Chapter 1

Managing and Understanding the Boot Procedure

1.1 Boot Procedure Generic Overview

On starting up, the computer perform a "Power On Self Test" (POST). During this, the computer checks all the connected hardware and finds the boot device, which is typically a hard disk / solid state drive (HDD/SSD). On the boot device, the computer access Grub 2, the boot loader, that loads the **kernel** and **initrd**.

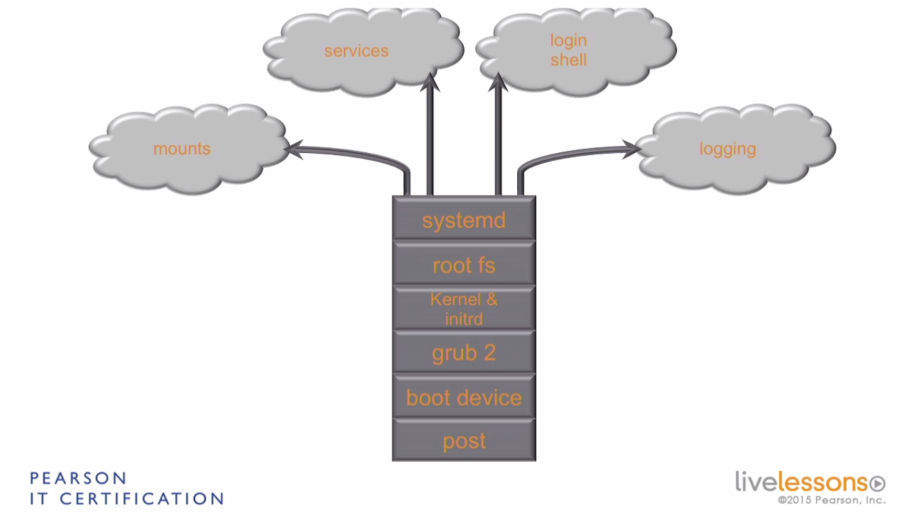

Figure 1.1: Boot Procedure

Next, the root file system is mounted (by the kernel) and then, **systemd** is started. Once systemd has started, everything else can begin, such as: logging, mounting the other file systems, starting all services and preparing the login shell.

1.2 Understanding Grub2

The very first thing from the linux perspective (i.e., the first thing that's executed) when a computer boots is Grub2 (Grand Unified Boot-loader).

The /etc/default/grub is the most important configuration file for Grub2. Most of the customizations/modifications by an user is done to this file. There are also additional configuration files in the /etc/grub.d directory. If any of the configuration files have been updated, the boot loader needs to be updated as well, by using the grub2-mkconfig command. This updates the data in the Master Boot Record (MBR) and the metadata in the first few sectors of our hard drives.

Once the computer boots, we can access the Grub boot menu by pressing the *escape* key. When this is done, we can enter special boot instructions on it.

1.2.1 Booting in emergency mode

On the boot menu, we need to enter systemd.unit=emergency.target as a boot option to start up the computer in emergency mode, which is used in case the computer can't boot normally.

The diagram below explains the entire boot procedure. Once the power is supplied to the computer, it performs the Power On Self Test and then loads the boot loader from the MBR. Now, we have the option to enter the boot menu by pressing the escape key, and enter the boot options, like booting in emergency mode.

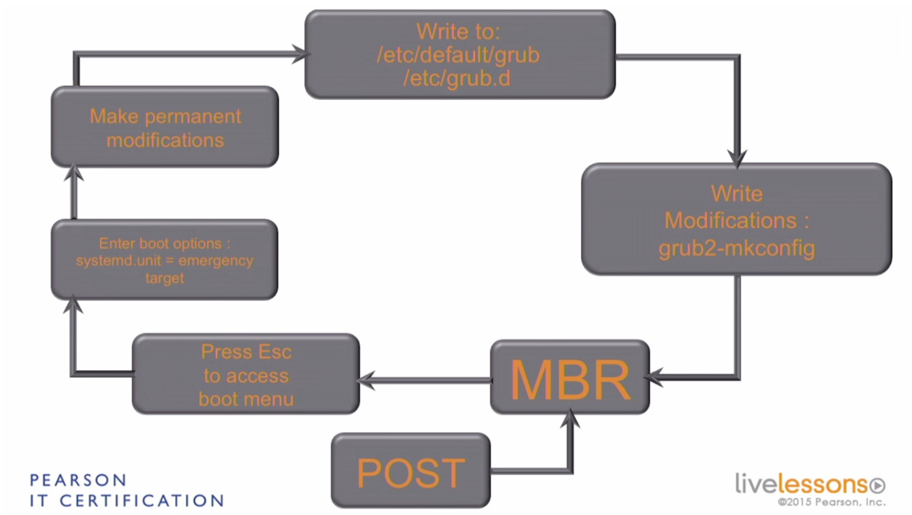

Figure 1.2: Booting in emergency mode

In case there's something wrong with the bootloader itself, we can make permanent modifications by editing the files: /etc/default/grub and the config files in /etc/grub.d directory. Once these modifications have been written, the boot loader needs to be updated using grub2-mkconfig command. This ensures that the next time the MBR will be read, the edited grub2 configuration files will be used.

1.3 Modifying Grub2 Parameters

The primary grub configuration file is /etc/default/grub. The default contents of it look like:

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
```

```
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb
↪  quiet"
GRUB_DISABLE_RECOVERY="true"
```

The `GRUB_TIMEOUT` parameter defines the amount of time the system waits in the Grub boot menu for user input. The most important parameter is `GRUB_CMDLINE_LINUX` which defines which arguments are passed on to the linux kernel when the system is booting. The last portion of this parameter, `rhgb quiet` stops Grub from showing us what it's doing during boot. To enable this feature, we need to delete those arguments.

Next, we take a look at the `/etc/grub.d` directory. The files in here are shell scripts that aren't meant to be changed normally, and help with the boot process.

After making all changes, we need to execute the `grub2-mkconfig` command to update the changes to the Grub2 boot-loader. The command reads (and compiles) every grub related config file. This generates a grub configuration file (to send to the boot loader).

To see all these changes, we need to reboot our computer. To verify that the changes have been applied correctly, we can enter the Grub menu using the *Escape* key. In there, we can find the kernel line with all the options that are used. A *CTRL+X* at this point causes a reboot with the new parameters passed to the kernel.

1.4 Understanding Systemd

Systemd is a major new feature added in RHEL 7. It is a new init system that starts things - it both bootstraps the current user-space as well as manage the system processes after booting.

During startup, right after the loading of the kernel, systemd is started, and systemd in turn takes care of starting everything else. Unlike the older *runlevel* system, where only services were started, systemd takes care of services, mounting partitions, auto mounting file systems, and much more.

1.4.1 Unit file

A **unit** file is the replacement of the old init script. Init scripts were relatively more difficult to understand. The unit files have greater readability.

This unit file defines how to start services and everything else systemd can do, as well as define the relation between all those things. Systemd has two different locations for the storing of scripts - the default scripts are stored in `/usr/lib/systemd` and the administrator's custom scripts reside in the `/etc/systemd` directory.

1.5 Managing Services in a systemd Environment

To get a task done in Linux, we need services, which are started using systemd. The directory `/usr/lib/systemd/system` contains many service scripts (among other files). This directory is for the default services that are installed by the RPMs. Thus, we shouldn't edit the files in this directory.

For our own system service management needs, we go to the `/etc/systemd/system` folder. This has two advantages: i) updates to the RPMs that dropped the service scripts in `/usr/lib/systemd/system` won't overwrite our scripts, and ii) Our scripts in `/etc/systemd/system` will overwrite those in the other folder.

1.5.1 Service files

The services form the basic unit of management in systemd is a service. The service files contain all the information required to start a service.

Let us consider the `/usr/lib/systemd/system/httpd.service` file:

```
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/httpd
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/bin/kill -WINCH ${MAINPID}
# We want systemd to give httpd some time to finish gracefully, but still want
# it to kill httpd after TimeoutStopSec if something went wrong during the
# graceful stop. Normally, Systemd sends SIGTERM signal right after the
# ExecStop, which would kill httpd. We are sending useless SIGCONT here to give
# httpd time to finish.
KillSignal=SIGCONT
PrivateTmp=true

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

Due to the usage of systemd, a service in RHEL 7 is much more powerful than a service in previous versions. It is possible to basically turn everything into a service and control it using systemd.

The `[Install]` section of the file defines how the service should be started. The `WantedBy` parameter is defined to set this. Here, the service must be started by a *target*. Services are assigned to targets and the targets take care of starting up the services.

Next, we take a look at the service definition. In earlier versions of RHEL, this section was implemented by the help of large shell scripts. Now, only a few lines of configuration settings are needed. This section defines what should be started and how.

1.5.2 systemctl

Services are managed using the `systemctl` command. For example, to see the status of a service, we write:

```
# systemctl status httpd -l
● httpd.service - The Apache HTTP Server
Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
```

```
Active: active (running) since Sat 2017-12-16 09:31:03 IST; 3s ago
Docs: man:httpd(8)
man:apachectl(8)
Main PID: 5831 (httpd)
Status: "Processing requests..."
CGroup: /system.slice/httpd.service
├─5831 /usr/sbin/httpd -DFOREGROUND
├─5840 /usr/sbin/httpd -DFOREGROUND
├─5842 /usr/sbin/httpd -DFOREGROUND
├─5843 /usr/sbin/httpd -DFOREGROUND
├─5844 /usr/sbin/httpd -DFOREGROUND
└─5845 /usr/sbin/httpd -DFOREGROUND

Dec 16 09:31:01 vmPrime.somuVMnet.local systemd[1]: Starting The Apache HTTP Server...
Dec 16 09:31:03 vmPrime.somuVMnet.local systemd[1]: Started The Apache HTTP Server.
```

To start a service we use:

```
# systemctl start httpd
```

To stop it we use:

```
# systemctl stop httpd
```

To permanently remove the service from the startup procedure of our OS, we use:

```
# systemctl disable httpd
Removed symlink /etc/systemd/system/multi-user.target.wants/httpd.service.
```

To enable the service again:

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

1.5.3 Targets

Our systems can enter different states called **targets**, which are also defined in `/usr/lib/systemd/system` and `/etc/systemd/system`. Targets act as a collection of services, and we can specify dependency relations within the target file. Two of the most important targets are: `multi-user.target` and `graphical.target`, both present in `/usr/lib/systemd/system` directory.

The *graphical.target* is started as the default environment when a GUI is running. Contrastingly, the *multi-user.target* is used as a default environment on servers where a GUI isn't present.

1.5.4 Wants

In order to put services in a specific target, we create a *wants* directory for that target and put a symbolic link to that service in that directory. Services belong to a specific target. When a service is enabled, a symbolic link is created in some *Wants* directory. Each target has its own wants directory that ends with the name of the target followed by a `.wants`. For example, the `multi-user.target` has a corresponding directory called `multi-user.target.wants` in the same folder.

These directories only contain symbolic links to services that should be available at all times in that particular target. For example, the `multi-user.target.wants` contains:

```
# ls -l /usr/lib/systemd/system/multi-user.target.wants/
total 0
lrwxrwxrwx. 1 root root 16 Nov 25 08:50 brandbot.path -> ../brandbot.path
lrwxrwxrwx. 1 root root 15 Nov 25 08:50 dbus.service -> ../dbus.service
lrwxrwxrwx. 1 root root 15 Nov 25 10:14 getty.target -> ../getty.target
lrwxrwxrwx. 1 root root 24 Nov 25 08:50 plymouth-quit.service -> ../plymouth-quit.service
lrwxrwxrwx. 1 root root 29 Nov 25 08:50 plymouth-quit-wait.service ->
  ↪  ../plymouth-quit-wait.service
lrwxrwxrwx. 1 root root 33 Nov 25 10:14 systemd-ask-password-wall.path ->
  ↪  ../systemd-ask-password-wall.path
lrwxrwxrwx. 1 root root 25 Nov 25 10:14 systemd-logind.service ->
  ↪  ../systemd-logind.service
lrwxrwxrwx. 1 root root 39 Nov 25 10:14 systemd-update-utmp-runlevel.service ->
  ↪  ../systemd-update-utmp-runlevel.service
lrwxrwxrwx. 1 root root 32 Nov 25 10:14 systemd-user-sessions.service ->
  ↪  ../systemd-user-sessions.service
```

Further, there are also the services resident in `/etc/systemd/system/multi-user.target.wants` which will also be included:

```
# ls -l /etc/systemd/system/multi-user.target.wants/
total 0
lrwxrwxrwx. 1 root root 41 Nov 25 08:51 abrt-ccpp.service ->
  ↪  /usr/lib/systemd/system/abrt-ccpp.service
lrwxrwxrwx. 1 root root 37 Nov 25 08:50 abrtd.service ->
  ↪  /usr/lib/systemd/system/abrtd.service
lrwxrwxrwx. 1 root root 41 Nov 25 08:50 abrt-oops.service ->
  ↪  /usr/lib/systemd/system/abrt-oops.service
lrwxrwxrwx. 1 root root 43 Nov 25 08:51 abrt-vmcore.service ->
  ↪  /usr/lib/systemd/system/abrt-vmcore.service
lrwxrwxrwx. 1 root root 41 Nov 25 08:50 abrt-xorg.service ->
  ↪  /usr/lib/systemd/system/abrt-xorg.service
lrwxrwxrwx. 1 root root 35 Nov 25 08:59 atd.service ->
  ↪  /usr/lib/systemd/system/atd.service
lrwxrwxrwx. 1 root root 38 Nov 25 08:51 auditd.service ->
  ↪  /usr/lib/systemd/system/auditd.service
lrwxrwxrwx. 1 root root 44 Nov 25 08:59 avahi-daemon.service ->
  ↪  /usr/lib/systemd/system/avahi-daemon.service
lrwxrwxrwx. 1 root root 39 Nov 25 08:51 chronyd.service ->
  ↪  /usr/lib/systemd/system/chronyd.service
lrwxrwxrwx. 1 root root 37 Nov 25 08:50 crond.service ->
  ↪  /usr/lib/systemd/system/crond.service
lrwxrwxrwx. 1 root root 33 Nov 25 08:55 cups.path -> /usr/lib/systemd/system/cups.path
lrwxrwxrwx. 1 root root 36 Nov 25 08:55 cups.service ->
  ↪  /usr/lib/systemd/system/cups.service
lrwxrwxrwx. 1 root root 41 Nov 25 08:51 firewalld.service ->
  ↪  /usr/lib/systemd/system/firewalld.service
lrwxrwxrwx. 1 root root 37 Dec 16 11:32 httpd.service ->
  ↪  /usr/lib/systemd/system/httpd.service
lrwxrwxrwx. 1 root root 42 Nov 25 08:59 irqbalance.service ->
  ↪  /usr/lib/systemd/system/irqbalance.service
lrwxrwxrwx. 1 root root 37 Nov 25 08:51 kdump.service ->
  ↪  /usr/lib/systemd/system/kdump.service
```

```
lrwxrwxrwx. 1 root root 35 Nov 25 08:51 ksm.service ->
 ↪ /usr/lib/systemd/system/ksm.service
lrwxrwxrwx. 1 root root 40 Nov 25 08:51 ksmtuned.service ->
 ↪ /usr/lib/systemd/system/ksmtuned.service
lrwxrwxrwx. 1 root root 46 Nov 25 08:50 libstoragemgmt.service ->
 ↪ /usr/lib/systemd/system/libstoragemgmt.service
lrwxrwxrwx. 1 root root 40 Nov 25 08:52 libvirtd.service ->
 ↪ /usr/lib/systemd/system/libvirtd.service
lrwxrwxrwx. 1 root root 38 Nov 25 08:59 mcelog.service ->
 ↪ /usr/lib/systemd/system/mcelog.service
lrwxrwxrwx. 1 root root 41 Nov 25 08:51 mdmonitor.service ->
 ↪ /usr/lib/systemd/system/mdmonitor.service
lrwxrwxrwx. 1 root root 44 Nov 25 08:59 ModemManager.service ->
 ↪ /usr/lib/systemd/system/ModemManager.service
lrwxrwxrwx. 1 root root 46 Nov 25 08:50 NetworkManager.service ->
 ↪ /usr/lib/systemd/system/NetworkManager.service
lrwxrwxrwx. 1 root root 41 Nov 25 08:52 nfs-client.target ->
 ↪ /usr/lib/systemd/system/nfs-client.target
lrwxrwxrwx. 1 root root 39 Nov 25 08:59 postfix.service ->
 ↪ /usr/lib/systemd/system/postfix.service
lrwxrwxrwx. 1 root root 40 Nov 25 08:50 remote-fs.target ->
 ↪ /usr/lib/systemd/system/remote-fs.target
lrwxrwxrwx. 1 root root 36 Nov 25 08:59 rngd.service ->
 ↪ /usr/lib/systemd/system/rngd.service
lrwxrwxrwx. 1 root root 39 Nov 25 08:59 rsyslog.service ->
 ↪ /usr/lib/systemd/system/rsyslog.service
lrwxrwxrwx. 1 root root 38 Nov 25 08:59 smartd.service ->
 ↪ /usr/lib/systemd/system/smartd.service
lrwxrwxrwx. 1 root root 36 Nov 25 08:59 sshd.service ->
 ↪ /usr/lib/systemd/system/sshd.service
lrwxrwxrwx. 1 root root 39 Nov 25 08:59 sysstat.service ->
 ↪ /usr/lib/systemd/system/sysstat.service
lrwxrwxrwx. 1 root root 37 Nov 25 08:59 tuned.service ->
 ↪ /usr/lib/systemd/system/tuned.service
lrwxrwxrwx. 1 root root 40 Nov 25 08:51 vmtoolsd.service ->
 ↪ /usr/lib/systemd/system/vmtoolsd.service
```

Now, the `/etc/systemd/system/default.target` defines which target (graphical or multi-user) is set as the default environment post-boot for the users.

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Nov 25 09:08 /etc/systemd/system/default.target ->
 ↪ /lib/systemd/system/graphical.target
```

Above, we can see that the graphical.target is set as the default. If we want to change that, we just change the link to point to `/lib/systemd/system/multi-user.target` to operate in a CLI by default.

1.5.5 Viewing Currently Loaded Targets

To view the currently loaded targets we use:

```
# systemctl list-units --type=target
UNIT                   LOAD   ACTIVE SUB    DESCRIPTION
basic.target           loaded active active Basic System
bluetooth.target       loaded active active Bluetooth
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-online.target  loaded active active Network is Online
network-pre.target     loaded active active Network (Pre)
network.target         loaded active active Network
nfs-client.target      loaded active active NFS client services
nss-user-lookup.target loaded active active User and Group Name Lookups
paths.target           loaded active active Paths
remote-fs-pre.target   loaded active active Remote File Systems (Pre)
remote-fs.target       loaded active active Remote File Systems
slices.target          loaded active active Slices
sockets.target         loaded active active Sockets
sound.target           loaded active active Sound Card
swap.target            loaded active active Swap
sysinit.target         loaded active active System Initialization
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.

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

The services provided by our entire OS are not packed together into one monolithic target, but broken down into several targets that concurrently active, as can be seen above. How these targets are supposed to work together is also defined in the target files. For example, in the `/usr/lib/systemd/system/multi-user.target` file:

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

In this, we can see that the `multi-user.target` requires the *basic.target* to be loaded. It conflicts with *rescue.target* and it has to be loaded only after the *basic.target* is loaded.

Thus, when systemd will try to load the *multi-user.target*, it'll first check the dependencies of the target, which is *basic.target*. If it's not currently loaded, systemd attempts to start the *basic.target* after resolving all of its dependencies, and so on.

1.6 Understanding systemd Targets

Unit files are everything that can be started by systemd. A category of unit files are *targets*. Systemd targets are a collection of unit files that are meant to work together to let the system enter a specific state. Some of these targets are the equivalent of runlevels in the previous versions of RHEL. For example:

Options	Description
poweroff.target	State that shuts down the computer.
rescue.target	Lets the system enter a troubleshooting mode.
multi-user.target	Fully operational server with a command line, but without a GUI.
graphical.target	Fully operational server with a GUI.
reboot.target	State that causes the computer to reboot.
emergency.target	Minimalistic rescue mode, to be used when rescue mode fails.

1.6.1 Services related to targets

The services need to know which target they belong to, and the targets themselves need to know about the ordering. By the use of *wants*, every service knows by which target it is wanted. For example, every service has an Install section containing the name of the target that wants it. The `sshd.service` has:

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

Ordering

The order between targets is defined in targets. For example, the *multi-user.target* file contains:

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

Here, we see that the target (and consequently, the services in it) can only be loaded if the basic.target is already loaded (since it's required). Further, systemd may only attempt to start the services in this target *after* the basic.target has been loaded, and the conflicted *rescue.target* was ordered to load (but didn't). The `AllowIsolate=yes` signifies whether the system can jump from another target to this target to change it's state.

1.7 Switching between systemd Targets

While changing from one system state to another, only certain targets may switch to another one from an operational environment, but in may cases, we can't. For example, it is possible to go from an operational environment to a minimal environment such as the rescue mode.

However, any target can be booted to from the Grub boot menu. The currently active targets can be listed with:

```
# systemctl list-units --type=target
UNIT                    LOAD   ACTIVE SUB    DESCRIPTION
basic.target            loaded active active Basic System
bluetooth.target        loaded active active Bluetooth
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-online.target  loaded active active Network is Online
network-pre.target     loaded active active Network (Pre)
network.target         loaded active active Network
nfs-client.target      loaded active active NFS client services
nss-user-lookup.target loaded active active User and Group Name Lookups
paths.target           loaded active active Paths
remote-fs-pre.target   loaded active active Remote File Systems (Pre)
remote-fs.target       loaded active active Remote File Systems
slices.target          loaded active active Slices
sockets.target         loaded active active Sockets
sound.target           loaded active active Sound Card
swap.target            loaded active active Swap
sysinit.target         loaded active active System Initialization
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.

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

1.7.1 Switching to another target from an operational environment

Working environments consist of multiple targets, some of which are listed above. To change to another target (mode), we use the `systemctl isolate` command:

```
# systemctl isolate rescue.target
Give root password for maintenance
(or type Control-D to continue):
```

To exit rescue mode, we must just type `exit` and let the computer reboot, since it's not possible to easily switch from the rescue mode to any other mode.

1.7.2 Selecting target from Grub Boot menu

When the grub boot menu is displayed, and the available kernels are shown, we can press the `e` key to enter boot options. In here, we have to go down to the line that starts with `linux16` and at the very end, we type: `systemd.unit=<targetName>.target` to boot into it. For example, to boot into the rescue mode during boot, we use:

```
systemd.unit=rescue.target
```

Then, we have to press *CTRL+X* to execute. This will directly boot us into the rescue mode. In this mode, the `systemctl list-units --type=target` returns only a few targets, which proves that this mode is indeed minimalistic, but also the loaded targets (i.e., the services loaded by them) are essential for proper functioning of the computer.

1.7.3 Emergency mode

To boot into the emergency mode we need to use `systemctl.unit=emergency.target`. In this mode, `systemctl list-units --type=targets` doesn't return anything. We can use `systemctl default` to start the default target.

1.8 Managing File System mounts in a systemd Environment

Other than using `/etc/fstab`, systemd also provides a way to mount file systems. Further, not all file systems are mounted (or available) using `/etc/fstab`. The file systems that can be mounted using systemd can be obtained by:

```
# ls *.mount
dev-hugepages.mount            sys-kernel-config.mount
dev-mqueue.mount               sys-kernel-debug.mount
proc-fs-nfsd.mount             tmp.mount
proc-sys-fs-binfmt_misc.mount  var-lib-nfs-rpc_pipefs.mount
sys-fs-fuse-connections.mount
```