s primary group in `/etc/passwd`.

4. A new line for a group named `juan` is created in `/etc/gshadow`:

   ```
   juan:!::
   ```

   The line has the following characteristics:

   - It begins with the group name `juan`.

- An exclamation mark (`!`) appears in the password field of the `/etc/gshadow` file, which locks the group.
- All other fields are blank.

5. A directory for user `juan` is created in the `/home` directory:

```
~]# ls -ld /home/juan
drwx------. 4 juan juan 4096 Mar  3 18:23 /home/juan
```

This directory is owned by user `juan` and group `juan`. It has *read*, *write*, and *execute* privileges only for the user `juan`. All other permissions are denied.

6. The files within the `/etc/skel/` directory (which contain default user settings) are copied into the new `/home/juan/` directory:

```
~]# ls -la /home/juan
total 28
drwx------. 4 juan juan 4096 Mar  3 18:23 .
drwxr-xr-x. 5 root root 4096 Mar  3 18:23 ..
-rw-r—r--. 1 juan juan   18 Jun 22  2010 .bash_logout
-rw-r—r--. 1 juan juan  176 Jun 22  2010 .bash_profile
-rw-r—r--. 1 juan juan  124 Jun 22  2010 .bashrc
drwxr-xr-x. 4 juan juan 4096 Nov 23 15:09 .mozilla
```

At this point, a locked account called `juan` exists on the system. To activate it, the administrator must next assign a password to the account using the `passwd` command and, optionally, set password aging guidelines.

### 4.6.2. Adding a new group

To add a new group to the system, type the following at a shell prompt as `root`:

```
groupadd options group_name
```

Here `options` are command-line options as described in Table 4.3, "Common groupadd command-line options".

**Table 4.3. Common groupadd command-line options**

| Option | Description |
|---|---|
| `-f`, `--force` | When used with `-g` *gid* and *gid* already exists, **`groupadd`** will choose another unique *gid* for the group. |
| `-g` *gid* | Group ID for the group, which must be unique and greater than 999. |
| `-K`, `--key` *key=value* | Override **`/etc/login.defs`** defaults. |
| `-o`, `--non-unique` | Allows creating groups with duplicate GID. |

| Option | Description |
|---|---|
| **-p**, **--password** *password* | Use this encrypted password for the new group. |
| **-r** | Create a system group with a GID less than 1000. |

### 4.6.3. Adding an existing user to an existing group

Use the `usermod` utility to add an already existing user to an already existing group.

Various options of `usermod` have different impact on user's primary group and on his or her supplementary groups.

To override user's primary group, run the following command as `root`:

```
usermod -g group_name user_name
```

To override user's supplementary groups, run the following command as `root`:

```
usermod -G group_name1,group_name2,… user_name
```

Note that in this case all previous supplementary groups of the user are replaced by the new group or several new groups.

To add one or more groups to user's supplementary groups, run one of the following commands as `root`:

```
usermod -aG group_name1,group_name2,… user_name
```

```
usermod --append -G group_name1,group_name2,… user_name
```

Note that in this case the new group is added to user's current supplementary groups.

### 4.6.4. Creating group directories

System administrators usually like to create a group for each major project and assign people to the group when they need to access that project's files. With this traditional scheme, file management is difficult; when someone creates a file, it is associated with the primary group to which they belong. When a single person works on multiple projects, it becomes difficult to associate the right files with the right group. However, with the UPG scheme, groups are automatically assigned to files created within a directory with the *setgid* bit set. The *setgid* bit makes managing group projects that share a common directory very simple because any files a user creates within the directory are owned by the group that owns the directory.

For example, a group of people need to work on files in the `/opt/myproject/` directory. Some people are trusted to modify the contents of this directory, but not everyone.

1. As `root`, create the `/opt/myproject/` directory by typing the following at a shell prompt:

   ```
   mkdir /opt/myproject
   ```

2. **Add the `myproject` group to the system:**

```
groupadd myproject
```

3. **Associate the contents of the `/opt/myproject/` directory with the `myproject` group:**

```
chown root:myproject /opt/myproject
```

4. **Allow users in the group to create files within the directory and set the *setgid* bit:**

```
chmod 2775 /opt/myproject
```

At this point, all members of the `myproject` group can create and edit files in the `/opt/myproject/` directory without the administrator having to change file permissions every time users write new files. To verify that the permissions have been set correctly, run the following command:

```
ls -ld /opt/myproject
drwxrwsr-x. 3 root myproject 4096 Mar  3 18:31 /opt/myproject
```

5. **Add users to the `myproject` group:**

```
usermod -aG myproject username
```

### 4.6.5. Setting default permissions for new files using umask

When a process creates a file, the file has certain default permissions, for example, `-rw-rw-r--`. These initial permissions are partially defined by the *file mode creation mask*, also called *file permission mask* or *umask*. Every process has its own umask, for example, bash has *umask* `0022` by default. Process *umask* can be changed.

#### 4.6.5.1. What umask consists of

A *umask* consists of bits corresponding to standard file permissions. For example, for *umask* `0137`, the digits mean that:

- `0` = no meaning, it is always `0` (umask does not affect special bits)
- `1` = for owner permissions, the execute bit is set
- `3` = for group permissions, the execute and write bits are set
- `7` = for others permissions, the execute, write, and read bits are set

*Umasks* can be represented in binary, octal, or symbolic notation. For example, the octal representation `0137` equals symbolic representation `u=rw-,g=r--,o=---`. Symbolic notation specification is the reverse of the octal notation specification: it shows the allowed permissions, not the prohibited permissions.

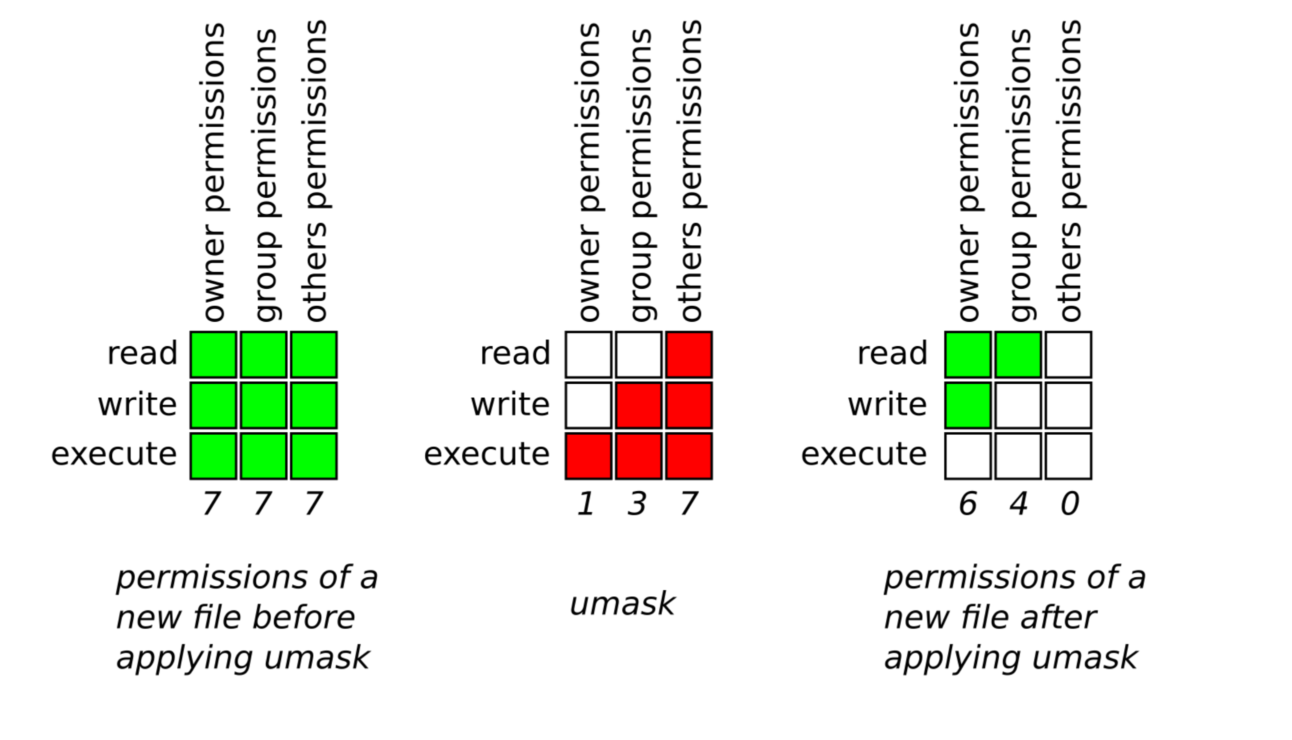
#### 4.6.5.2. How umask works

*Umask* prohibits permissions from being set for a file:

- When a bit is set in *umask*, it is unset in the file.
- When a bit is not set in *umask*, it can be set in the file, depending on other factors.

The following figure shows how *umask* `0137` affects creating a new file.

**Figure 4.3. Applying umask when creating a file**




**IMPORTANT**

For security reasons, a regular file cannot have execute permissions by default. Therefore, even if *umask* is `0000`, which does not prohibit any permissions, a new regular file still does not have execute permissions. However, directories can be created with execute permissions:

```
[john@server tmp]$ umask 0000
[john@server tmp]$ touch file
[john@server tmp]$ mkdir directory
[john@server tmp]$ ls -lh .
total 0
drwxrwxrwx. 2 john john 40 Nov  2 13:17 directory
-rw-rw-rw-. 1 john john  0 Nov  2 13:17 file
```

### 4.6.5.3. Managing umask in Shells

For popular shells, such as `bash`, `ksh`, `zsh` and `tcsh`, *umask* is managed using the `umask` shell `builtin`. Processes started from shell inherit its umask.

#### 4.6.5.3.1. Displaying the current mask

To show the current *umask* in octal notation:

```
~]$ umask
0022
```

To show the current *umask* in symbolic notation:

```
umask -S
u=rwx,g=rx,o=rx
```

#### 4.6.5.3.2. Setting mask in shell using umask

To set *umask* for the current shell session using octal notation run:

```
umask octal_mask
```

Substitute *octal_mask* with four or less digits from `0` to `7`. When three or less digits are provided, permissions are set as if the command contained leading zeros. For example, *umask* `7` translates to `0007`.

> **Example 4.1. Setting umask Using Octal Notation**
>
> To prohibit new files from having write and execute permissions for owner and group, and from having any permissions for others:
>
> ```
> umask 0337
> ```
>
> Or:
>
> ```
> umask 337
> ```

To set *umask* for the current shell session using symbolic notation:

```
umask -S symbolic_mask
```

> **Example 4.2. Setting umask Using Symbolic Notation**
>
> To set *umask* `0337` using symbolic notation:
>
> ```
> umask -S u=r,g=r,o=
> ```

#### 4.6.5.3.3. Working with the default shell umask

Shells usually have a configuration file where their default *umask* is set. For `bash`, it is `/etc/bashrc`. To show the default `bash` umask:

```
grep -i -B 1 umask /etc/bashrc
```

The output shows if *umask* is set, either using the `umask` command or the `UMASK` variable. In the following example, *umask* is set to `022` using the `umask` command:

```
grep -i -B 1 umask /etc/bashrc
    # By default, we want umask to get set. This sets it for non-login
shell. —     if [ $UID -gt 199 ] && [ “id -gn” = “id -un” ]; then
       umask 002
    else
       umask 022
```

To change the default *umask* for bash, change the `umask` command call or the `UMASK` variable assignment in `/etc/bashrc`. This example changes the default *umask* to `0227`:

```
    if [ $UID -gt 199 ] && [ “id -gn” = “id -un” ]; then
       umask 002
    else
       umask 227
```

#### 4.6.5.3.4. Working with the default shell umask of a specific user

By default, bash *umask* of a new user defaults to the one defined in `/etc/bashrc`.

To change bash *umask*for a particular user, add a call to the `umask` command in `$HOME/.bashrc` file of that user. For example, to change bash *umask* of user `john` to `0227`:

```
john@server ~]$ echo 'umask 227' [] /home/john/.bashrc
```

#### 4.6.5.3.5. Setting default permissions for newly created home directories

To change permissions with which user home directories are created, change the `UMASK` variable in the `/etc/login.defs` file:

```
# The permission mask is initialized to this value. If not specified,
# the permission mask will be initialized to 022.
UMASK 077
```

## 4.7. ADDITIONAL RESOURCES

For more information on how to manage users and groups on Red Hat Enterprise Linux, see the resources listed below.

### 4.7.1. Installed Documentation

For information about various utilities for managing users and groups, see the following manual pages:

- `useradd`(8) — The manual page for the `useradd` command documents how to use it to create new users.
- `userdel`(8) — The manual page for the `userdel` command documents how to use it to delete users.
- `usermod`(8) — The manual page for the `usermod` command documents how to use it to modify users.

- `groupadd`(8) — The manual page for the `groupadd` command documents how to use it to create new groups.
- `groupdel`(8) — The manual page for the `groupdel` command documents how to use it to delete groups.
- `groupmod`(8) — The manual page for the `groupmod` command documents how to use it to modify group membership.
- `gpasswd`(1) — The manual page for the `gpasswd` command documents how to manage the `/etc/group` file.
- `grpck`(8) — The manual page for the `grpck` command documents how to use it to verify the integrity of the `/etc/group` file.
- `pwck`(8) — The manual page for the `pwck` command documents how to use it to verify the integrity of the `/etc/passwd` and `/etc/shadow` files.
- `pwconv`(8) — The manual page for the `pwconv`, `pwunconv`, `grpconv`, and `grpunconv` commands documents how to convert shadowed information for passwords and groups.
- `id`(1) — The manual page for the `id` command documents how to display user and group IDs.

For information about related configuration files, see:

- `group`(5) — The manual page for the `/etc/group` file documents how to use this file to define system groups.
- `passwd`(5) — The manual page for the `/etc/passwd` file documents how to use this file to define user information.
- `shadow`(5) — The manual page for the `/etc/shadow` file documents how to use this file to set passwords and account expiration information for the system.

# CHAPTER 5. USING THE CHRONY SUITE TO CONFIGURE NTP

## 5.1. INTRODUCTION TO CONFIGURING NTP WITH CHRONY

Accurate timekeeping is important for a number of reasons in IT. In networking for example, accurate time stamps in packets and logs are required. In Linux systems, the `NTP` protocol is implemented by a daemon running in user space.

The user space daemon updates the system clock running in the kernel. The system clock can keep time by using various clock sources. Usually, the *Time Stamp Counter* (TSC) is used. The TSC is a CPU register which counts the number of cycles since it was last reset. It is very fast, has a high resolution, and there are no interruptions.

Red Hat Enterprise Linux 8, the `NTP` protocol is implemented by the `chronyd` daemon, available from the repositories in the `chrony` package.

These sections describe the use of the chrony suite.

## 5.2. INTRODUCTION TO CHRONY SUITE

chrony is an implementation of the `Network Time Protocol (NTP)`. You can use chrony:

- To synchronize the system clock with `NTP` servers
- To synchronize the system clock with a reference clock, for example a GPS receiver
- To synchronize the system clock with a manual time input
- As an `NTPv4(RFC 5905)` server or peer to provide a time service to other computers in the network

chrony performs well in a wide range of conditions, including intermittent network connections, heavily congested networks, changing temperatures (ordinary computer clocks are sensitive to temperature), and systems that do not run continuously, or run on a virtual machine.

Typical accuracy between two machines synchronized over the Internet is within a few milliseconds, and for machines on a LAN within tens of microseconds. Hardware timestamping or a hardware reference clock may improve accuracy between two machines synchronized to a sub-microsecond level.

chrony consists of `chronyd`, a daemon that runs in user space, and chronyc, a command line program which can be used to monitor the performance of `chronyd` and to change various operating parameters when it is running.

The chrony daemon, `chronyd`, can be monitored and controlled by the command line utility chronyc. This utility provides a command prompt which allows entering a number of commands to query the current state of `chronyd` and make changes to its configuration. By default, `chronyd` accepts only commands from a local instance of chronyc, but it can be configured to accept monitoring commands also from remote hosts. The remote access should be restricted.

### 5.2.1. Using chronyc to control chronyd

To make changes to the local instance of `chronyd` using the command line utility chronyc in interactive mode, enter the following command as `root`:

```
~]# chronyc
```

chronyc must run as `root` if some of the restricted commands are to be used.

The chronyc command prompt will be displayed as follows:

```
chronyc>
```

You can type `help` to list all of the commands.

The utility can also be invoked in non-interactive command mode if called together with a command as follows:

```
chronyc command
```

**NOTE**

Changes made using chronyc are not permanent, they will be lost after a `chronyd` restart. For permanent changes, modify `/etc/chrony.conf`.

## 5.3. DIFFERENCES BETWEEN CHRONY AND NTP

`Network Time Protocol (NTP)` has two different implementations with similar basic functionality - ntp and chrony.

Both ntp and chrony can operate as an `NTP` client in order to synchronize the system clock with `NTP` servers and they can operate as an `NTP` server for other computers in the network. Each implementation has some unique features. For comparison of ntp and chrony, see Comparison of NTP implementations.

Configuration specific to an `NTP` client is identical in most cases. `NTP` servers are specified with the `server` directive. A pool of servers can be specified with the `pool` directive.

Configuration specific to an `NTP` server differs in how the client access is controlled. By default, `ntpd` responds to client requests from any address. The access can be restricted with the `restrict` directive, but it is not possible to disable the access completely if `ntpd` uses any servers as a client. `chronyd` allows no access by default and operates as an `NTP` client only. To make chrony operate as an `NTP` server, you need to specify some addresses within the `allow` directive.

`ntpd` and `chronyd` differ also in the default behavior with respect to corrections of the system clock. `ntpd` corrects the clock by step when the offset is larger than 128 milliseconds. If the offset is larger than 1000 seconds, `ntpd` exits unless it is the first correction of the clock and `ntpd` is started with the `-g` option. `chronyd` does not step the clock by default, but the default `chrony.conf` file provided in the `chrony` package allows steps in the first three updates of the clock. After that, all corrections are made slowly by speeding up or slowing down the clock. The `chronyc makestep` command can be issued to force `chronyd` to step the clock at any time.

## 5.4. MIGRATING TO CHRONY

In Red Hat Enterprise Linux 7, users could choose between ntp and chrony to ensure accurate timekeeping. For differences between ntp and chrony, `ntpd` and `chronyd`, see Differences between ntpd and chronyd.

In Red Hat Enterprise Linux 8, ntp is no longer supported. chrony is enabled by default. For this reason, you might need to migrate from ntp to chrony.

Migrating from ntp to chrony is straightforward in most cases. The corresponding names of the programs, configuration files and services are:

**Table 5.1. Corresponding names of the programs, configuration files and services when migrating from ntp to chrony**

| ntp name | chrony name |
|---|---|
| /etc/ntp.conf | /etc/chrony.conf |
| /etc/ntp/keys | /etc/chrony.keys |
| ntpd | chronyd |
| ntpq | chronyc |
| ntpd.service | chronyd.service |
| ntp-wait.service | chrony-wait.service |

The ntpdate and sntp utilities, which are included in the `ntp` distribution, can be replaced with `chronyd` using the `-q` option or the `-t` option. The configuration can be specified on the command line to avoid reading `/etc/chrony.conf`. For example, instead of running `ntpdate ntp.example.com`, `chronyd` could be started as:

```
# chronyd -q 'server ntp.example.com iburst'
2018-05-18T12:37:43Z chronyd version 3.3 starting (+CMDMON +NTP +REFCLOCK
+RTC +PRIVDROP +SCFILTER +SIGND +ASYNCDNS +SECHASH +IPV6 +DEBUG)
2018-05-18T12:37:43Z Initial frequency -2.630 ppm
2018-05-18T12:37:48Z System clock wrong by 0.003159 seconds (step)
2018-05-18T12:37:48Z chronyd exiting
```

The ntpstat utility, which was previously included in the `ntp` package and supported only `ntpd`, now supports both `ntpd` and `chronyd`. It is available in the `ntpstat` package.

### 5.4.1. Migration script

A Python script called `ntp2chrony.py` is included in the documentation of the `chrony` package (`/usr/share/doc/chrony`). The script automatically converts an existing `ntp` configuration to `chrony`. It supports the most common directives and options in the `ntp.conf` file. Any lines that are ignored in the conversion are included as comments in the generated `chrony.conf` file for review. Keys that are specified in the `ntp` key file, but are not marked as trusted keys in `ntp.conf` are included in the generated `chrony.keys` file as comments.

By default, the script does not overwrite any files. If /etc/chrony.conf or /etc/chrony.keys already exist, the -b option can be used to rename the file as a backup. The script supports other options. The --help option prints all supported options.

An example of an invocation of the script with the default ntp.conf provided in the ntp package is:

```
# python3 /usr/share/doc/chrony/ntp2chrony.py -b -v
Reading /etc/ntp.conf
Reading /etc/ntp/crypto/pw
Reading /etc/ntp/keys
Writing /etc/chrony.conf
Writing /etc/chrony.keys
```

The only directive ignored in this case is disable monitor, which has a chrony equivalent in the noclientlog directive, but it was included in the default ntp.conf only to mitigate an amplification attack.

The generated chrony.conf file typically includes a number of allow directives corresponding to the restrict lines in ntp.conf. If you do not want to run chronyd as an NTP server, remove all allow directives from chrony.conf.

## 5.5. CONFIGURING CHRONY

The default configuration file for chronyd is /etc/chrony.conf. The -f option can be used to specify an alternate configuration file path. See the chrony.conf(5) man page for further options. For a complete list of the directives that can be used see The chronyd configuration file.

Below is a selection of chronyd configuration options:

Comments

Comments should be preceded by #, %, ; or !

allow

Optionally specify a host, subnet, or network from which to allow NTP connections to a machine acting as NTP server. The default is not to allow connections.
Examples:

```
allow 192.0.2.0/24
```

Use this command to grant access to a specific network.

```
allow 2001:0db8:85a3::8a2e:0370:7334
```

Use this this command to grant access to an IPv6.

The UDP port number 123 needs to be open in the firewall in order to allow the client access:

```
~]#  firewall-cmd --zone=public --add-port=123/udp
```

If you want to open port 123 permanently, use the --permanent option:

```
~]#  firewall-cmd --permanent --zone=public --add-port=123/udp
```

**cmdallow**

This is similar to the `allow` directive (see section `allow`), except that it allows control access (rather than `NTP` client access) to a particular subnet or host. (By "control access" is meant that chronyc can be run on those hosts and successfully connect to `chronyd` on this computer.) The syntax is identical. There is also a `cmddeny all` directive with similar behavior to the `cmdallow all` directive.

**dumpdir**

Path to the directory to save the measurement history across restarts of `chronyd` (assuming no changes are made to the system clock behavior whilst it is not running). If this capability is to be used (via the `dumponexit` command in the configuration file, or the `dump` command in chronyc), the `dumpdir` command should be used to define the directory where the measurement histories are saved.

**dumponexit**

If this command is present, it indicates that `chronyd` should save the measurement history for each of its time sources recorded whenever the program exits. (See the `dumpdir` command above).

**hwtimestamp**

The `hwtimestamp` directive enables hardware timestamping for extremely accurate synchronization. For more details, see the `chrony.conf(5)` manual page.

**local**

The `local` keyword is used to allow `chronyd` to appear synchronized to real time from the viewpoint of clients polling it, even if it has no current synchronization source. This option is normally used on the "master" computer in an isolated network, where several computers are required to synchronize to one another, and the "master" is kept in line with real time by manual input.
An example of the command is:

```
local stratum 10
```

A large value of 10 indicates that the clock is so many hops away from a reference clock that its time is unreliable. If the computer ever has access to another computer which is ultimately synchronized to a reference clock, it will almost certainly be at a stratum less than 10. Therefore, the choice of a high value like 10 for the `local` command prevents the machine's own time from ever being confused with real time, were it ever to leak out to clients that have visibility of real servers.

**log**

The `log` command indicates that certain information is to be logged. It accepts the following options:

**measurements**

This option logs the raw `NTP` measurements and related information to a file called `measurements.log`.

**statistics**

This option logs information about the regression processing to a file called `statistics.log`.

**tracking**

This option logs changes to the estimate of the system's gain or loss rate, and any slews made, to a file called `tracking.log`.

rtc

This option logs information about the system's real-time clock.

refclocks

This option logs the raw and filtered reference clock measurements to a file called `refclocks.log`.

tempcomp

This option logs the temperature measurements and system rate compensations to a file called `tempcomp.log`.
The log files are written to the directory specified by the `logdir` command.

An example of the command is:

```
log measurements statistics tracking
```

logdir

This directive allows the directory where log files are written to be specified.
An example of the use of this directive is:

```
logdir /var/log/chrony
```

makestep

Normally `chronyd` will cause the system to gradually correct any time offset, by slowing down or speeding up the clock as required. In certain situations, the system clock may be so far adrift that this slewing process would take a very long time to correct the system clock. This directive forces `chronyd` to step system clock if the adjustment is larger than a threshold value, but only if there were no more clock updates since `chronyd` was started than a specified limit (a negative value can be used to disable the limit). This is particularly useful when using reference clock, because the `initstepslew` directive only works with `NTP` sources.
An example of the use of this directive is:

```
makestep 1000 10
```

This would step the system clock if the adjustment is larger than 1000 seconds, but only in the first ten clock updates.

maxchange

This directive sets the maximum allowed offset corrected on a clock update. The check is performed only after the specified number of updates to allow a large initial adjustment of the system clock. When an offset larger than the specified maximum occurs, it will be ignored for the specified number of times and then `chronyd` will give up and exit (a negative value can be used to never exit). In both cases a message is sent to syslog.
An example of the use of this directive is:

```
maxchange 1000 1 2
```

After the first clock update, `chronyd` will check the offset on every clock update, it will ignore two adjustments larger than 1000 seconds and exit on another one.

maxupdateskew

One of `chronyd`'s tasks is to work out how fast or slow the computer's clock runs relative to its reference sources. In addition, it computes an estimate of the error bounds around the estimated value.
If the range of error is too large, it indicates that the measurements have not settled down yet, and that the estimated gain or loss rate is not very reliable.

The `maxupdateskew` parameter is the threshold for determining whether an estimate is too unreliable to be used. By default, the threshold is 1000 ppm.

The format of the syntax is:

```
maxupdateskew skew-in-ppm
```

Typical values for *skew-in-ppm* might be 100 for a dial-up connection to servers over a telephone line, and 5 or 10 for a computer on a LAN.

It should be noted that this is not the only means of protection against using unreliable estimates. At all times, `chronyd` keeps track of both the estimated gain or loss rate, and the error bound on the estimate. When a new estimate is generated following another measurement from one of the sources, a weighted combination algorithm is used to update the master estimate. So if `chronyd` has an existing highly-reliable master estimate and a new estimate is generated which has large error bounds, the existing master estimate will dominate in the new master estimate.

minsources

The `minsources` directive sets the minimum number of sources that need to be considered as selectable in the source selection algorithm before the local clock is updated.
The format of the syntax is:

```
minsources number-of-sources
```

By default, *number-of-sources* is 1. Setting minsources to a larger number can be used to improve the reliability, because multiple sources will need to correspond with each other.

noclientlog

This directive, which takes no arguments, specifies that client accesses are not to be logged. Normally they are logged, allowing statistics to be reported using the clients command in chronyc and enabling the clients to use interleaved mode with the `xleave` option in the `server` directive.

reselectdist

When `chronyd` selects synchronization source from available sources, it will prefer the one with minimum synchronization distance. However, to avoid frequent reselecting when there are sources with similar distance, a fixed distance is added to the distance for sources that are currently not selected. This can be set with the `reselectdist` option. By default, the distance is 100 microseconds.
The format of the syntax is:

```
reselectdist dist-in-seconds
```

-

stratumweight

The `stratumweight` directive sets how much distance should be added per stratum to the synchronization distance when `chronyd` selects the synchronization source from available sources.
The format of the syntax is:

```
stratumweight dist-in-seconds
```

By default, *dist-in-seconds* is 1 millisecond. This means that sources with lower stratum are usually preferred to sources with higher stratum even when their distance is significantly worse. Setting `stratumweight` to 0 makes `chronyd` ignore stratum when selecting the source.

rtcfile

The `rtcfile` directive defines the name of the file in which `chronyd` can save parameters associated with tracking the accuracy of the system's real-time clock (RTC).
The format of the syntax is:

```
rtcfile /var/lib/chrony/rtc
```

`chronyd` saves information in this file when it exits and when the `writertc` command is issued in chronyc. The information saved is the RTC's error at some epoch, that epoch (in seconds since January 1 1970), and the rate at which the RTC gains or loses time. Not all real-time clocks are supported as their code is system-specific. Note that if this directive is used then the real-time clock should not be manually adjusted as this would interfere with chrony's need to measure the rate at which the real-time clock drifts if it was adjusted at random intervals.

rtcsync

The `rtcsync` directive is present in the `/etc/chrony.conf` file by default. This will inform the kernel the system clock is kept synchronized and the kernel will update the real-time clock every 11 minutes.

### 5.5.1. Configuring chrony for security

chronyc can access `chronyd` in two ways:

- Internet Protocol, IPv4 or IPv6.
- Unix domain socket, which is accessible locally by the `root` or `chrony` user.

By default, chronyc connects to the Unix domain socket. The default path is `/var/run/chrony/chronyd.sock`. If this connection fails, which can happen for example when chronyc is running under a non-root user, chronyc tries to connect to 127.0.0.1 and then ::1.

Only the following monitoring commands, which do not affect the behavior of `chronyd`, are allowed from the network:

- activity
- manual list

- rtcdata
- smoothing
- sources
- sourcestats
- tracking
- waitsync

The set of hosts from which `chronyd` accepts these commands can be configured with the `cmdallow` directive in the configuration file of `chronyd`, or the `cmdallow` command in chronyc. By default, the commands are accepted only from localhost (127.0.0.1 or ::1).

All other commands are allowed only through the Unix domain socket. When sent over the network, `chronyd` responds with a `Not authorised` error, even if it is from localhost.

**Accessing chronyd remotely with chronyc**

1. Allow access from both IPv4 and IPv6 addresses by adding the following to the `/etc/chrony.conf` file:

   ```
   bindcmdaddress 0.0.0.0
   ```

   or

   ```
   bindcmdaddress :
   ```

2. Allow commands from the remote IP address, network, or subnet by using the `cmdallow` directive.
   Add the following content to the `/etc/chrony.conf` file:

   ```
   cmdallow 192.168.1.0/24
   ```

3. Open port 323 in the firewall to connect from a remote system.

   ```
   ~]#  firewall-cmd --zone=public --add-port=323/udp
   ```

   If you want to open port 323 permanently, use the `--permanent`.

   ```
   ~]#  firewall-cmd --permanent --zone=public --add-port=323/udp
   ```

Note that the `allow` directive is for `NTP` access whereas the `cmdallow` directive is to enable receiving of remote commands. It is possible to make these changes temporarily using chronyc running locally. Edit the configuration file to make permanent changes.

## 5.6. USING CHRONY

### 5.6.1. Installing chrony

The chrony suite is installed by default on Red Hat Enterprise Linux. To ensure that it is, run the following command as `root`:

```
~]# yum install chrony
```

The default location for the chrony daemon is `/usr/sbin/chronyd`. The command line utility will be installed to `/usr/bin/chronyc`.

### 5.6.2. Checking the status of chronyd

To check the status of `chronyd`, issue the following command:

```
~]$ systemctl status chronyd
chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled)
   Active: active (running) since Wed 2013-06-12 22:23:16 CEST; 11h ago
```

### 5.6.3. Starting chronyd

To start `chronyd`, issue the following command as `root`:

```
~]# systemctl start chronyd
```

To ensure `chronyd` starts automatically at system start, issue the following command as `root`:

```
~]# systemctl enable chronyd
```

### 5.6.4. Stopping chronyd

To stop `chronyd`, issue the following command as `root`:

```
~]# systemctl stop chronyd
```

To prevent `chronyd` from starting automatically at system start, issue the following command as `root`:

```
~]# systemctl disable chronyd
```

### 5.6.5. Checking if chrony is synchronized

To check if chrony is synchronized, make use of the `tracking`, `sources`, and `sourcestats` commands.

#### 5.6.5.1. Checking chrony tracking

To check chrony tracking, issue the following command:

```
~]$ chronyc tracking
Reference ID    : CB00710F (foo.example.net)
Stratum         : 3
```

```
Ref time (UTC)  : Fri Jan 27 09:49:17 2017
System time     :  0.000006523 seconds slow of NTP time
Last offset     : -0.000006747 seconds
RMS offset      : 0.000035822 seconds
Frequency       : 3.225 ppm slow
Residual freq   : 0.000 ppm
Skew            : 0.129 ppm
Root delay      : 0.013639022 seconds
Root dispersion : 0.001100737 seconds
Update interval : 64.2 seconds
Leap status     : Normal
```

The fields are as follows:

Reference ID

This is the reference ID and name (or `IP` address) if available, of the server to which the computer is currently synchronized. Reference ID is a hexadecimal number to avoid confusion with IPv4 addresses.

Stratum

The stratum indicates how many hops away from a computer with an attached reference clock we are. Such a computer is a stratum-1 computer, so the computer in the example is two hops away (that is to say, a.b.c is a stratum-2 and is synchronized from a stratum-1).

Ref time

This is the time (UTC) at which the last measurement from the reference source was processed.

System time

In normal operation, `chronyd` never steps the system clock, because any jump in the timescale can have adverse consequences for certain application programs. Instead, any error in the system clock is corrected by slightly speeding up or slowing down the system clock until the error has been removed, and then returning to the system clock’s normal speed. A consequence of this is that there will be a period when the system clock (as read by other programs using the `gettimeofday()` system call, or by the date command in the shell) will be different from `chronyd`'s estimate of the current true time (which it reports to `NTP` clients when it is operating in server mode). The value reported on this line is the difference due to this effect.

Last offset

This is the estimated local offset on the last clock update.

RMS offset

This is a long-term average of the offset value.

Frequency

The "frequency" is the rate by which the system’s clock would be wrong if `chronyd` was not correcting it. It is expressed in ppm (parts per million). For example, a value of 1 ppm would mean that when the system’s clock thinks it has advanced 1 second, it has actually advanced by 1.000001 seconds relative to true time.

Residual freq

This shows the "residual frequency" for the currently selected reference source. This reflects any difference between what the measurements from the reference source indicate the frequency should be and the frequency currently being used.
The reason this is not always zero is that a smoothing procedure is applied to the frequency. Each time a measurement from the reference source is obtained and a new residual frequency

computed, the estimated accuracy of this residual is compared with the estimated accuracy (see `skew`) of the existing frequency value. A weighted average is computed for the new frequency, with weights depending on these accuracies. If the measurements from the reference source follow a consistent trend, the residual will be driven to zero over time.

Skew

This is the estimated error bound on the frequency.

Root delay

This is the total of the network path delays to the stratum-1 computer from which the computer is ultimately synchronized. Root delay values are printed in nanosecond resolution. In certain extreme situations, this value can be negative. (This can arise in a symmetric peer arrangement where the computers' frequencies are not tracking each other and the network delay is very short relative to the turn-around time at each computer.)

Root dispersion

This is the total dispersion accumulated through all the computers back to the stratum-1 computer from which the computer is ultimately synchronized. Dispersion is due to system clock resolution, statistical measurement variations etc. Root dispersion values are printed in nanosecond resolution.

Leap status

This is the leap status, which can be Normal, Insert second, Delete second or Not synchronized.

#### 5.6.5.2. Checking chrony sources

The sources command displays information about the current time sources that `chronyd` is accessing.

The optional argument -v can be specified, meaning verbose. In this case, extra caption lines are shown as a reminder of the meanings of the columns.

```
~]$ chronyc sources
 210 Number of sources = 3
MS Name/IP address         Stratum Poll Reach LastRx Last sample
=====================================================================
=====
#* GPS0                          0   4   377    11   -479ns[ -621ns] /-
134ns
^? a.b.c                         2   6   377    23   -923us[ -924us] +/-
43ms
^ d.e.f                         1   6   377    21  -2629us[-2619us] +/-
86ms
```

The columns are as follows:

M

This indicates the mode of the source. ^ means a server, = means a peer and # indicates a locally connected reference clock.

S

This column indicates the state of the sources. "*" indicates the source to which `chronyd` is currently synchronized. "+" indicates acceptable sources which are combined with the selected source. "-" indicates acceptable sources which are excluded by the combining algorithm. "?" indicates sources to which connectivity has been lost or whose packets do not pass all tests. "x" indicates a clock which `chronyd` thinks is a *falseticker* (its time is inconsistent with a majority of other sources). "~" indicates a source whose time appears to have too much variability. The "?" condition is also shown at start-up, until at least 3 samples have been gathered from it.

Name/IP address

This shows the name or the `IP` address of the source, or reference ID for reference clock.

Stratum

This shows the stratum of the source, as reported in its most recently received sample. Stratum 1 indicates a computer with a locally attached reference clock. A computer that is synchronized to a stratum 1 computer is at stratum 2. A computer that is synchronized to a stratum 2 computer is at stratum 3, and so on.

Poll

This shows the rate at which the source is being polled, as a base-2 logarithm of the interval in seconds. Thus, a value of 6 would indicate that a measurement is being made every 64 seconds.
`chronyd` automatically varies the polling rate in response to prevailing conditions.

Reach

This shows the source's reach register printed as an octal number. The register has 8 bits and is updated on every received or missed packet from the source. A value of 377 indicates that a valid reply was received for all of the last eight transmissions.

LastRx

This column shows how long ago the last sample was received from the source. This is normally in seconds. The letters `m`, `h`, `d` or `y` indicate minutes, hours, days or years. A value of 10 years indicates there were no samples received from this source yet.

Last sample

This column shows the offset between the local clock and the source at the last measurement. The number in the square brackets shows the actual measured offset. This may be suffixed by `ns` (indicating nanoseconds), `us` (indicating microseconds), `ms` (indicating milliseconds), or `s` (indicating seconds). The number to the left of the square brackets shows the original measurement, adjusted to allow for any slews applied to the local clock since. The number following the `+/-` indicator shows the margin of error in the measurement. Positive offsets indicate that the local clock is ahead of the source.

### 5.6.5.3. Checking chrony source statistics

The `sourcestats` command displays information about the drift rate and offset estimation process for each of the sources currently being examined by `chronyd`.

The optional argument `-v` can be specified, meaning verbose. In this case, extra caption lines are shown as a reminder of the meanings of the columns.

```
~]$ chronyc sourcestats
210 Number of sources = 1
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset
Std Dev
==========================================================================
=====
abc.def.ghi                11   5   46m     -0.001      0.045      1us
25us
```

-

The columns are as follows:

Name/IP address
: This is the name or `IP` address of the `NTP` server (or peer) or reference ID of the reference clock to which the rest of the line relates.

NP
: This is the number of sample points currently being retained for the server. The drift rate and current offset are estimated by performing a linear regression through these points.

NR
: This is the number of runs of residuals having the same sign following the last regression. If this number starts to become too small relative to the number of samples, it indicates that a straight line is no longer a good fit to the data. If the number of runs is too low, `chronyd` discards older samples and re-runs the regression until the number of runs becomes acceptable.

Span
: This is the interval between the oldest and newest samples. If no unit is shown the value is in seconds. In the example, the interval is 46 minutes.

Frequency
: This is the estimated residual frequency for the server, in parts per million. In this case, the computer's clock is estimated to be running 1 part in $10^9$ slow relative to the server.

Freq Skew
: This is the estimated error bounds on Freq (again in parts per million).

Offset
: This is the estimated offset of the source.

Std Dev
: This is the estimated sample standard deviation.

### 5.6.6. Manually Adjusting the System Clock

To step the system clock immediately, bypassing any adjustments in progress by slewing, issue the following command as `root`:

```
~]# chronyc makestep
```

If the `rtcfile` directive is used, the real-time clock should not be manually adjusted. Random adjustments would interfere with chrony's need to measure the rate at which the real-time clock drifts.

## 5.7. SETTING UP CHRONY FOR DIFFERENT ENVIRONMENTS

### 5.7.1. Setting up chrony for a system in an isolated network

For a network that is never connected to the Internet, one computer is selected to be the master timeserver. The other computers are either direct clients of the master, or clients of clients. On the master, the drift file must be manually set with the average rate of drift of the system clock. If the master is rebooted, it will obtain the time from surrounding systems and calculate an average to set its system clock. Thereafter it resumes applying adjustments based on the drift file. The drift file will be updated automatically when the `settime` command is used.

On the system selected to be the master, using a text editor running as `root`, edit `/etc/chrony.conf` as follows:

```
driftfile /var/lib/chrony/drift
commandkey 1
keyfile /etc/chrony.keys
initstepslew 10 client1 client3 client6
local stratum 8
manual
allow 192.0.2.0
```

Where `192.0.2.0` is the network or subnet address from which the clients are allowed to connect.

On the systems selected to be direct clients of the master, using a text editor running as `root`, edit the `/etc/chrony.conf` as follows:

```
server master
driftfile /var/lib/chrony/drift
logdir /var/log/chrony
log measurements statistics tracking
keyfile /etc/chrony.keys
commandkey 24
local stratum 10
initstepslew 20 master
allow 192.0.2.123
```

Where `192.0.2.123` is the address of the master, and`master` is the host name of the master. Clients with this configuration will resynchronize the master if it restarts.

On the client systems which are not to be direct clients of the master, the `/etc/chrony.conf` file should be the same except that the `local` and `allow` directives should be omitted.

In an isolated network, you can also use the `local` directive that enables a local reference mode, which allows `chronyd` operating as an `NTP` server to appear synchronized to real time, even when it was never synchronized or the last update of the clock happened a long time ago.

To allow multiple servers in the network to use the same local configuration and to be synchronized to one another, without confusing clients that poll more than one server, use the `orphan` option of the `local` directive which enables the orphan mode. Each server needs to be configured to poll all other servers with `local`. This ensures that only the server with the smallest reference ID has the local reference active and other servers are synchronized to it. When the server fails, another one will take over.

## 5.8. CHRONY WITH HW TIMESTAMPING

### 5.8.1. Understanding hardware timestamping

Hardware timestamping is a feature supported in some Network Interface Controller (NICs) which provides accurate timestamping of incoming and outgoing packets. `NTP` timestamps are usually created by the kernel and chronyd with the use of the system clock. However, when HW timestamping is enabled, the NIC uses its own clock to generate the timestamps when packets are entering or leaving the link layer or the physical layer. When used with `NTP`, hardware timestamping can significantly improve the accuracy of synchronization. For best accuracy, both `NTP` servers and `NTP` clients need to use hardware timestamping. Under ideal conditions, a sub-microsecond accuracy may be possible.

Another protocol for time synchronization that uses hardware timestamping is `PTP`.

Unlike `NTP`, `PTP` relies on assistance in network switches and routers. If you want to reach the best accuracy of synchronization, use `PTP` on networks that have switches and routers with `PTP` support, and prefer `NTP` on networks that do not have such switches and routers.

### 5.8.2. Verifying support for hardware timestamping

To verify that hardware timestamping with `NTP` is supported by an interface, use the `ethtool -T` command. An interface can be used for hardware timestamping with `NTP` if `ethtool` lists the `SOF_TIMESTAMPING_TX_HARDWARE` and `SOF_TIMESTAMPING_TX_SOFTWARE` capabilities and also the `HWTSTAMP_FILTER_ALL` filter mode.

**Example 5.1. Verifying support for hardware timestamping on a specific interface**

```
~]# ethtool -T eth0
```

Output:

```
Timestamping parameters for eth0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
        ptpv1-l4-sync         (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
        ptpv1-l4-delay-req    (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
        ptpv2-l4-sync         (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
        ptpv2-l4-delay-req    (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
        ptpv2-l2-sync         (HWTSTAMP_FILTER_PTP_V2_L2_SYNC)
        ptpv2-l2-delay-req    (HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ)
        ptpv2-event           (HWTSTAMP_FILTER_PTP_V2_EVENT)
        ptpv2-sync            (HWTSTAMP_FILTER_PTP_V2_SYNC)
        ptpv2-delay-req       (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

### 5.8.3. Enabling hardware timestamping

To enable hardware timestamping, use the `hwtimestamp` directive in the `/etc/chrony.conf` file. The directive can either specify a single interface, or a wildcard character can be used to enable hardware timestamping on all interfaces that support it. Use the wildcard specification in case that no other application, like ptp4l from the `linuxptp` package, is using hardware timestamping on an interface. Multiple `hwtimestamp` directives are allowed in the chrony configuration file.

**Example 5.2. Enabling hardware timestamping by using the hwtimestamp directive**

```
hwtimestamp eth0
hwtimestamp eth1
hwtimestamp *
```

### 5.8.4. Configuring client polling interval

The default range of a polling interval (64-1024 seconds) is recommended for servers on the Internet. For local servers and hardware timestamping, a shorter polling interval needs to be configured in order to minimize offset of the system clock.

The following directive in `/etc/chrony.conf` specifies a local `NTP` server using one second polling interval:

```
server ntp.local minpoll 0 maxpoll 0
```

### 5.8.5. Enabling interleaved mode

`NTP` servers that are not hardware `NTP` appliances, but rather general purpose computers running a software `NTP` implementation, like chrony, will get a hardware transmit timestamp only after sending a packet. This behavior prevents the server from saving the timestamp in the packet to which it corresponds. In order to enable `NTP` clients receiving transmit timestamps that were generated after the transmission, configure the clients to use the `NTP` interleaved mode by adding the `xleave` option to the server directive in `/etc/chrony.conf`:

```
server ntp.local minpoll 0 maxpoll 0 xleave
```

### 5.8.6. Configuring server for large number of clients

The default server configuration allows a few thousands of clients at most to use the interleaved mode concurrently. To configure the server for a larger number of clients, increase the `clientloglimit` directive in `/etc/chrony.conf`. This directive specifies the maximum size of memory allocated for logging of clients' access on the server:

```
clientloglimit 100000000
```

### 5.8.7. Verifying hardware timestamping

To verify that the interface has successfully enabled hardware timestamping, check the system log. The log should contain a message from `chronyd` for each interface with successfully enabled hardware timestamping.

**Example 5.3. Log messages for interfaces with enabled hardware timestamping**

```
chronyd[4081]: Enabled HW timestamping on eth0
chronyd[4081]: Enabled HW timestamping on eth1
```

When `chronyd` is configured as an `NTP` client or peer, you can have the transmit and receive timestamping modes and the interleaved mode reported for each `NTP` source by the `chronyc ntpdata` command:

**Example 5.4. Reporting the transmit, receive timestamping and interleaved mode for each NTP source**

```
~]# chronyc ntpdata
```

Output:

```
Remote address  : 203.0.113.15 (CB00710F)
Remote port     : 123
Local address   : 203.0.113.74 (CB00714A)
Leap status     : Normal
Version         : 4
Mode            : Server
Stratum         : 1
Poll interval   : 0 (1 seconds)
Precision       : -24 (0.000000060 seconds)
Root delay      : 0.000015 seconds
Root dispersion : 0.000015 seconds
Reference ID    : 47505300 (GPS)
Reference time  : Wed May 03 13:47:45 2017
Offset          : -0.000000134 seconds
Peer delay      : 0.000005396 seconds
Peer dispersion : 0.000002329 seconds
Response time   : 0.000152073 seconds
Jitter asymmetry: +0.00
NTP tests       : 111 111 1111
Interleaved     : Yes
Authenticated   : No
TX timestamping : Hardware
RX timestamping : Hardware
Total TX        : 27
Total RX        : 27
Total valid RX  : 27
```

**Example 5.5. Reporting the stability of NTP measurements**

```
# chronyc sourcestats
```

With hardware timestamping enabled, stability of `NTP` measurements should be in tens or hundreds of nanoseconds, under normal load. This stability is reported in the `Std Dev` column of the output of the `chronyc sourcestats` command:

Output:

```
210 Number of sources = 1
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset
Std Dev
ntp.local                  12   7    11     +0.000      0.019     +0ns
49ns
```

### 5.8.8. Configuring PTP-NTP bridge

If a highly accurate Precision Time Protocol (PTP) grandmaster is available in a network that does not have switches or routers with PTP support, a computer may be dedicated to operate as a PTP slave and a stratum-1 NTP server. Such a computer needs to have two or more network interfaces, and be close to the grandmaster or have a direct connection to it. This will ensure highly accurate synchronization in the network.

Configure the ptp4l and phc2sys programs from the `linuxptp` packages to use one interface to synchronize the system clock using PTP.

Configure `chronyd` to provide the system time using the other interface:

**Example 5.6. Configuring chronyd to provide the system time using the other interface**

```
bindaddress 203.0.113.74
hwtimestamp eth1
local stratum 1
```

## 5.9. ACHIEVING SOME SETTINGS PREVIOUSLY SUPPORTED BY NTP IN CHRONY

Some settings that were in previous major version of Red Hat Enterprise Linux supported by ntp, are not supported by chrony. This section lists such settings, and describes ways to achieve them on a system with chrony.

### 5.9.1. Monitoring by ntpq and ntpdc

`chronyd` cannot be monitored by the ntpq and ntpdc utilities from the ntp distribution, because chrony does not support the NTP modes 6 and 7. It supports a different protocol and chronyc is the client implementation. For more information, see the `chronyc(1)` man page.

To monitor the status of the system clock sychronized by `chronyd`, you can:

- Use the tracking command
- Use the ntpstat utility, which supports chrony and provides a similar output as it used to with `ntpd`

**Example 5.7. Using the tracking command**

```
$ chronyc -n tracking
Reference ID    : 0A051B0A (10.5.27.10)
Stratum         : 2
```

```
Ref time (UTC)  : Thu Mar 08 15:46:20 2018
System time     : 0.000000338 seconds slow of NTP time
Last offset     : +0.000339408 seconds
RMS offset      : 0.000339408 seconds
Frequency       : 2.968 ppm slow
Residual freq   : +0.001 ppm
Skew            : 3.336 ppm
Root delay      : 0.157559142 seconds
Root dispersion : 0.001339232 seconds
Update interval : 64.5 seconds
Leap status     : Normal
```

**Example 5.8. Using the ntpstat utility**

```
$ ntpstat
synchronised to NTP server (10.5.27.10) at stratum 2
   time correct to within 80 ms
   polling server every 64 s
```

### 5.9.2. Using authentication mechanism based on public key cryptography

In Red Hat Enterprise Linux 7, ntp supported Autokey, which is an authentication mechanism based on public key cryptography. Autokey is not supported in `chronyd`.

On a Red Hat Enterprise Linux 8 system, it is recommended to use symmetric keys instead. Generate the keys with the `chronyc keygen` command. A client and server need to share a key specified in `/etc/chrony.keys`. The client can enable authentication using the `key` option in the `server`, `pool`, or `peer` directive.

### 5.9.3. Using ephemeral symmetric associations

In Red Hat Enterprise Linux 7, `ntpd` supported ephemeral symmetric associations, which can be mobilized by packets from peers which are not specified in the `ntp.conf` configuration file. In Red Hat Enterprise Linux 8, `chronyd` needs all peers to be specified in `chrony.conf`. Ephemeral symmetric associations are not supported.

Note that using the client/server mode enabled by the `server` or `pool` directive is more secure compared to the symmetric mode enabled by the `peer` directive.

### 5.9.4. multicast/broadcast client

Red Hat Enterprise Linux 7 supported the broadcast/multicast `NTP` mode, which simplifies configuration of clients. With this mode, clients can be configured to just listen for packets sent to a multicast/broadcast address instead of listening for specific names or addresses of individual servers, which may change over time.

In Red Hat Enterprise Linux 8, `chronyd` does not support the broadcast/multicast mode. The main reason is that it is less accurate and less secure than the ordinary client/server and symmetric modes.

There are several options of migration from an `NTP` broadcast/multicast setup:

- **Configure DNS to translate a single name, such as ntp.example.com, to multiple addresses of different servers**
  Clients can have a static configuration using only a single pool directive to synchronize with multiple servers. If a server from the pool becomes unreacheable, or otherwise unsuitable for synchronization, the clients automatically replace it with another server from the pool.

- **Distribute the list of `NTP` servers over DHCP**
  When NetworkManager gets a list of `NTP` servers from the DHCP server, `chronyd` is automatically configured to use them. This feature can be disabled by adding `PEERNTP=no` to the `/etc/sysconfig/network` file.

- **Use the `Precision Time Protocol (PTP)`**
  This option is suitable mainly for environments where servers change frequently, or if a larger group of clients needs to be able to synchronize to each other without having a designated server.

  `PTP` was designed for multicast messaging and works similarly to the `NTP` broadcast mode. A `PTP` implementation is available in the `linuxptp` package.

  `PTP` normally requires hardware timestamping and support in network switches to perform well. However, `PTP` is expected to work better than `NTP` in the broadcast mode even with software timestamping and no support in network switches.

  In networks with very large number of `PTP` slaves in one communication path, it is recommended to configure the `PTP` slaves with the `hybrid_e2e` option in order to reduce the amount of network traffic generated by the slaves. You can configure a computer running `chronyd` as an `NTP` client, and possibly `NTP` server, to operate also as a `PTP` grandmaster to distribute synchronized time to a large number of computers using multicast messaging.

## 5.10. ADDITIONAL RESOURCES

The following sources of information provide additional resources regarding chrony.

### 5.10.1. Installed Documentation

- `chronyc(1)` man page — Describes the chronyc command-line interface tool including commands and command options.

- `chronyd(8)` man page — Describes the `chronyd` daemon including commands and command options.

- `chrony.conf(5)` man page — Describes the chrony configuration file.

### 5.10.2. Online Documentation

- https://chrony.tuxfamily.org/doc/3.3/chronyc.html

- https://chrony.tuxfamily.org/doc/3.3/chronyd.html

- https://chrony.tuxfamily.org/doc/3.3/chrony.conf.html

For answers to FAQs, see https://chrony.tuxfamily.org/faq.html

# CHAPTER 6. USING PYTHON IN RED HAT ENTERPRISE LINUX 8

## 6.1. INTRODUCTION TO PYTHON

Python is a high-level programming language that supports multiple programming paradigms, such as object-oriented, imperative, functional, and procedural. Python has dynamic semantics and can be used for general-purpose programming.

With Red Hat Enterprise Linux, many packages that are installed on the system, such as packages providing system tools, tools for data analysis or web applications are written in Python. To be able to use these packages, you need to have the `python` packages installed.

### 6.1.1. Python versions

Two incompatible versions of Python are widely used, Python 2.x and Python 3.x.

Red Hat Enterprise Linux 8 uses Python 3.6 by default. However, Python 2.7 is also provided to support existing software.

> **WARNING**
>
> Neither the default `python` package nor the unversioned `/usr/bin/python` executable is distributed with Red Hat Enterprise Linux 8.

> **IMPORTANT**
>
> Always specify the major version of Python when installing it, invoking it, or otherwise interacting with it. For example, use `python3`, instead of `python`, in package and command names. All Python-related commands should also include the version, for example, `pip3` or `pip2`.
>
> Alternatively, configure the system default version by using the `alternatives` command as described in Configuring the unversioned Python.

As a system administrator, you are recommended to use preferably Python 3 for the following reasons:

- Python 3 represents the main development direction of the Python project.
- Support for Python 2 in the upstream community ends in 2020.
- Popular Python libraries are dropping Python 2 support in upstream.
- Python 2 in Red Hat Enterprise Linux 8 will have a shorter life cycle and its aim is to facilitate smoother transition to `Python 3` for customers.

For developers, Python 3 has the following advantages over Python 2:

- Python 3 allows writing expressive, maintainable, and correct code more easily.
- Code written in Python 3 will have greater longevity.
- Python 3 has new features, including asyncio, f-strings, advanced unpacking, keyword only arguments, chained exceptions and more.

However, existing software tends to require `/usr/bin/python` to be Python 2. For this reason, no default `python` package is distributed with Red Hat Enterprise Linux 8, and you can choose between using Python 2 and 3 as `/usr/bin/python`, as described in Section 6.2.3, “Configuring the unversioned Python”.

### 6.1.2. The internal platform-python package

System tools in Red Hat Enterprise Linux 8 use a Python version 3.6 provided by the internal `platform-python` package. Red Hat advises customers to use the `python36` package instead.

## 6.2. INSTALLING AND USING PYTHON

**WARNING**

Using the unversioned `python` command to install or run Python does not work by default due to ambiguity. Always specify the major version of Python as described in Installing and using Python 3 and Installing and using Python 2, or configure the system default version by using the `alternatives` command as described in Section 6.2.3, “Configuring the unversioned Python”.

### 6.2.1. Installing and using Python 3

In Red Hat Enterprise Linux 8, Python 3 is distributed as the `python36` module in the AppStream repository.

For details regarding modules, see Using Application Stream.

#### 6.2.1.1. Installing Python 3

To install Python 3, use the following command as root:

```
yum install python3
```

This command installs Python 3.6 from the `python36` module in AppStream.

#### 6.2.1.2. Using Python 3

To run Python 3, use the `python3` command. Use the version in all other related commands, such as `pip3`.

#### 6.2.1.3. Naming conventions for Python 3 packages

Packages with add-on modules for Python 3 generally use the `python3-` prefix.

For example, to install the Requests module that is used for writing HTTP clients, run:

```
yum install python3-requests
```

### 6.2.2. Installing and using Python 2

Some software has not yet been fully ported to Python 3, and needs Python 2 to operate. Red Hat Enterprise Linux 8 allows parallel installation of Python 3 and Python 2. If you need the Python 2 functionality, install the `python27` module, which is available in the AppStream repository.

For details regarding modules, see Using Application Stream.

> **WARNING**
>
> Note that Python 3 is the main development direction of the Python project. The support for Python 2 is being phased out. The `python27` module has a shorter support period than Red Hat Enterprise Linux 8.

#### 6.2.2.1. Installing Python 2

To install Python 2, run as root:

```
yum install python2
```

This command installs Python 2.7 from the `python27` module in AppStream.

#### 6.2.2.2. Using Python 2

To run Python 2, use the `python2` command. Use the version in all other related commands, such as `pip2`.

##### 6.2.2.2.1. Naming conventions for Python 2 packages

Packages with add-on modules for Python 2 generally use the `python2-` prefix.

For example, to install the Requests module that is used for writing HTTP clients, run:

```
yum install python2-requests
```

### 6.2.3. Configuring the unversioned Python

System administrators can configure the unversioned `python` command on the system using the `alternatives` command. Note that the required package, either `python3` or `python2`, needs to be installed before configuring the unversioned command to the respective version.

To configure the unversioned `python` command to Python 3 directly, run:

```
alternatives --set python /usr/bin/python3
```

Use an analogous command if you choose Python 2.

Alternatively, you can configure the unversioned `python` command interactively:

1. Run the following command:

   ```
   alternatives --config python
   ```

2. Select the required version from the provided list.

To reset this configuration and remove the unversioned `python` command, run:

```
alternatives --auto python
```

> **WARNING**
>
> Additional Python-related commands, such as `pip3`, do not have configurable unversioned variants.

## 6.3. MIGRATING FROM PYTHON 2 TO PYTHON 3

As a developer, you may want to migrate your former code that is written in Python 2 to Python 3. For more information on how to migrate large code bases to Python 3, see The Conservative Python 3 Porting Guide.

Note that after this migration, the original Python 2 code becomes interpretable by the Python 3 interpreter and stays interpretable for the Python 2 interpreter as well.

## 6.4. PACKAGING OF PYTHON 3 RPMS

Most Python projects use Setuptools for packaging, and define package information in the `setup.py` file. For more information on Setuptools packaging, see Setuptools documentation.

You can also package your Python project into an RPM package, which provides the following advantages compared to Setuptools packaging:

- Specification of dependencies of a package on other RPMs (even non-Python)
- Cryptographic signing
  With cryptographic signing, content of RPM packages can be verified, integrated, and tested with the rest of the operating system.

For more information on RPM packaging, see the RPM Packaging Guide.

### 6.4.1. Typical SPEC file description for a Python RPM package

A SPEC file contains instructions that the `rpmbuild` utility uses to build an RPM. The instructions are included in a series of sections. A SPEC file has two main parts in which the sections are defined:

- Preamble (contains a series of metadata items that are used in the Body)
- Body (contains the main part of the instructions)

For further information about SPEC files, see the RPM Packaging Guide.

An RPM SPEC file for Python projects has some specifics compared to non-Python RPM SPEC files. Most notably, a name of any RPM package of a Python library must always include the `python3` prefix.

Other specifics are shown in the following SPEC file example for the `python3-detox` package. For description of such specifics, see the notes below the example.

```
%global modname detox
1

Name:           python3-detox
2
Version:        0.12
Release:        4%{?dist}
Summary:        Distributing activities of the tox tool
License:        MIT
URL:            https://pypi.io/project/detox
Source0:        https://pypi.io/packages/source/d/%{modname}/%{modname}-%
{version}.tar.gz

BuildArch:      noarch

BuildRequires:  python36-devel
3
BuildRequires:  python3-setuptools
BuildRequires:  python36-rpm-macros
BuildRequires:  python3-six
BuildRequires:  python3-tox
BuildRequires:  python3-py
BuildRequires:  python3-eventlet

%?python_enable_dependency_generator
4

%description

Detox is the distributed version of the tox python testing tool. It makes
efficient use of multiple CPUs by running all possible activities in
parallel.
Detox has the same options and configuration that tox has, so after
installation you can run it in the same way and with the same options that
you use for tox.

    $ detox
```

```
%prep
%autosetup -n %{modname}-%{version}

%build
%py3_build
5

%install
%py3_install

%check
%{__python3} setup.py test
6

%files -n python3-%{modname}
%doc CHANGELOG
%license LICENSE
%{_bindir}/detox
%{python3_sitelib}/%{modname}/
%{python3_sitelib}/%{modname}-%{version}*

%changelog
...
```

1. The modname macro contains the name of the Python project. In this example it is `detox`.

2. When packaging a Python project into RPM, the `python3` prefix always needs to be added to the original name of the project. The original name here is `detox` and the name of the RPM is `python3-detox`.

3. BuildRequires specifies what packages are required to build and test this package. In BuildRequires, always include items providing tools necessary for building Python packages: `python36-devel` and `python3-setuptools`. The `python36-rpm-macros` package is required so that files with `/usr/bin/python3` shebangs are automatically changed to `/usr/bin/python3.6`. For more information, see Section 6.4.4, "Handling hashbangs in Python scripts".

4. Every Python package requires some other packages to work correctly. Such packages need to be specified in the SPEC file as well. To specify the dependencies, you can use the %python_enable_dependency_generator macro to automatically use dependencies defined in the `setup.py` file. If a package has dependencies that are not specified using Setuptools, specify them within additional `Requires` directives.

5. The %py3_build and %py3_install macros run the `setup.py build` and `setup.py install` commands, respectively, with additional arguments to specify installation locations, the interpreter to use, and other details.

6. The check section provides a macro that runs the correct version of Python. The %{__python3} macro contains a path for the Python 3 interpreter, for example `/usr/bin/python3`. We recommend to always use the macro rather than a literal path.

### 6.4.2. Common macros for Python 3 RPM packages

In a SPEC file, always use the macros below rather than hardcoding their values.

In macro names, always use `python3` or `python2` instead of unversioned `python`.

| Macro | Normal Definition | Decription |
|---|---|---|
| %{__python3} | /usr/bin/python3 | Python 3 interpreter |
| %{python3_version} | 3.6 | The full version of the Python 3 interpreter. |
| %{python3_sitelib} | /usr/lib/python3.6/site-packages | Where pure-Python modules are installed. |
| %{python3_sitearch} | /usr/lib64/python3.6/site-packages | Where modules containing architecture-specific extensions are installed. |
| %py3_build | | Runs the **`setup.py build`** command with arguments suitable for a system package. |
| %py3_install | | Runs the **`setup.py install`** command with arguments suitable for a system package. |

### 6.4.3. Automatic provides for Python RPM packages

When packaging a Python project, make sure that, if present, the following directories are included in the resulting RPM:

- **`.dist-info`**
- **`.egg-info`**
- **`.egg-link`**

From these directories, the RPM build process automatically generates virtual `pythonX.Ydist` provides, for example `python3.6dist(detox)`. These virtual provides are used by packages that are specified by the %python_enable_dependency_generator macro.

### 6.4.4. Handling hashbangs in Python scripts

In Red Hat Enterprise Linux 8, executable Python scripts are expected to use hashbangs (shebangs) specifying explicitly at least the major Python version.

The `/usr/lib/rpm/redhat/brp-mangle-shebangs` buildroot policy (BRP) script is run automatically when building any RPM package, and attempts to correct hashbangs in all executable files. The BRP script will generate errors when encountering a Python script with an ambiguous hashbang, such as:

```
#! /usr/bin/python
```

or

```
#! /usr/bin/env python
```

To modify hashbangs in the Python scripts causing these build errors at RPM build time, use the `pathfix.py` script from the `platform-python-devel` package:

```
pathfix.py -pn -i %{__python3} PATH …
```

Multiple *PATH*s can be specified. If a *PATH* is a directory, `pathfix.py` recursively scans for any Python scripts matching the pattern `^[a-zA-Z0-9_]+\.py$`, not only those with an ambiguous hashbang. Add this command to the `%prep` section or at the end of the `%install` section.

Alternatively, modify the packaged Python scripts so that they conform to the expected format. For this purpose, `pathfix.py` can be used outside the RPM build process, too. When running `pathfix.py` outside a RPM build, replace `__python3` from the example above with a path for the hashbang, such as `/usr/bin/python3`.

If the packaged Python scripts require Python version 2, replace the number 3 with 2 in the commands above.

Additionally, hashbangs in the form `/usr/bin/python3` are by default replaced with hashbangs pointing to Python from the `platform-python` package used for system tools with Red Hat Enterprise Linux.

To change the `/usr/bin/python3` hashbangs in their custom packages to point to a version of Python installed from Application Stream, in the form `/usr/bin/python3.6`, add the `python36-rpm-macros` package into the BuildRequires section of the SPEC file:

```
BuildRequires:  python36-rpm-macros
```

**NOTE**

To prevent hashbang check and modification by the BRP script, use the following RPM directive:

```
%undefine %brp_mangle_shebangs
```

# CHAPTER 7. INSTALLING AND USING LANGPACKS

## 7.1. INTRODUCTION TO LANGPACKS

Langpacks are meta-packages which install extra add-on packages containing translations, dictionaries and locales for every package installed on the system.

On a Red Hat Enterprise Linux 8 system, langpacks installation is based on the `langpacks-<langcode>` language meta-packages and RPM weak dependencies (Supplements tag).

There are two prerequisites to be able to use langpacks for a selected language:

1. The `langpacks-<langcode>` language meta-package for the selected language has been installed on the system.
   On Red Hat Enterprise Linux 8, the langpacks meta packages are installed automatically with the initial installation of the operating system using the Anaconda installer, because these packages are available in the in Application Stream repository.

   To find which languages provide langpacks, execute the following command:

   ```
   ~]# yum list langpacks-*
   ```

2. The base package, for which you want to search the local packages, has already been installed on the system.

If these prerequisites are fulfilled, the language meta-packages pull their langpack for the selected language automatically in the transaction set.

## 7.2. WORKING WITH RPM WEAK DEPENDENCY-BASED LANGPACKS

### 7.2.1. Querying RPM weak dependency-based langpacks

To list the already installed language support, run the following command:

```
~]# yum list installed langpacks*
```

To check if any language support is available for another language, run the following command:

```
~]# yum list available langpacks*
```

To list what packages get installed for any language, run the following command:

```
~]# yum repoquery --whatsupplements langpacks-<locale_code>
```

### 7.2.2. Installing language support

To add new a language support, run the following command as the root user:

```
~]# yum install langpacks-<locale_code>
```

### 7.2.3. Removing language support

To remove any installed language support, run the following command as the root user:

```
~]# yum remove langpacks-<locale_code>
```

## 7.3. SAVING DISK SPACE BY USING GLIBC-LANGPACK-<LOCALE_CODE>

Currently, all locales are stored in the `/usr/lib/locale/locale-archive` file, which requires a lot of disk space.

On systems where disk space is a critical issue, such as containers and cloud images, or only a few locales are needed, you can use the glibc locale langpack packages (`glibc-langpack-<locale_code>`). By using the following command instead of that described in Section 7.2.2, “Installing language support”, you can install locales individually and thus gain a smaller package installation footprint:

```
~]# yum install glibc-langpack-<locale_code>
```

When installing the operating system with Anaconda, `glibc-langpack-<locale_code>` is installed for the language you used during the installation and also for the languages you selected as additional languages. Note that `glibc-all-langpacks`, which contains all locales, is installed by default, so some locales are duplicated. If you installed `glibc-langpack-<locale_code>` for one or more selected languages, you can delete `glibc-all-langpacks` after the installation to save the disk space.

Note that installing only selected `glibc-langpack-<locale_code>` packages instead of `glibc-all-langpacks` has impact on run time performance.

**NOTE**

If disk space is not an issue, keep all locales installed by using the `glibc-all-langpacks` package.

# CHAPTER 8. GETTING STARTED WITH TCL/TK ON RED HAT ENTERPRISE LINUX 8

## 8.1. INTRODUCTION TO TCL/TK

Tool command language (Tcl) is a dynamic programming language. The interpreter for this language, together with the C library, is provided by the `tcl` package.

Using Tcl paired with Tk (Tcl/Tk) enables creating cross-platform GUI applications. Tk is provided by the `tk` package.

Note that Tk can refer to any of the the following:

- A programming toolkit for multiple languages
- A Tk C library bindings available for multiple languages, such as C, Ruby, Perl and Python
- A wish interpreter that instantiates a Tk console
- A Tk extension that adds a number of new commands to a particular Tcl interpreter

For more information about Tcl/Tk, see the Tcl/Tk manual or Tcl/Tk documentation web page.

## 8.2. NOTABLE CHANGES IN TCL/TK 8.6

Major changes in Tcl/Tk 8.6 compared to Tcl/Tk 8.5 are:

- Object-oriented programming support
- Stackless evaluation implementation
- Enhanced exceptions handling
- Collection of third-party packages built and installed with Tcl
- Multi-thread operations enabled
- SQL database-powered scripts support
- IPv6 networking support
- Built-in Zlib compression
- List processing
  Two new commands, `lmap` and `dict map` are available, which allow the expression of transformations over Tcl containers.
- Stacked channels by script
  Two new commands, `chan push` and `chan pop` are available, which allow to add or remove transformations to or from I/O channels.

Major changes in Tk include:

- Built-in PNG image support

- Busy windows
  A new command, `tk busy` is available, which disables user interaction for a window or a widget and shows the busy cursor.

- New font selection dialog interface

- Angled text support

- Moving things on a canvas support

For the detailed list of changes between Tcl 8.5 and Tcl 8.6, see Changes in Tcl/Tk 8.6.

## 8.3. MIGRATING TO TCL/TK 8.6

Red Hat Enterprise Linux 7 used Tcl/Tk 8.5. With Red Hat Enterprise Linux 8, Tcl/Tk version 8.6 is provided in the Base OS repository.

This section describes migration path to Tcl/Tk 8.6 for:

- developers writing Tcl extensions or embedding Tcl interpreter into their applications (see Section 8.3.1, “Migration path for developers of Tcl extensions”)

- users scripting tasks with Tcl/Tk (see Section 8.3.2, “Migration path for users scripting their tasks with Tcl/Tk”)

### 8.3.1. Migration path for developers of Tcl extensions

Tcl 8.6 limits direct access to members of the `interp` structure. For example, if your code reads `interp→errorLine`, rewrite the code to use the `Tcl_GetErrorLine(interp)` function.

To make your code compatible with both Tcl 8.5 and Tcl 8.6, use the following code snippet in a header file of your C or C++ application or extension that includes the Tcl library:

```
# include <tcl.h>
# if !defined(Tcl_GetErrorLine)
# define Tcl_GetErrorLine(interp) (interp→errorLine)
# endif
```

### 8.3.2. Migration path for users scripting their tasks with Tcl/Tk

In Tcl 8.6, most scripts work the same way as with the previous version of Tcl. When writing a portable code, make sure to not use the commands that are no longer supported in Tk 8.6:

- tkIconList_Arrange
- tkIconList_AutoScan
- tkIconList_Btn1
- tkIconList_Config
- tkIconList_Create
- tkIconList_CtrlBtn1

- **tkIconList_Curselection**
- **tkIconList_DeleteAll**
- **tkIconList_Double1**
- **tkIconList_DrawSelection**
- **tkIconList_FocusIn**
- **tkIconList_FocusOut**
- **tkIconList_Get**
- **tkIconList_Goto**
- **tkIconList_Index**
- **tkIconList_Invoke**
- **tkIconList_KeyPress**
- **tkIconList_Leave1**
- **tkIconList_LeftRight**
- **tkIconList_Motion1**
- **tkIconList_Reset**
- **tkIconList_ReturnKey**
- **tkIconList_See**
- **tkIconList_Select**
- **tkIconList_Selection**
- **tkIconList_ShiftBtn1**
- **tkIconList_UpDown**

**You can check the list of unsupported commands also in the `/usr/share/tk8.6/unsupported.tcl` file.**

# CHAPTER 9. USING PREFIXDEVNAME FOR NAMING OF ETHERNET NETWORK INTERFACES

This documentation describes how to set the prefixes for consistent naming ot Ethernet network interfaces in case that you do not want to use the default naming scheme of such interfaces.

However, Red Hat recommends to use the default naming scheme, which is the same as in Red Hat Enterprise Linux 7. For more details about this scheme, see Consistent Network Device Naming.

## 9.1. INTRODUCTION TO PREFIXDEVNAME

The `prefixdevname` tool is a udev helper utility that enables you to define your own prefix used for naming of the Ethernet network interfaces.

## 9.2. SETTING PREFIXDEVNAME

The setting of the prefix with `prefixdevname` is done during system installation.

To set and activate the required prefix for your Ethernet network interfaces, add the following on the kernel command line:

```
net.ifnames.prefix=<required prefix>
```

> **WARNING**
>
> Red Hat does not support the use of `prefixdevname` on already deployed systems.

After the prefix was once set, and the operating system was rebooted, the prefix is effective every time when a new network interface appears. The new device is assigned a name in the form of <PREFIX><INDEX>. For example, if your selected prefix is `net`, and the interfaces with `net0` and `net1` prefixes already exist on the system, the new interface is named`net2`. The `prefixdevname` utility then generates the new `.link` file in the `/etc/systemd/network` directory that applies the name to the interface with the MAC address that just appeared. The configuration is persistent across reboots.

## 9.3. LIMITATIONS OF PREFIXDEVNAME

There are certain limitations for prefixes of Ethernet network interfaces.

The prefix that you choose must meet the following requirements:

- be ASCII string
- be alphanumeric string
- be shorter than 16 characters

**WARNING**

**The prefix cannot conflict with any other well-known prefix used for network interface naming on Linux. Specifically, you cannot use these prefixes: `eth`, `eno`, `ens`, `em`.**