

Managing Services using 'systemctl'

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. What is ‘systemctl’ ? 1
2. What is ‘systemd’ ? 2
3. Relationship between ‘systemd’ and ‘systemctl’ 3
4. Why was ‘systemd’ introduced? 3
5. Basic ‘systemctl’ syntax 4
6. What is a ‘systemd’ unit ? 5
7. Services 5
8. Checking whether systemd is running 6
9. systemctl –help 6
10. ‘`systemctl’ status` 7
11. Understanding `Active: active (running)` 7
12. `active (exited)` 8
13. `inactive (dead)` 8
14. `failed` 8
15. Starting a service 9
16. Stopping a service 9
17. Restarting a service 10
18. Reloading a service 10
19. Restart vs Reload 11
20. `reload-or-restart` 11
21. Enabling a service 12
22. Enable and start together 12
23. Disabling a service 13
24. `is-active` 13
25. `is-enabled` 14
26. `is-failed` 14
27. Listing running services 14

28. Listing all installed service units 15
29. Listing failed services 15
30. Listing all unit types 16
31. Viewing unit dependencies 16
32. Why dependencies matter 17
33. `systemctl cat` 17
34. `systemctl show` 17
35. `systemctl edit` 18
36. Why not directly edit `/lib/systemd/system/`? 18
37. Daemon reload 19
38. `daemon-reexec` 19
39. Masking a service 20
40. Unmasking a service 20
41. Disable vs mask 20
42. `systemctl reboot` 21
43. `systemctl poweroff` 21
44. `systemctl halt` 22
45. `systemctl suspend` 22
46. `systemctl hibernate` 22
47. `systemctl hybrid-sleep` 22
48. Targets 23
49. Changing the default target 23
50. Switching targets 23
51. Rescue mode 24
52. Emergency mode 24
53. System state 24
54. What does `degraded` mean ? 25
55. `systemctl status` 25
56. Viewing boot information 26
57. Boot time analysis 26

58. `systemctl list-jobs` 26
59. Service logs 27
60. Viewing recent service logs 27
61. Service troubleshooting workflow 27
62. Checking service configuration 28
63. Finding unit files 28
64. `systemctl list-unit-files` 29
65. Static units 29
66. Generated units 29
67. Alias units 30
68. Service status with `–no-pager` 30
69. Avoiding colored output 30
70. Quiet mode 31
71. Using `systemctl` in shell scripts 31
72. Checking whether a service is enabled in a script 31
73. Root privileges 32
74. Why `sudo` is required 32
75. `systemctl` and processes 33
76. Main PID 33
77. Service states 33
78. Load state 34
79. Active state 34
80. Sub-state 34
81. `systemctl reset-failed` 35
82. Restarting a failed service 35
83. `try-restart` 36
84. `condrestart` 36
85. `start` vs `enable` 37
86. Example 37
87. `enable –now` 38

88. `disable –now` 38
89. Service dependencies 39
90. `After=` does not necessarily mean dependency 39
91. `Requires=` vs `Wants=` 40
92. Unit dependency graph 40
93. Timers 41
94. Why systemd timers are useful 41
95. Example timer concept 42
96. Socket units 42
97. Mount units 43
98. Device units 43
99. Path units 43
100. Slice units 44
101. Scope units 44
102. User-level systemd 44
103. System-wide vs user services 45
104. `systemctl --user list-units` 45
105. Boot target and graphical.target 45
106. multi-user.target 46
107. rescue.target 46
108. emergency.target 46
109. Boot process overview 46
110. Why PID 1 is special 47
111. What happens when a service crashes? 47
112. Restart policies 48
113. Viewing restart configuration 48
114. Service startup timeout 49
115. Environment variables 49
116. Security features 49
117. Running a service as another user 50
118. Viewing service resource information 50

119. cgroups 50
120. Why `systemctl` is so important 51
121. Common commands - quick reference 52
122. Most important commands to memorize 53
123. Practical example - SSH 53
124. Practical example - Apache 54
125. Practical example - NetworkManager 55
126. Practical example - failed service 55
127. Practical example - service won't start 56
128. Checking current boot logs 57
129. Combining `systemctl` and `journalctl` 57
130. `systemctl` vs `service` 57
131. `systemctl` vs `/etc/init.d/` 58
132. Why systemd can start services in parallel 58
133. Important caution about `systemctl` 59
134. Common mistakes 59
135. Exam questions and answers 61
136. Easy way to remember the main commands 63
137. Complete practical workflow 64
138. Final mental model 65
139. The 15 commands you should remember 65
140. One-page revision 66

systemctl

`systemctl` is one of the **most important commands in modern Linux**, especially on systems that use **systemd** as their initialization and service-management system. It is used to **start, stop, restart, enable, disable, inspect, monitor, and manage system services and other systemd units**.

If you are learning Linux commands for an exam, interview, system administration, or practical work, understanding **`systemctl`** is extremely useful.

---

1. What is ‘`systemctl`’ ?

The basic syntax is:

```
systemctl [OPTIONS] COMMAND [UNIT]
```

For example:

```
systemctl status ssh
```

This asks systemd for the current status of the SSH service.

Another example:

```
sudo systemctl restart apache2
```

This restarts the Apache web server.

In simple words:

> **`systemctl` is the command-line tool used to communicate with and control the systemd system and service manager.**

2. What is ‘`systemd`’ ?

Before understanding **`systemctl`**, you need to understand **`systemd`**.

`systemd` is a system and service manager used by many Linux distributions.

Examples include:

- Ubuntu
- Debian
- Fedora
- RHEL
- CentOS
- Rocky Linux
- AlmaLinux
- Arch Linux
- openSUSE

When a Linux computer starts, many things have to happen:

1. The kernel starts.
2. Hardware is detected.
3. Filesystems are mounted.
4. Networking is initialized.
5. Logging starts.
6. Services are started.
7. Users can log in.
8. Background processes are launched.

`systemd` coordinates much of this process.

You can think of it as the **manager of the Linux operating system after the kernel has started**.

3. Relationship between ‘systemd’ and ‘systemctl’

A simple way to remember the relationship:

```
systemd
    │
    ├── manages services
    ├── manages processes
    ├── manages boot
    ├── manages sockets
    ├── manages mounts
    ├── manages timers
    ├── manages targets
    └── manages other units
            │
            ↓
       systemctl
            │
            ↓
   Administrator
```

systemd does the actual management.

systemctl is one of the primary interfaces administrators use to tell systemd what to do.

For example:

```
sudo systemctl start ssh
```

means:

> Tell systemd to start the SSH service.

---

4. Why was ‘systemd’ introduced?

Traditional Linux systems used several different mechanisms to manage startup and services.

Older systems commonly used:

```
SysVinit
```

with commands and scripts such as:

```
service
```

and:

```
/etc/init.d/
```

systemd was designed to provide a more integrated approach.

It can manage:

- services
- startup
- dependencies
- sockets
- devices
- mount points
- timers
- targets
- user sessions
- resource controls
- logging integration

---

5. Basic ‘`systemctl`’ syntax

The general format is:

```
systemctl COMMAND UNIT
```

For example:

```
systemctl status ssh
```

Here:

```
systemctl → command
status    → operation
ssh       → unit
```

With administrative privileges:

```
sudo systemctl restart ssh
```

6. What is a ‘systemd’ unit?

One of the most important concepts is the **unit**.

A unit is an object managed by systemd.

There are several types.

Common types include:

Unit type	Extension	Purpose
Service	`.service`	System services
Socket	`.socket`	IPC/network sockets
Target	`.target`	Groups of units
Mount	`.mount`	Filesystem mounts
Automount	`.automount`	Automatic mounting
Timer	`.timer`	Scheduled activation
Device	`.device`	Kernel devices
Path	`.path`	Monitors filesystem paths
Swap	`.swap`	Swap devices/files
Scope	`.scope`	External processes
Slice	`.slice`	Resource management

The most commonly encountered type is:

```
.service
```

---

7. Services

A service represents a background program or daemon.

Examples:

```
ssh.service
apache2.service
nginx.service
cron.service
systemd-resolved.service
```

Usually you can omit `.service`.

For example:

```
systemctl status ssh
```

is generally equivalent to:

```
systemctl status ssh.service
```

---

8. Checking whether systemd is running

You can use:

```
ps -p 1 -o comm=
```

On a system using systemd, you will normally see:

```
systemd
```

Another command:

```
systemctl --version
```

This displays information about the installed systemd version.

Example:

```
systemd 249
```

The exact version depends on your Linux distribution.

---

9. `systemctl --help`

To see available options:

```
systemctl --help
```

This provides a list of commands and options.

You can also use:

```
man systemctl
```

The manual page is particularly useful because `systemctl` has many capabilities.

10. ‘systemctl’ status

One of the most frequently used commands is:

```
systemctl status SERVICE
```

Example:

```
systemctl status ssh
```

It provides information such as:

- whether the service is running
- whether it failed
- when it started
- process ID
- recent log messages
- memory usage
- CPU usage

Example output may look similar to:

```
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded
     Active: active (running)
   Main PID: 1234 (sshd)
      Tasks: 1
     Memory: 3.2M
```

---

11. Understanding `Active: active (running)`

Suppose you see:

```
Active: active (running)
```

This generally means:

> The service is currently running successfully.

Other possible states include:

```
active (exited)
inactive (dead)
failed
activating
deactivating
```

12. `active (exited)`

You may encounter:

```
Active: active (exited)
```

This does not necessarily mean something is wrong.

Some services perform a task and then exit.

For example, a oneshot service can run a command and terminate successfully.

So:

```
active (exited)
```

can mean:

> The service completed successfully but does not remain running.

---

13. `inactive (dead)`

If you see:

```
Active: inactive (dead)
```

the service is currently not running.

This may be completely normal for some services.

For a service that should always be running, however, it may indicate a problem.

---

14. `failed`

If you see:

```
Active: failed
```

systemd attempted to activate the service but the service failed.

For example:

```
systemctl status apache2
```

might show:

```
Active: failed
```

You should then inspect the logs.

For example:

```
journalctl -u apache2
```

---

15. Starting a service

Use:

```
sudo systemctl start SERVICE
```

Example:

```
sudo systemctl start ssh
```

This starts the SSH service.

Important:

> `start` normally affects the **current running system**. It does not necessarily configure the service to start automatically at the next boot.

---

16. Stopping a service

Use:

```
sudo systemctl stop SERVICE
```

Example:

```
sudo systemctl stop apache2
```

This stops the Apache service.

Be careful when stopping important services.

For example:

```
sudo systemctl stop NetworkManager
```

could disrupt your network connection depending on your system configuration.

17. Restarting a service

Use:

```
sudo systemctl restart SERVICE
```

Example:

```
sudo systemctl restart ssh
```

Restarting generally means:

```
Stop service
     ↓
Start service
```

This is commonly used after changing a service's configuration.

For example:

```
sudo nano /etc/ssh/sshd_config
```

After making an appropriate configuration change:

```
sudo systemctl restart ssh
```

---

18. Reloading a service

Another important operation is:

```
sudo systemctl reload SERVICE
```

Reload means:

> Ask the running service to reread its configuration without completely stopping and starting it.

For example:

```
sudo systemctl reload nginx
```

Whether reload works depends on the service.

Some services support reload; others do not.

19. Restart vs Reload

This is an important distinction.

Restart

```
sudo systemctl restart nginx
```

Conceptually:

```
Stop
 ↓
Start
```

Reload

```
sudo systemctl reload nginx
```

Conceptually:

```
Keep process running
       ↓
Reread configuration
```

Reload is often preferable when supported because it can reduce service interruption.

---

20. `reload-or-restart`

Another useful command is:

```
sudo systemctl reload-or-restart SERVICE
```

This means:

> Try to reload the service; if reload is not supported, restart it.

Example:

```
sudo systemctl reload-or-restart nginx
```

21. Enabling a service

Suppose you want SSH to start automatically when Linux boots.

You can use:

```
sudo systemctl enable ssh
```

This configures the service to be started automatically during the appropriate boot target.

Important distinction:

```
systemctl start ssh
```

starts it **now**.

```
systemctl enable ssh
```

configures it to start **automatically in the future**.

---

22. Enable and start together

You can do both:

```
sudo systemctl enable --now ssh
```

This means:

```
Enable service
     +
Start service now
```

This is very convenient.

For example:

```
sudo systemctl enable --now nginx
```

23. Disabling a service

Use:

```
sudo systemctl disable SERVICE
```

Example:

```
sudo systemctl disable apache2
```

This prevents the service from being automatically started through its normal enablement configuration.

It does **not necessarily stop the service immediately**.

To stop it now as well:

```
sudo systemctl disable --now apache2
```

---

24. `is-active`

To check whether a service is currently active:

```
systemctl is-active ssh
```

Possible output:

```
active
```

or:

```
inactive
```

or:

```
failed
```

This command is useful in scripts.

For example:

```
systemctl is-active --quiet ssh
```

can be used for conditional logic.

25. `is-enabled`

To check whether a service is enabled:

```
systemctl is-enabled ssh
```

Possible output:

```
enabled
```

Other possible states include:

```
disabled
static
masked
indirect
generated
```

---

26. `is-failed`

You can check whether a unit is in a failed state:

```
systemctl is-failed ssh
```

For example:

```
failed
```

or:

```
active
```

---

27. Listing running services

To list currently active units:

```
systemctl list-units
```

To focus on services:

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

This is extremely useful when troubleshooting.

28. Listing all installed service units

Use:

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

This is different from:

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

The distinction is important.

`list-units`

Shows units currently loaded/known to systemd, commonly emphasizing active units.

`list-unit-files`

Shows installed unit files and their enablement state.

---

29. Listing failed services

One of the best troubleshooting commands is:

```
systemctl --failed
```

or:

```
systemctl list-units --failed
```

This helps identify services that failed to start.

Example:

```
UNIT              LOAD   ACTIVE SUB    DESCRIPTION
apache2.service   loaded failed failed Apache Web Server
```

30. Listing all unit types

You can use:

```
systemctl list-units
```

You can also specify a type.

For example:

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

systemctl list-units --type=socket

systemctl list-units --type=timer

systemctl list-units --type=mount
```

---

31. Viewing unit dependencies

A powerful feature is:

```
systemctl list-dependencies
```

For a specific service:

```
systemctl list-dependencies ssh
```

This shows relationships between units.

For example:

```
ssh.service
├─system.slice
├─sshd-keygen.target
└─sysinit.target
```

The exact output varies.

32. Why dependencies matter

Linux services often depend on other components.

For example:

```
Network
   ↓
Database
   ↓
Web application
```

systemd needs to understand these relationships.

If service A depends on service B, systemd can arrange activation appropriately.

This is one of the major advantages of systemd's dependency model.

---

33. `systemctl cat`

You can display the unit configuration:

```
systemctl cat ssh
```

This is useful because it shows the service unit file and possible drop-in configuration.

For example:

```
systemctl cat nginx
```

---

34. `systemctl show`

For detailed machine-readable properties:

```
systemctl show ssh
```

This can produce many properties.

For example:

```
Id=ssh.service
Type=notify
LoadState=loaded
ActiveState=active
SubState=running
```

You can query a specific property:

```
systemctl show ssh -p ActiveState
```

Possible result:

```
ActiveState=active
```

---

35. `systemctl edit`

You can create an override configuration for a service:

```
sudo systemctl edit SERVICE
```

For example:

```
sudo systemctl edit nginx
```

This is generally preferred over directly modifying packaged unit files.

An override can be used to customize settings while keeping distribution-provided files intact.

---

36. Why not directly edit `/lib/systemd/system/`?

You may find unit files under locations such as:

```
/lib/systemd/system/
```

or:

```
/usr/lib/systemd/system/
```

Depending on the distribution.

Directly modifying package-managed unit files is generally a bad practice because package updates may overwrite your changes.

Instead, use:

```
sudo systemctl edit SERVICE
```

for overrides.

37. Daemon reload

After modifying unit definitions, you may need:

```
sudo systemctl daemon-reload
```

This tells systemd to reload its unit configuration.

Important:

`daemon-reload`

does **not** restart every service.

It means:

> Re-read unit files and rebuild systemd's understanding of the configuration.

For example:

```
sudo systemctl edit myservice
sudo systemctl daemon-reload
sudo systemctl restart myservice
```

---

38. `daemon-reexec`

Another command is:

```
sudo systemctl daemon-reexec
```

This is different from:

```
systemctl daemon-reload
```

`daemon-reload` rereads unit configuration.

`daemon-reexec` causes the systemd manager itself to re-execute.

It is generally used for specialized system administration situations rather than routine service configuration.

39. Masking a service

You can prevent a service from being started manually or automatically:

```
sudo systemctl mask SERVICE
```

For example:

```
sudo systemctl mask apache2
```

A masked unit is typically linked to:

```
/dev/null
```

This is stronger than simply disabling it.

---

40. Unmasking a service

To remove the mask:

```
sudo systemctl unmask apache2
```

After that, the service can again be started normally.

---

41. Disable vs mask

This is an important exam/interview question.

Disable

```
systemctl disable apache2
```

Means:

> Don't automatically start it through its normal enablement configuration.

It may still be manually started.

Mask

```
systemctl mask apache2
```

Means:

> Prevent the unit from being started through normal systemd activation.

So:

```
disable < mask
```

in terms of how strongly activation is prevented.

---

42. `systemctl reboot`

You can reboot the system using:

```
sudo systemctl reboot
```

This asks systemd to perform a system reboot.

Equivalent alternatives may include:

```
sudo reboot
```

depending on the system.

---

43. `systemctl poweroff`

To shut down and power off:

```
sudo systemctl poweroff
```

This is useful for system administration.

44. `systemctl halt`

You can halt the system:

```
sudo systemctl halt
```

However, `halt` and `poweroff` are not conceptually identical.

`halt` stops the system, while `poweroff` requests powering off the machine.

Exact hardware behavior depends on the platform and configuration.

---

45. `systemctl suspend`

To suspend:

```
systemctl suspend
```

The computer enters a low-power state.

Resume normally occurs through keyboard, mouse, power button, or another configured wake source.

---

46. `systemctl hibernate`

On systems where hibernation is configured:

```
systemctl hibernate
```

The system saves memory state to persistent storage and powers down.

Not every Linux installation supports hibernation properly.

---

47. `systemctl hybrid-sleep`

Some systems support:

```
systemctl hybrid-sleep
```

This combines aspects of suspend and hibernation.

Support depends on hardware and system configuration.

48. Targets

Targets are another important systemd concept.

A target is roughly a way of grouping units and representing a particular system state.

Examples:

```
multi-user.target
graphical.target
rescue.target
emergency.target
```

You can view the default target:

```
systemctl get-default
```

For example:

```
graphical.target
```

---

49. Changing the default target

You can configure the default boot target.

For graphical systems:

```
sudo systemctl set-default graphical.target
```

For a text-oriented multi-user environment:

```
sudo systemctl set-default multi-user.target
```

This changes what target systemd normally reaches during boot.

---

50. Switching targets

You can manually start a target:

```
sudo systemctl isolate multi-user.target
```

or:

```
sudo systemctl isolate graphical.target
```

`isolate` is powerful because systemd may stop units that are not part of the requested target.

Therefore, use it carefully.

51. Rescue mode

You can enter rescue mode using:

```
sudo systemctl rescue
```

Rescue mode is useful for system maintenance and troubleshooting.

It provides a limited environment compared with normal operation.

---

52. Emergency mode

Another recovery state is:

```
sudo systemctl emergency
```

Emergency mode is more minimal than rescue mode.

It is intended for serious troubleshooting and recovery situations.

---

53. System state

You can view the current system state using:

```
systemctl is-system-running
```

Possible outputs include:

```
running
degraded
starting
stopping
offline
maintenance
initializing
```

For example:

```
running
```

generally indicates that systemd considers the system operational.

54. What does `degraded` mean?

Suppose:

```
systemctl is-system-running
```

returns:

```
degraded
```

This commonly means:

> The system is running, but one or more units have failed.

Check:

```
systemctl --failed
```

This is an excellent troubleshooting sequence.

---

55. `systemctl status`

Without specifying a unit:

```
systemctl status
```

This provides a summary of systemd's state.

It can show:

- system state
- number of loaded units
- active units
- failed units
- jobs
- recent system messages

56. Viewing boot information

`systemctl` can also provide boot-related information.
For example:

```
systemd-analyze
```

Although `systemd-analyze` is a separate command, it is part of the broader systemd ecosystem.

You can see boot timing:

```
systemd-analyze
```

Example:

```
Startup finished in 5.123s
```

---

57. Boot time analysis

To see which services consumed boot time:
`systemd-analyze blame`

This can produce output such as:

```
5.200s NetworkManager.service
3.100s apache2.service
1.500s systemd-udev-settle.service
```

This is useful for finding slow boot components.

---

58. `systemctl list-jobs`

systemd may have jobs currently being processed.

You can see them using:

```
systemctl list-jobs
```

A job might involve:

```
start
stop
restart
reload
```

59. Service logs

Although `journalctl` is the main log command, it works closely with systemd.

For example:

```
journalctl -u ssh
```

This displays logs associated with the SSH service.

For Apache:

```
journalctl -u apache2
```

For nginx:

```
journalctl -u nginx
```

---

60. Viewing recent service logs

Use:

```
journalctl -u ssh -n 50
```

This displays the last 50 log entries.

You can also use:

```
journalctl -u ssh -f
```

The `-f` option follows new log entries, similar to:

```
tail -f
```

---

61. Service troubleshooting workflow

Suppose Apache is not working.

First:

```
systemctl status apache2
```

If it says:

```
failed
```

check logs:

```
journalctl -u apache2
```

Then check configuration:

```
apache2ctl configtest
```

If the configuration is fixed:

```
sudo systemctl restart apache2
```

Then:

```
systemctl status apache2
```

This is a practical troubleshooting cycle.

---

62. Checking service configuration

For a service:

```
systemctl cat apache2
```

This helps you understand how systemd launches it.

You can also inspect:

```
systemctl show apache2
```

for detailed properties.

---

63. Finding unit files

You can use:

```
systemctl show -p FragmentPath ssh
```

This can show the primary unit-file location.

For example:

```
FragmentPath=/usr/lib/systemd/system/ssh.service
```

The actual location varies between distributions.

64. `systemctl list-unit-files`

This command is very useful:

```
systemctl list-unit-files
```

It shows unit files and their states.

For example:

```
UNIT FILE               STATE
ssh.service             enabled
apache2.service         enabled
cups.service            disabled
```

---

65. Static units

You may see:

```
static
```

in the STATE column.

A static unit does not have normal enablement metadata.

It is typically activated indirectly as a dependency or by another mechanism.

Therefore, this may not work meaningfully:

```
systemctl enable some-static-service
```

because static units are not normally enabled directly.

---

66. Generated units

You may also encounter:

```
generated
```

These are units generated dynamically by systemd tools or generators.

They can represent configuration discovered at boot or runtime.

67. Alias units

systemd supports aliases.

For example, multiple names may refer to the same underlying unit.

You may encounter relationships such as:

```
ssh.service
sshd.service
```

depending on the distribution.

Always check your system rather than assuming names are identical everywhere.

---

68. Service status with `--no-pager`

Sometimes `systemctl status` opens output through a pager.

You can avoid that:

```
systemctl --no-pager status ssh
```

This is especially useful in scripts or when copying output.

---

69. Avoiding colored output

For scripts or automated processing, you can also use options that control output formatting.

For example:

```
systemctl --no-pager status ssh
```

is generally easier to process than interactive pager output.

70. Quiet mode

Some commands support:

```
--quiet
```

For example:

```
systemctl is-active --quiet ssh
```

The useful information can then be obtained through the command's exit status.

This is particularly useful in shell scripts.

---

71. Using `systemctl` in shell scripts

For example:

```
if systemctl is-active --quiet nginx; then
    echo "Nginx is running"
else
    echo "Nginx is not running"
fi
```

This allows scripts to make decisions based on service state.

---

72. Checking whether a service is enabled in a script

Example:

```
if systemctl is-enabled --quiet ssh; then
    echo "SSH is enabled"
else
    echo "SSH is not enabled"
fi
```

This is useful for automated server administration.

73. Root privileges

Many `systemctl` operations require root privileges.

For example:

```
sudo systemctl restart apache2
```

But information-only commands can often be executed without `sudo`:

```
systemctl status apache2
```

The exact permission behavior depends on the operation and system configuration.

---

74. Why `sudo` is required

Starting, stopping, enabling, disabling, masking, or changing system services can affect the entire computer.

Therefore, ordinary users generally cannot perform these operations.

Example:

```
systemctl stop ssh
```

may return a permission-related error.

Use:

```
sudo systemctl stop ssh
```

when administrative privileges are required.

75. `systemctl` and processes

A service generally starts one or more processes.

For example:

```
systemd
   │
   └── ssh.service
           │
           └── sshd process
```

You can see the process ID in:

```
systemctl status ssh
```

You can also inspect processes using:

```
ps
```

or:

```
pgrep
```

---

76. Main PID

In a service status output, you might see:

```
Main PID: 1234
```

This identifies the primary process associated with the service.

You can investigate it:

```
ps -fp 1234
```

---

77. Service states

A service can have multiple state dimensions.

For example:

```
LoadState
ActiveState
SubState
```

You can see them with:

```
systemctl show ssh
```

Example:

```
LoadState=loaded
ActiveState=active
SubState=running
```

---

78. Load state

`LoadState` tells you whether systemd successfully loaded the unit definition.

Typical value:

```
loaded
```

Other possibilities include states indicating missing or invalid configuration.

---

79. Active state

`ActiveState` represents the broad activation state.
Common values include:

```
active
inactive
failed
activating
deactivating
```

---

80. Sub-state

`SubState` provides more detailed information.

For a running service:

```
SubState=running
```

For an inactive service:

```
SubState=dead
```

81. `systemctl reset-failed`

After a service fails, systemd records that failure state.

You can reset failure information:

```
sudo systemctl reset-failed
```

For one unit:

```
sudo systemctl reset-failed apache2
```

This resets the recorded failed state.

It does **not automatically fix the underlying problem**.

---

82. Restarting a failed service

A common workflow is:

```
systemctl status apache2
```

then:

```
journalctl -u apache2
```

Fix the problem.

Then:

```
sudo systemctl restart apache2
```

Then:

```
systemctl status apache2
```

If needed:

```
sudo systemctl reset-failed apache2
```

83. `try-restart`

Some systemd versions support:

```
sudo systemctl try-restart SERVICE
```

This restarts the service only if it is already running.

Conceptually:

```
Running?
   │
   ├── Yes → restart
   │
   └── No  → don't start
```

This differs from:

```
systemctl restart
```

which can start an inactive service.

---

84. `condrestart`

You may encounter compatibility behavior around:

```
condrestart
```

It is related to conditional restarting and is often associated with compatibility interfaces.

For modern systemd administration, learning:

```
try-restart
```

is generally more useful.

85. `start` vs `enable`

This is one of the most important distinctions.

Start

```
sudo systemctl start nginx
```

Means:

> Start nginx now.

Enable

```
sudo systemctl enable nginx
```

Means:

> Configure nginx to start automatically during the appropriate boot sequence.

They are independent concepts.

---

86. Example

Suppose:

```
systemctl is-active nginx
```

returns:

```
inactive
```

but:

```
systemctl is-enabled nginx
```

returns:

```
enabled
```

This means:

```
Currently:
    nginx = stopped

At boot:
    nginx = configured to start
```

You can start it immediately:

```
sudo systemctl start nginx
```

---

87. `enable --now`

Instead of:

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

you can use:

```
sudo systemctl enable --now nginx
```

This is one of the most useful shortcuts to remember.

---

88. `disable --now`

Similarly:

```
sudo systemctl disable --now nginx
```

means:

```
Disable automatic startup
+
Stop it now
```

89. Service dependencies

systemd supports relationships such as:

```
Requires=
Wants=
After=
Before=
Conflicts=
PartOf=
```

These are normally specified in unit files.

For example:

```
After=network-online.target
```

means systemd should order activation after that target.

---

90. `After=` does not necessarily mean dependency

This is a subtle but important point.

Suppose a unit has:

```
After=network.target
```

This primarily defines **ordering**.

It does not necessarily mean:

> Start network.target automatically because this service needs it.

Dependencies and ordering are related but different concepts.

This distinction is important for advanced systemd administration.

91. `Requires=` vs `Wants=`

A unit can express relationships such as:

```
Requires=foo.service
```

or:

```
Wants=foo.service
```

`Requires=` represents a stronger dependency relationship.

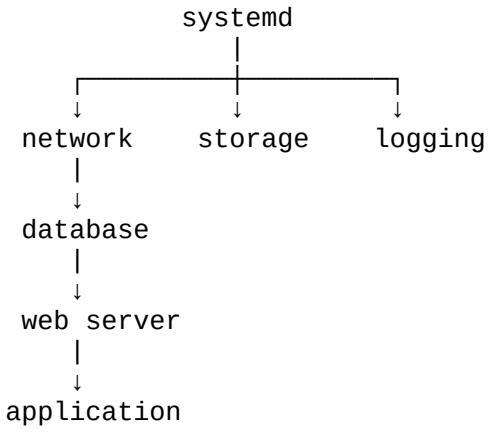
`Wants=` is generally weaker.

This allows systemd to construct a dependency graph.

---

92. Unit dependency graph

Think of systemd as managing a graph:



systemd determines how units are activated and ordered based on their relationships.

93. Timers

systemd can also schedule tasks using timer units.

List timers:

```
systemctl list-timers
```

This can show:

- last execution
- next execution
- time remaining
- timer unit
- associated service

---

94. Why systemd timers are useful

Traditionally, Linux administrators often use:

```
cron
```

for scheduled jobs.

systemd provides:

```
.timer
```

units.

Advantages can include:

- integration with systemd
- dependency handling
- centralized status
- journal logging
- service management

95. Example timer concept

Suppose you have:

```
backup.service
backup.timer
```

The timer can activate:

```
backup.service
```

at a scheduled time.

You can check:

```
systemctl status backup.timer
```

and:

```
systemctl list-timers
```

---

96. Socket units

systemd can also manage sockets.

For example:

```
something.socket
```

A socket can listen for connections and activate an associated service when required.

This is called:

socket activation

It can allow services to start on demand rather than remaining active all the time.

97. Mount units

systemd can manage filesystem mounts.

For example:

`home.mount`

You can inspect mount units:

```
systemctl list-units --type=mount
```

This integrates filesystem mounting into systemd's dependency model.

---

98. Device units

systemd can represent devices as units.

For example, hardware devices exposed through the kernel can be represented using `.device` units.

You can inspect them:

```
systemctl list-units --type=device
```

---

99. Path units

A `.path` unit can monitor a filesystem path.

For example:

`backup.path`

could trigger:

`backup.service`

when a particular filesystem event occurs.

List path units:

```
systemctl list-units --type=path
```

100. Slice units

Slices are related to resource management.

For example:

```
system.slice
user.slice
```

They can be used to organize processes and apply resource controls.

You can view them:

```
systemctl list-units --type=slice
```

---

101. Scope units

Scope units represent groups of externally created processes.

For example, user sessions and desktop environments can involve scopes.

List them:

```
systemctl list-units --type=scope
```

---

102. User-level systemd

systemd can also operate in the context of individual users.

For example:

```
systemctl --user status
```

This communicates with the current user's systemd manager rather than the system-wide manager.

This is different from:

```
sudo systemctl status
```

103. System-wide vs user services

System-wide:

```
sudo systemctl start example.service
```

User-level:

```
systemctl --user start example.service
```

Think of it as:

```
systemctl
    ↓
system-wide systemd
```

```
systemctl --user
    ↓
user's systemd instance
```

---

104. `systemctl --user list-units`

You can list user units:

```
systemctl --user list-units
```

This is useful for desktop applications and user-specific background services.

---

105. Boot target and graphical.target

On a desktop Linux system, you may commonly see:

```
graphical.target
```

This represents a system state in which graphical login and related services are available.

You can check:

```
systemctl get-default
```

106. multi-user.target

Another important target:

`multi-user.target`

It generally represents a fully operational non-graphical multi-user environment.

Servers commonly use this target.

---

107. rescue.target

`rescue.target`

represents a recovery/maintenance environment.

It provides a more limited system state.

---

108. emergency.target

`emergency.target`

is an even more minimal recovery state.

It is useful when normal boot cannot proceed.

---

109. Boot process overview

A simplified modern Linux boot sequence:

```
BIOS/UEFI
    ↓
Bootloader
    ↓
Linux Kernel
    ↓
systemd (PID 1)
    ↓
early system initialization
    ↓
local filesystems
    ↓
services
```

```
    ↓
network
    ↓
login/graphical environment
```

`systemd` normally runs as:

```
PID 1
```

---

110. Why PID 1 is special

You can check:

```
ps -p 1 -f
```

On a typical systemd system, you may see:

```
/usr/lib/systemd/systemd
```

or a similar path.

PID 1 has special responsibilities, including managing the initial user-space process hierarchy and reaping orphaned processes.

---

111. What happens when a service crashes?

Suppose:

```
nginx.service
```

is running.

The nginx process crashes.

systemd can detect the service state.

Depending on its configuration, it may:

```
Detect failure
      ↓
Record failure
      ↓
Restart service
```

if the unit has a suitable restart policy.

For example:

```
Restart=on-failure
```

may be configured.

---

112. Restart policies

Service unit files can contain settings such as:

```
Restart=always
```

or:

```
Restart=on-failure
```

These control what systemd does when the service exits.

This is a major reason systemd is useful for server applications.

---

113. Viewing restart configuration

You can inspect:

```
systemctl show nginx -p Restart
```

Possible output:

```
Restart=on-failure
```

The exact value depends on the service.

114. Service startup timeout

Services can have timeout settings.

For example:

```
TimeoutStartSec=
```

You can inspect properties with:

```
systemctl show SERVICE
```

This can help when troubleshooting services that appear stuck during startup.

---

115. Environment variables

systemd services can have their own environment settings.

You can inspect them with:

```
systemctl show SERVICE
```

or inspect the actual unit configuration using:

```
systemctl cat SERVICE
```

---

116. Security features

systemd service units can include security restrictions.

Examples include:

```
User=
Group=
NoNewPrivileges=
PrivateTmp=
ProtectSystem=
ProtectHome=
```

These can restrict what a service can access.

This is important for server security.

117. Running a service as another user

A service can be configured with:

```
User=someuser
```

Then systemd starts the service under that user instead of root.

This reduces privileges and can improve security.

---

118. Viewing service resource information

`systemctl status` may show:

```
Tasks
Memory
CPU
```

You can obtain detailed properties with:

```
systemctl show SERVICE
```

Resource controls are closely integrated with systemd and Linux control groups.

---

119. cgroups

systemd uses Linux **control groups**, commonly called cgroups.

They allow processes to be grouped and controlled.

Conceptually:

```
systemd
   │
   ├── service A
   │       ├── process 1
   │       └── process 2
   │
   └── service B
           ├── process 3
           └── process 4
```

This makes resource management and process tracking more reliable.

120. Why `systemctl` is so important

The command is not simply a replacement for:

```
service
```

It provides access to a much larger management system.

It can manage:

```
Services
Targets
Timers
Sockets
Mounts
Devices
Paths
Slices
Scopes
User units
```

121. Common commands — quick reference

Command	Purpose
`systemctl status`	Show system manager status
`systemctl status ssh`	Show SSH status
`systemctl start ssh`	Start SSH
`systemctl stop ssh`	Stop SSH
`systemctl restart ssh`	Restart SSH
`systemctl reload ssh`	Reload SSH configuration
`systemctl enable ssh`	Enable SSH at boot
`systemctl disable ssh`	Disable SSH at boot
`systemctl enable --now ssh`	Enable and start
`systemctl disable --now ssh`	Disable and stop
`systemctl is-active ssh`	Check active state
`systemctl is-enabled ssh`	Check enablement
`systemctl is-failed ssh`	Check failure
`systemctl --failed`	Show failed units
`systemctl list-units`	List loaded units
`systemctl list-unit-files`	List unit files
`systemctl list-timers`	List timers
`systemctl list-dependencies ssh`	Show dependencies
`systemctl cat ssh`	Show unit configuration
`systemctl show ssh`	Show properties
`systemctl edit ssh`	Create override
`systemctl daemon-reload`	Reload unit definitions
`systemctl mask ssh`	Prevent activation
`systemctl unmask ssh`	Remove mask
`systemctl reboot`	Reboot
`systemctl poweroff`	Power off
`systemctl suspend`	Suspend
`systemctl hibernate`	Hibernate

122. Most important commands to memorize

If you are preparing for an exam, memorize these first:

```
systemctl status SERVICE

sudo systemctl start SERVICE

sudo systemctl stop SERVICE

sudo systemctl restart SERVICE

sudo systemctl reload SERVICE

sudo systemctl enable SERVICE

sudo systemctl disable SERVICE

sudo systemctl enable --now SERVICE

systemctl is-active SERVICE

systemctl is-enabled SERVICE

systemctl --failed

systemctl list-units --type=service

systemctl list-unit-files --type=service

journalctl -u SERVICE
```

---

123. Practical example - SSH

Check SSH:

```
systemctl status ssh
```

Start it:

```
sudo systemctl start ssh
```

Stop it:

```
sudo systemctl stop ssh
```

Restart it:

```
sudo systemctl restart ssh
```

Enable at boot:

```
sudo systemctl enable ssh
```

Enable and start:

```
sudo systemctl enable --now ssh
```

Check:

```
systemctl is-active ssh
```

Check boot enablement:

```
systemctl is-enabled ssh
```

View logs:

```
journalctl -u ssh
```

---

124. Practical example - Apache

Check:

```
systemctl status apache2
```

Start:

```
sudo systemctl start apache2
```

Stop:

```
sudo systemctl stop apache2
```

Restart:

```
sudo systemctl restart apache2
```

Enable:

```
sudo systemctl enable apache2
```

Disable:

```
sudo systemctl disable apache2
```

Logs:

```
journalctl -u apache2
```

---

125. Practical example - NetworkManager

On systems using NetworkManager:

```
systemctl status NetworkManager
```

Restarting it:

```
sudo systemctl restart NetworkManager
```

can restart network management.

Be careful: if you are connected remotely over SSH, restarting networking can disconnect your session.

---

126. Practical example - failed service

Imagine:

```
systemctl status myservice
```

shows:

```
Active: failed
```

Do not immediately keep restarting it.

Instead:

Step 1

```
systemctl status myservice
```

Step 2

```
journalctl -u myservice
```

Step 3

Check configuration.

Step 4

Fix the problem.

Step 5

```
sudo systemctl restart myservice
```

Step 6

```
systemctl status myservice
```

---

127. Practical example - service won't start

Suppose:

```
sudo systemctl start nginx
```

returns an error.

Run:

```
systemctl status nginx
```

Then:

```
journalctl -u nginx -b
```

The `-b` option restricts the journal to the current boot.

This is extremely useful.

128. Checking current boot logs

For systemd-related troubleshooting:

```
journalctl -b
```

This shows messages from the current boot.

Previous boot:

```
journalctl -b -1
```

assuming previous boot logs are available.

---

129. Combining `systemctl` and `journalctl`

A powerful troubleshooting pair is:

```
systemctl status SERVICE
```

and:

```
journalctl -u SERVICE
```

Think:

```
systemctl
   ↓
"What is the service doing?"
```

```
journalctl
   ↓
"What happened to the service?"
```

---

130. `systemctl` vs `service`

Older Linux systems commonly used:

```
service apache2 restart
```

Modern systemd systems commonly use:

```
sudo systemctl restart apache2
```

You may still find the `service` command for compatibility.

But when working directly with systemd:

```
systemctl
```

is the preferred interface.

---

131. `systemctl` vs `/etc/init.d/`

Older systems commonly used scripts under:

```
/etc/init.d/
```

systemd uses unit files instead.

For example:

```
apache2.service
```

may be represented by a systemd unit.

The modern architecture provides dependency management, parallel startup, resource control, logging integration, and other features.

---

132. Why systemd can start services in parallel

Traditional startup systems often followed a more sequential model.

systemd builds a dependency and ordering graph.

If two services do not need to wait for each other:

```
Service A ————→ starts
Service B ————→ starts
```

they can potentially start concurrently.

This can improve boot performance.

133. Important caution about `systemctl`

`systemctl` is powerful.

Commands like:

```
sudo systemctl stop NetworkManager
```

or:

```
sudo systemctl isolate rescue.target
```

can significantly change system behavior.

Always understand the service before stopping or disabling it.

---

134. Common mistakes

Mistake 1

Using:

```
systemctl enable SERVICE
```

and expecting it to start immediately.

Correction:

```
systemctl enable --now SERVICE
```

Mistake 2

Using:

```
systemctl start SERVICE
```

and expecting it to start after reboot.

Correction:

```
systemctl enable SERVICE
```

Mistake 3

Seeing:

```
inactive
```

and assuming the service is broken.

Some services are designed to be inactive until needed.

Mistake 4

Using `restart` when only a configuration reload is needed.

If supported:

```
systemctl reload SERVICE
```

may cause less interruption.

Mistake 5

Editing packaged unit files directly.

Prefer:

```
systemctl edit SERVICE
```

for local overrides.

135. Exam questions and answers

Q1. What is `systemctl`?

Answer:

`systemctl` is the command-line utility used to control and inspect the systemd system and service manager.

Q2. What is systemd?

Answer:

systemd is a system and service manager that manages system initialization, services, dependencies, targets, timers, sockets, mounts, and other units.

Q3. How do you start a service?

```
sudo systemctl start SERVICE
```

Q4. How do you stop a service?

```
sudo systemctl stop SERVICE
```

Q5. How do you restart a service?

```
sudo systemctl restart SERVICE
```

Q6. How do you enable a service at boot?

```
sudo systemctl enable SERVICE
```

Q7. How do you enable and immediately start a service?

```
sudo systemctl enable --now SERVICE
```

Q8. How do you check service status?

```
systemctl status SERVICE
```

Q9. How do you list failed units?

```
systemctl --failed
```

Q10. How do you see service logs?

```
journalctl -u SERVICE
```

Q11. Difference between `start` and `enable`?

`start` activates a service now.

`enable` configures it to start automatically during boot.

Q12. Difference between `disable` and `mask`?

`disable` removes normal boot enablement.

`mask` prevents normal activation of the unit.

Q13. What does `daemon-reload` do?

```
sudo systemctl daemon-reload
```

It causes systemd to reread unit files.

Q14. What does `systemctl --failed` do?

It displays units that systemd considers failed.

Q15. What is a target?

A target is a systemd unit used to group units and represent a particular system state.

Examples:

```
multi-user.target
graphical.target
rescue.target
```

---

136. Easy way to remember the main commands

Use this memory pattern:

```
START
STOP
RESTART
RELOAD
ENABLE
DISABLE
STATUS
```

Commands:

```
systemctl start SERVICE
systemctl stop SERVICE
systemctl restart SERVICE
systemctl reload SERVICE
systemctl enable SERVICE
systemctl disable SERVICE
systemctl status SERVICE
```

And remember:

```
START   = NOW
ENABLE  = BOOT
```

This one distinction solves a large number of beginner questions.

137. Complete practical workflow

When working with any service, use this sequence:

```
systemctl status SERVICE
```

If stopped:

```
sudo systemctl start SERVICE
```

Check:

```
systemctl status SERVICE
```

If you want automatic startup:

```
sudo systemctl enable SERVICE
```

If configuration changes:

```
sudo systemctl reload SERVICE
```

If reload isn't supported:

```
sudo systemctl restart SERVICE
```

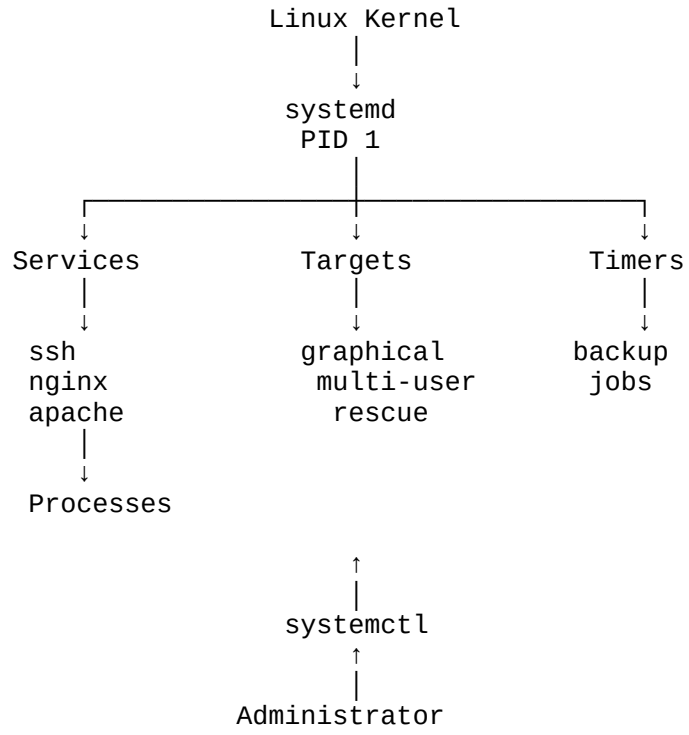
If something fails:

```
journalctl -u SERVICE
```

This workflow is worth memorizing.

138. Final mental model

Think of Linux as:



So when you type:

```
sudo systemctl restart nginx
```

you are essentially saying:

> **"Ask the systemd manager to restart the nginx service."**

---

139. The 15 commands you should remember

```
systemctl status SERVICE

sudo systemctl start SERVICE

sudo systemctl stop SERVICE

sudo systemctl restart SERVICE

sudo systemctl reload SERVICE
```

```
sudo systemctl enable SERVICE

sudo systemctl disable SERVICE

sudo systemctl enable --now SERVICE

sudo systemctl disable --now SERVICE

systemctl is-active SERVICE

systemctl is-enabled SERVICE

systemctl --failed

systemctl list-units --type=service

systemctl list-unit-files --type=service

journalctl -u SERVICE
```

---

140. One-page revision

`systemctl` = systemd control utility

Service management

```
systemctl status ssh
sudo systemctl start ssh
sudo systemctl stop ssh
sudo systemctl restart ssh
sudo systemctl reload ssh
```

Boot management

```
sudo systemctl enable ssh
sudo systemctl disable ssh
sudo systemctl enable --now ssh
sudo systemctl disable --now ssh
```

Checking

```
systemctl is-active ssh
systemctl is-enabled ssh
systemctl is-failed ssh
systemctl --failed
```

Listing

```
systemctl list-units
systemctl list-units --type=service
systemctl list-unit-files
systemctl list-timers
```

Configuration

```
systemctl cat ssh
systemctl show ssh
sudo systemctl edit ssh
sudo systemctl daemon-reload
```

Dependencies

```
systemctl list-dependencies ssh
```

Logs

```
journalctl -u ssh
journalctl -u ssh -n 50
journalctl -u ssh -f
journalctl -b
```

System control

```
sudo systemctl reboot
sudo systemctl poweroff
systemctl suspend
systemctl hibernate
```

Recovery

```
sudo systemctl rescue
sudo systemctl emergency
```

Targets

```
systemctl get-default
sudo systemctl set-default graphical.target
sudo systemctl set-default multi-user.target
```

Final definition for an exam

> **`systemctl`** is a command-line utility used to control and inspect the systemd system and service manager in Linux. It allows administrators to start, stop, restart, reload, enable, disable, mask, unmask, monitor, and troubleshoot services and other systemd units. It can also manage targets, timers, sockets, mounts, and other units, and provides commands for controlling system states such as reboot, shutdown, suspend, rescue, and emergency modes.

If you remember only one thing:

```
systemctl = control systemd
```

and:

```
start   → start now
stop    → stop now
restart → stop + start
reload  → reread configuration
enable  → start automatically at boot
disable → don't start automatically at boot
status  → see what's happening
```

That is the **core of `systemctl`**.