Booting with GRUB Legacy, GRUB 2 and UEFI

Author: Yogesh Babar
Technical Reviewer: Chris Negus
10/20/2015

INTRODUCTION

GRUB is a separate world itself. It's so amazing and so huge that someone could write a whole book on it.

The first stable release of GRUB 2 was in June, 2012. It started shipping in enterprise systems with Red Hat Enterprise Linux 7 (December, 2013) and CentOS 7 (July, 2014). Its release created a buzz, but was also confusing to users when they tried to use, understand, or configure GRUB 2. Users tried to compare it with the original GRUB (now called GRUB legacy), but nothing matched the way that GRUB 2 worked.

So, first of all, do not compare it with GRUB legacy. The GRUB community has changed the entire structure for GRUB 2, as I will discuss soon. First let's try to understand why GRUB 2.

Why GRUB 2?

"Because GRUB Legacy has become unmaintainable, due to messy code and design failures. We received many feature requests, and extended GRUB beyond the original scope, without redesigning the framework. This resulted in the state that it was impossible to extend GRUB any further without rethinking everything from the ground."

-- GNU GRUB FAQ (https://www.gnu.org/software/grub/grub-faq.html)

In short, the GNU GRUB project decide to stop supporting the original version of GRUB because:

- New feature requests were being made against an old code base
- Newer hardware, boot environments needed to be supported
- The original GRUB code is more than a decade old
- The original code was unmaintainable going forward

It was not users, but rather the GRUB developers who decided to write something from a scratch to give extra features. Before we start to see what was developed, lets see what developers have planned for GRUB 2 in future. Its simply amazing. Below are some of the ideas which they might develop:

- Full USB support
- LUKS support
- A very fancy menu implementation, which supports animations, colorful effects, style sheets, etc.
- Adding the **parted** tool to the GRUB architecture. That means if you are in trouble at boot time, from the grub command prompt you can really play with your disks (creation, resize, deletion of partitions). Isn't this amazing?

As you may know, GRUB 2 works on BIOS as well as on UEFI firmwares. So here in this tech brief I use two systems: one is BIOS based and another one is UEFI based system. Also we would focus more on GRUB 2 with UEFI combination.

On BIOS-based systems

GRUB legacy keep all of its files in the **/boot/grub** directory but GRUB 2 keeps its configuration and binary files at 3 different locations:

- **/boot/grub2/** or **/boot/efi/EFI** (only on UEFI system)
- **/etc/default/**
- **/etc/grub.d/**

Let's see some examples of the contents of these files. On a system that boots from BIOS, type the following:

```
# ls /boot/grub2/
device.map  fonts  grub.cfg  grubenv  i386-pc  locale  themes
```

The main configuration file for GRUB 2 is **grub.cfg**. For GRUB legacy, this file was called **grub.conf.** Here is an example of the contents of the first part of a **grub.cfg** file:

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub2-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#

### BEGIN /etc/grub.d/00_header ###
set pager=1

if [ -s $prefix/grubenv ]; then
  load_env
fi
if [ "${next_entry}" ] ; then
```

```
   set default="${next_entry}"
   set next_entry=
   save_env next_entry
   set boot_once=true
else
   set default="${saved_entry}"
fi

if [ x"${feature_menuentry_id}" = xy ]; then
  menuentry_id_option="--id"
else
  menuentry_id_option=""
fi

export menuentry_id_option

if [ "${prev_saved_entry}" ]; then
  set saved_entry="${prev_saved_entry}"
  save_env saved_entry
  set prev_saved_entry=
  save_env prev_saved_entry
  set boot_once=true
fi

function savedefault {
  if [ -z "${boot_once}" ]; then
    saved_entry="${chosen}"
    save_env saved_entry
  fi
}

function load_video {
  if [ x$feature_all_video_module = xy ]; then
    insmod all_video
  else
    insmod efi_gop
    insmod efi_uga
    insmod ieee1275_fb
    insmod vbe
    insmod vga
    insmod video_bochs
    insmod video_cirrus
  fi
}

terminal_output console
if [ x$feature_timeout_style = xy ] ; then
  set timeout_style=menu
  set timeout=5
# Fallback normal timeout code in case the timeout_style feature is
# unavailable.
...
```

The **grub.cfg** file is a script file and it's harder to understand than GRUB legacy's **grub.conf** file. Also it is not recommended to edit this file directly, because it is created from the content of other configuration files. So any changes you make directly to **grub.cfg** will be overwritten and lost eventually.

To add custom entries to your GRUB 2 configuration, there are other files you should look at. Here are some examples of those files:

```
# cat /etc/default/grub
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="rd.lvm.lv=rhel_unused/root crashkernel=auto
  rd.lvm.lv=rhel_unused/swap vconsole.font=latarcyrheb-sun16
  vconsole.keymap=us rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
```

The **/etc/default/grub** file is used for customization like changing the font type or size, adding a different background or even passing a kernel parameter. To create the **grub.cfg** file itself, there is a set of files in the **/etc/grub.d/** directory:

```
# ls /etc/grub.d/
00_header   20_linux_xen     30_os-prober  41_custom
10_linux    20_ppc_terminfo  40_custom     README
```

The files in **/etc/grub.d/** directory are script files that have executable permissions and each runs based on the number prefix on its name. That means first **10_linux** file will run and then **30_os-prober**.

On UEFI based systems

There are different GRUB configuration files for UEFI based-systems. First, check the **/boot/grub2/** directory:

```
# ls /boot/grub2/
grubenv  themes
```

Notice there is no **grub.cfg** file here. For that, you need to look in the **/boot/efi/EFI/redhat/** directory:

```
# ls /boot/efi/EFI/redhat/
BOOT.CSV  gcdx64.efi  grubx64.efi     shim.efi
fonts     grub.cfg    MokManager.efi  shim-redhat.efi
```

Notice the **grub.cfg** file has been shifted to this location. The other ***.efi** files in this directory are GRUB 2 binaries which will be used by UEFI firmware at the time of boot.

```
# ls /etc/grub.d/
```

```
00_header   20_linux_xen      30_os-prober  41_custom
10_linux    20_ppc_terminfo   40_custom     README
```

Let's talk about these files:

- **00_header:** Is for internal GRUB 2's usage, which I will not be covering.
- **10_linux:** Is an interesting script file. It is responsible for finding the pre-installed Linux based operating system. That means it will be finding kernel and initramfs files from the harddrive and will be adding the entries into **grub.cfg** file. Does that mean you do not need to add the other Linux OS entries manually? Yes, it does.
- **20_linux_xen:** Will find xen kernels and add their entries into **grub.cfg**.
- **20_ppc_terminfo:** Is related to PPC architecture, which I will not be covering.
- **30_os-prober:** Is the most interesting executable file. This script file is responsible for finding the non-linux-based operating systems. Does that mean a Windows OS? Yes, when it runs, 30_os-prober finds any non-Linux operating systems from your hard drive and adds appropriate entries into **grub.cfg** file. That also means that if multiple operating systems are already on your systems, including Linux, Windows, or UNIX systems, if you are installing RHEL 7, CentOS 7 or Fedora, you do not need to add your earlier OS entries in GRUB. GRUB 2 will run these scripts and it will find out the other pre-installed operating systems on its own, then add the appropriate entries into the main configuration file (**grub.cfg**). This is simply amazing.
- **40_custom and 41_custom:** As we have discussed earlier it is not recommended to edit the **grub.cfg** file directly. But what if you want to add some custom entries into **grub.cfg**. These two files provide the answer for this. If you want to add custom entries, add them to **40_custom**' or in **41_custom** file. When you re-create a **grub.cfg** file it will execute these files and add your custom entries into **grub.cfg**. We see how to re-create a **grub,cfg** file later on.
  Also note that you can create your own script file for your custom entry. No need to depend on 40_custom or 41_custom files. Just make sure to assign a number to it along with executable permission.

Since we will be dealing with GRUB 2 on UEFI systems then we need to first understand the way UEFI firmware boots the Linux operating system.

UEFI firmware booting

Before beginning to work with GRUB on your computer, let's try to figure out whether your system has UEFI firmware or BIOS. There are a few ways you can check your system to see if it uses UEFI boot firmware:

1. A very simple trick is to go to your firmware. If you are able to use a mouse, it is a

UEFI system. In that case, you might see a nice graphical interface like the one shown in Figure 1:

Figure 1: UEFI firmware system with an Intel Visual BIOS GUI.

2. A way to check an installed Linux system to see if it has UEFI support is to run **efibootmgr -v**. The output here indicates that you have UEFI firmware:

```
# efibootmgr -v
BootCurrent: 0005
BootOrder: 0005,0000,0001,0002,0003,0004
Boot0000* EFI VMware Virtual SCSI Hard Drive (0.0)
     ACPI(a0341d0,0)PCI(10,0)SCSI(0,0)
Boot0001* EFI VMware Virtual SATA CDROM Drive (0.0)
    ACPI(a0341d0,0)PCI(11,0)PCI(5,0)03120a00000000000000
Boot0002* EFI VMware Virtual SATA CDROM Drive (1.0)
    ACPI(a0341d0,0)PCI(11,0)PCI(5,0)03120a00010000000000
Boot0003* EFI Network ACPI(a0341d0,0)PCI(11,0)PCI(1,0)MAC(000c2927a8fd,0)
Boot0004* EFI Internal Shell (Unsupported option) MM(b,e1a2000,e42ffff)
Boot0005* Red Hat Enterprise Linux
   HD(1,800,64000,a17c521e-8435-4859-83c0-cb923874a846)File(\EFI\redhat\shim.efi)
```

Here BootCurrent, BootOrder, and so on, are called as environment variables. If you fire up the same command on a BIOS-based system, you get something like this:

```
# efibootmgr -v
efibootmgr: EFI variables are not supported on this system.
```

UEFI firmware booting structure

UEFI firmware requires one FAT32 partition referred to as the EFI System Partition (ESP). The operating system has to put the bootloader into this partition.

The operating system has to maintain its own directory in the ESP and install a bootloader there. Red Hat use the following directory names for its operating systems:

```
For fedora OS ⇒ fedora
For cent OS   ⇒ centos
For RHEL OS   ⇒ redhat
```

All files related to GRUB 2 are in the directory that is associated with the operating system you want to run. Figure 2 illustrates a hard disk on a UEFI system that has three operating systems installed (RHEL 7, CentOS 7, and Fedora 22):

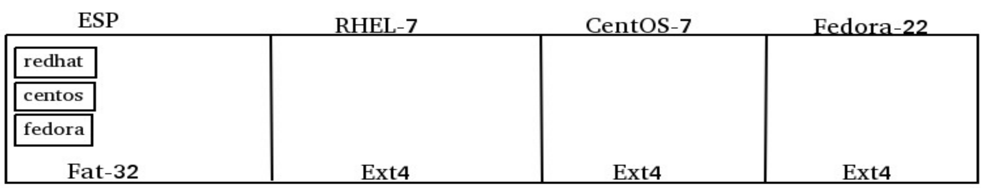


Figure 2: UEFI disk partitioning

For the three operating systems installed on this system, each has to either create or use the existing FAT-32 formatted ESP partition. At the time of installation, every OS installs their respective boot loaders (grub2) into their respective directories (inside ESP). On BIOS-based systems, the same boot loaders used to get installed in the boot sector (which is a 512-bytes reserved space at the beginning of partition). BIOS systems also used to have stage1, stage1.5 and stage2. GRUB 2 with UEFI does not have such stages.

A **/boot/efi** directory is mounted automatically at boot time inside each operating system's ESP partition, based on an entry in the **/etc/fstab** file:

```
# /etc/fstab
# Created by anaconda on Wed Oct  8 13:37:36 2014
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
/dev/mapper/rhel_unused-root               /         xfs   defaults    1 1
UUID=ef0e8997-77a0-4bfd-830d-eb9bc26ca3e3 /boot     xfs   defaults    1 2
UUID=918D-4200                            /boot/efi xfs   defaults    1 2
/dev/mapper/rhel_unused-swap              swap      swap  defaults    0 0
```

To see all the files related to GRUB boot loader for a UEFI system, you can run the tree command as follows:

```
# tree /boot/
/boot/
├── config-3.10.0-121.el7.x86_64
├── efi
│   ├── EFI
│   │   ├── BOOT
│       │   ├── BOOTX64.EFI
│       │   └── fallback.efi
│       |── redhat
│           ├── BOOT.CVS
│           ├── fonts
│           │   └── unicode.pf2
│           ├── gcdx64.efi
│           ├── grub.cfg
│           ├── grubx64.efi
│           ├── MokManager.efi
│           ├── shim.efi
│           └── shim-redhat.efi
├── grub2
│   ├── grubenv
│   ├── themes
│   │   └── system
├── initramfs-0-rescue-7099d5868c48473c993b85d60bafd021.img
├── initramfs-3.10.0-121.el7.x86_64.img
├── initramfs-3.10.0-121.el7.x86_64kdump.img
├── initrd-plymouth.img
├── symvers-3.10.0-121.el7.x86_64.gz
├── System.map-3.10.0-121.el7.x86_64
├── vmlinuz-0-rescue-7099d5868c48473c993b85d60bafd021
└── vmlinuz-3.10.0-121.el7.x86_64
```

Notice that the kernel (**vmlinuz**) and initial RAM filesystem (**initramfs**) files are inside **/boot** but the GRUB configuration files are inside the **/boot/efi** directory, which is mounted from a separate partition. You can see this by running the **df** command:

```
# df
Filesystem       1K-blocks     Used Available Use% Mounted on
...
/dev/sda2           508588   113272    395316  23% /boot
/dev/sda1           204580     9744    194836   5% /boot/efi
```

UEFI firmware booting sequence

When computer starts, UEFI firmware checks environment variables and gets the bootloader location from ESP. For example:

```
Boot0005* Red Hat Enterprise Linux
HD(1,800,64000,a17c521e-8435-4859-83c0-cb923874a846)File(\EFI\redhat\grubx64.efi)
```

It goes inside that particular directory and calls the bootloader. In this case, the bootloader is GRUB 2:

```
(\EFI\redhat\grubx64.efi)
```

Manual booting with GRUB

To understand GRUB, we need to understand first what it does in the background. Learning to boot an operating system manually from GRUB will help us in that. Here I will be using three systems. First we will see how to manually boot a system with GRUB legacy. Then we will see how to boot a system manually with GRUB 2, on BIOS as well as on UEFI system.

GRUB Legacy Manual Booting

First let's find out the root device name. We can get it by running **blkid** and **df**, and looking at the **/etc/fstab** file. On this RHEL 6 BIOS-based system, **/dev/sda2** is the root device:

```
# blkid
/dev/sda1: UUID="ef0e8997-77a0-4bfd-830d-eb9bc26ca3e3" TYPE="ext4"
/dev/sda2: UUID="PpR3qi-SGTa-j70a-Insi-FpYW-cQyV-qwYQE8" TYPE="ext4"
/dev/sda3: UUID="23c7864c-7247-4a24-8824-6765b35bbcc9" TYPE="swap"
# cat /etc/fstab
...
UUID=PpR3qi-SGTa-j70a-Insi-FpYW-cQyV-qwYQE8 /       ext4    defaults   1 1
UUID=ef0e8997-77a0-4bfd-830d-eb9bc26ca3e3   /boot   ext4    defaults   1 2
UUID=23c7864c-7247-4a24-8824-6765b35bbcc9   swap    ext4    defaults   1 2
...
# df
Filesystem       1K-blocks     Used Available Use% Mounted on
/dev/sda2          8985528  2693084   5835996  32% /
tmpfs               548284      224    548060   1% /dev/shm
/dev/sda1           297485    34690    247435  13% /boot
```

When you first boot your computer, the moment the GRUB screen comes up, you need to press **c** to drop to a grub> prompt. At the grub> prompt we need run three commands, as illustrated in Figure 3:

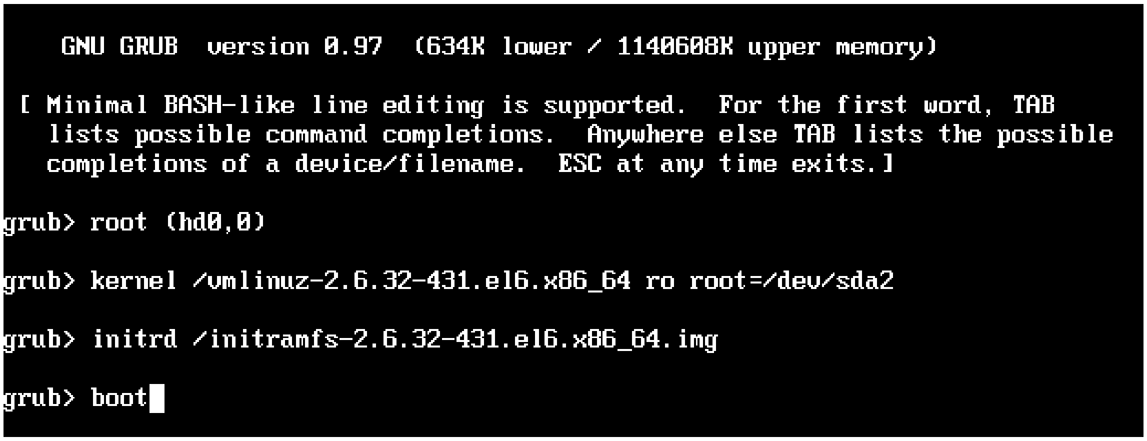


Figure 3: Booting from the grub> prompt

Here is what those three GRUB commands do:

`root (hd0,0)`

This command tells a GRUB legacy system where is the kernel (**vmlinuz**) and **initramfs** files stored. Remember do not get confused with the word **root**. In this case, it does not mean the root device where the operating system has been installed. Rather it means the **/boot** device where the **vmlinuz** and **initramfs** files are stored. Now if the **/boot** mount point is not a separate device, then you need to mention the root filesystem's device name. But if you have **/boot** on a different device (as is the default with RHEL) then you need to mention the **/boot** device name. In this example you can see in above screenshot that **/boot** is a separate device (**/dev/sda1**).

`kernel /vmlinuz-2.6.32-431.el6.x86_64 ro root=/dev/sda2`

As the name suggest here you need to mention the location of the kernel file (vmlinuz-XX) in relation to the root of the **/boot** filesystem (**root=/dev/sda2**).

`initrd` /initramfs-2.6.32-431.el6.x86_64.img

Here we specify the **initramfs-*xxx*.img** file name (relative to its location in the root of the **/boot** filesystem).

When we enter the **boot** command, GRUB legacy goes to the hard disk number 1, partition number 1 (remember in GRUB hard disk and partition number starts from 0) which holds the **/boot** filesystem. From there, it will copy the kernel and initramfs files to RAM. After this, the kernel mounts the root filesystem (/dev/sda2) in read only mode and start the **init** process. This **init** process will read the **/etc/fstab** file from mounted root filesystem and will remount the root filesystem in read-write mode.

If you do not interrupt GRUB legacy, as we just did, then GRUB takes root, kernel, and initramfs inputs from the **/boot/grub/grub.conf** file:

```
# cat /boot/grub/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/sda2
#          initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Workstation (2.6.32-431.el6.x86_64)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-573.7.1.el6.x86_64 ro root=/dev/mapper/HelpDeskRHEL6-Root
rd_LVM_LV=HelpDeskRHEL6/Swap rd_LUKS_UUID=luks-6806f875-ee29-47d5-8f59-cc3c32731cd9
rd_NO_MD SYSFONT=latarcyrheb-sun16 crashkernel=auto LANG=en_US.UTF-8  KEYBOARDTYPE=pc
KEYTABLE=us rd_LVM_LV=HelpDeskRHEL6/Root rd_NO_DM rhgb quiet vga=0x318 rhgb quiet
        initrd /initramfs-2.6.32-431 .el6.x86_64.img
```

GRUB 2 on BIOS

On this RHEL 7 BIOS-based system root device name is **/dev/mapper/rhel-root**. It's a LVM device:

```
# blkid
/dev/sda1: UUID="5baf9cbb-6b75-4733-bbb0-eb4389cda038" TYPE="ufs"
/dev/sda2: UUID="66Vk22-HGHS-3sH0-w7Zr-omJ0-xbtn-JOZRs0" TYPE="LVM2_member"
/dev/mapper/rhel-root: UUID="a997c74e-3dc2-4b1d-a7cc-c1abe40c1352" TYPE="xfs"
/dev/mapper/rhel-swap: UUID"6e877193-2b68-43c1-95d6-ca55433f91ce" TYPE="swap"
# cat /etc/fstab
...
/dev/mapper/rhel-root                      /       xfs    defaults  1 1
UUID=5baf9cbb-6b75-4733-bbb0-eb4389cda038  /boot   xfs    defaults  1 2
/dev/mapper/rhel-swap                      swap    swap   defaults  0 0
```

```
# df
Filesystem             1K-blocks     Used Available Use% Mounted on
/dev/mapper/rhel-root    8910848  3926964   4983884  45% /
...
/dev/sda1                 508588  4138442   2777912  24% /boot
```

As with GRUB legacy, on a GRUB 2 system you need to press **c** when GRUB 2's splash screen comes up. It will drop to GRUB 2's prompt. After that, there are three commands you need to type that are very similar to the GRUB legacy commands, but with a slight change. Figure 4 shows you an example of those commands on a GRUB 2 boot screen:

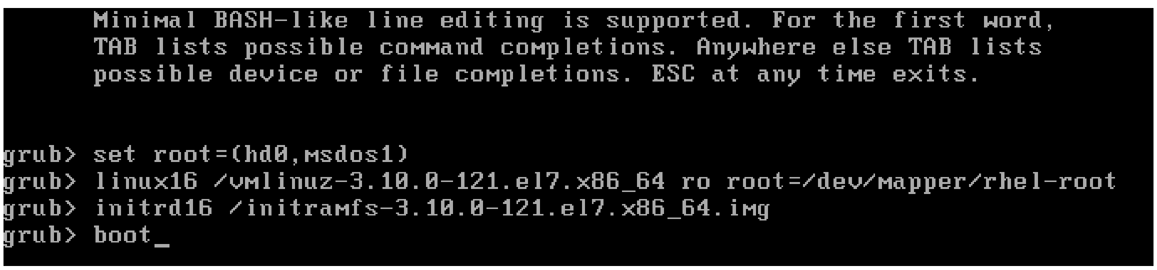


Figure 4: Booting GRUB 2 from the grub> prompt

Here are descriptions of those commands:

set root=(hd0,msdos1)

This is just like GRUB legacy **root** command. Here, it specifies the partition containing the **/boot** device (**/dev/sda1**). In GRUB 2 hard disk numbers still start with 0, but partition number starts from 1. Partition numbers will be like **msdos1**, **msdos2**, and so on. The msdos name tells BIOS firmware that the partition uses an msdos partition table.

linux16 /vmlinuz-3.10.0-121.el7.x86_64 ro root=/dev/mapper/rhel-root

The kernel filename is identified here along with the name of the partition containing the root (**/**) filesystem (**/dev/mapper/rhel-root**).

initrd16 /initramfs-3.10.0-121.el7.x86_64.img

The initramfs file identifies the location of the initial filename used to boot the kernel.

On BIOS-based systems, if we do not interrupt GRUB then it will take inputs for **set root**, **linux16**, and **initrd16** from **/boot/grub2/grub.cfg**. Here is an example of some of the contents of a **grub.cfg** file:

```
# Fallback normal timeout code in case the timeout_style feature is
# unavailable.
else
  set timeout=5
```

```
fi
### END /etc/grub.d/00_header ###

### BEGIN /etc/grub.d/10_linux ###
menuentry 'Red Hat Enterprise Linux Server (3.10.0-229.4.2.el7.x86_64) 7.0 (Maipo)'
--class red --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option
'gnulinux-3.10.0-123.el7.x86_64-advanced-64e5e9df-80a1-401f-9ddf-2db389f89fe8' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_msdos
        insmod xfs
        set root='hd0,msdos1'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1
--hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1'
ef0e8997-77a0-4bfd-830d-eb9bc26ca3e3
        else
          search --no-floppy --fs-uuid --set=root ef0e8997-77a0-4bfd-830d-eb9bc26ca3e3
        fi
        linux16 /vmlinuz-3.10.0-229.4.2.el7.x86_64 root=/dev/mapper/rhel_unused-root ro
rd.lvm.lv=rhel_unused/root  rd.lvm.lv=rhel_unused/swap vconsole.font=latarcyrheb-sun16
vconsole.keymap=us rhgb quiet LANG=en_US.UTF-8
        initrd16 /initramfs-3.10.0-229.4.2.el7.x86_64.img
}
...
```

After some default settings, content from the **/etc/grub.d/10_linux** file is included to create the menuentry for "Red Hat Enterpise Linux...". The linux16 and initrd16 lines under that menuentry identify the kernel and initramfs to use when that item is selected.

GRUB 2 on UEFI:

On this RHEL 7 UEFI-based system, the root device name is again **/dev/mapper/rhel-root** which is an LVM device:

```
# blkid
/dev/sda1: SEC_TYPE="msdos" UUID="91BD-420D" TYPE="vfat" PARTLABEL="EFI SystemPartition"
   PARTUUID="a17c521e-8435-4859-83c0-cb923874a846"
/dev/sda2: UUID="d2f1962d-2210-437e-b6ac-9c6afc8e462e" TYPE="xfs"
   PARTUUID="65d20f6e-cdd4-4a19-b51a-43075801c1"
/dev/sda3: UUID="d81Gqz-8Vd0-kLNI-9ICX-CThF-DEnW-xGskF1" TYPE="LVM2_member"
   PARTUUID="91a01233-5152-4a67-9a16-fffec2a8c286"
/dev/mapper/rhel-root: UUID="f99eeed5-f108-4ece-9361-3c49c8da3d8c" TYPE="xfs"
/dev/mapper/rhel-swap: UUID"bdfbd2c1-93ac-42a8-a011-40c08046aacb" TYPE="swap"
```

```
# cat /etc/fstab
...
/dev/mapper/rhel-root                      /         xfs    defaults                  1 1
UUID=d2f1962d-2210-437e-b6ac-9c6afc8e462e  /boot     xfs    defaults                  1 2
UUID="91BD-420D"                           /boot/efi vfat umask=0077,shortname=winnt  0 0
/dev/mapper/rhel-swap                      swap      swap defaults                    0 0
# df
Filesystem             1K-blocks     Used Available Use% Mounted on
/dev/mapper/rhel-root    8706048  3947040   4759008  46% /
...
/dev/sda2                 508588   113272    395316  24% /boot
/dev/sda1                 204580     9744    194836  24% /boot/efi
```

On UEFI systems, commands to boot from a GRUB command prompt are just about the same, with only a few slight change as illustrated in Figure 5:

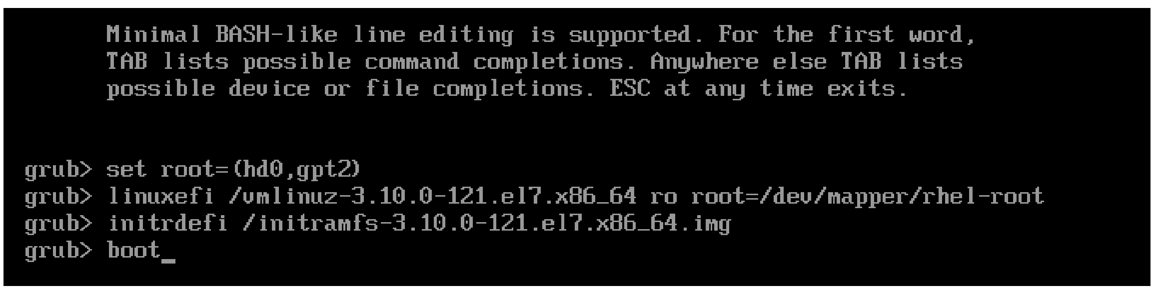


Figure 5: Booting GRUB 2 from the grub> prompt in UEFI

Here are descriptions of those commands:

set root=(hd0,gpt2)

As usual we need to mention a device name where the kernel (**vmlinuz**) and initramfs files have been stored. The hard disk number starts from 0 but the partition number starts from 1. The UEFI firmware uses a gpt partition table so the partition names will be gpt1, gpt2, and so on. On this system, partition number 1 is our ESP mounted on **/boot/efi** and partition number 2 is a **/boot** device which is **/dev/sda2**.

linuxefi /vmlinuz-3.10.0-121.el7.x86_64 ro root=/dev/mapper/rhel-root

As usual we need to pass the kernel filename (**vmlinuz**) along with the device that host the root (**/**) filesystem. In this system root device is **/dev/mapper/rhel-root** which is an LVM device.

initrdefi /initramfs-3.10.0.121.el7.x86_64.img

Here we need to pass initramfs file name. Its location is relative to the root of the **/boot/efi** directory.

On UEFI-based systems, if we do not interrupt GRUB then it will take inputs of **set root**, **linuxefi**, and **initrdefi** from **/boot/efi/EFI/redhat/grub.cfg**. Here is an example of the **grub.cfg** file on a UEFI-based system:

```
# Fallback normal timeout code in case the timeout_style feature is
# unavailable.
else
  set timeout=5
fi
### END /etc/grub.d/00_header ###

### BEGIN /etc/grub.d/10_linux ###
menuentry 'Red Hat Enterprise Linux Server (3.10.0-229.4.2.el7.x86_64) 7.0 (Maipo)'
--class red --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option
'gnulinux-3.10.0-123.el7.x86_64-advanced-f99eeed5-f108-4ece-9361-3c49c8da3d8c' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2
--hint-baremetal=ahci0,gpt2 d2f1962d-2210-437e-b6ac-9c6afc8e462e
        else
          search --no-floppy --fs-uuid --set=root d2f1962d-2210-437e-b6ac-9c6afc8e462e
        fi
        linuxefi /vmlinuz-3.10.0-229.4.2.el7.x86_64
root=UUID=f99eeed5-f108-4ece-9361-3c49c8da3d8c ro rc.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
vconsole rd.lvm.lv=rhel_unused/root  rd.lvm.lv=rhel_unused/swap
vconsole.font=latarcyrheb-sun16 vconsole.keymap=us rhgb quiet LANG=en_US.UTF-8
        initrd16 /initramfs-3.10.0-121.4.2.el7.x86_64.img
}
..
```

GRUB 2 is so smartly designed that you can even explore any particular partition from grub> prompt itself. Here are examples of commands run from the GRUB prompt to investigate file systems.

1. GRUB 2 has its own **ls** command that lists the attached storage devices along with its partitions:

   ```
   grub> ls
   (lvm/rhel-root) (lvm/rhel-swap) (hd0) (hd0,gpt3) (hd0,gpt2) (hd0,gpt1) (hd1)
   (hd1,msdos2) (cd0)
   ```

2. The same **ls** command can show us the any particular partition's filesystem:

```
grub> ls (hd0,gpt1)
 (hd0,gpt1): Filesystem is fat.
```

3. The GRUB ls command can even list the contents of the partition and directories:

```
grub> ls (hd0,gpt1)/
efi/
grub> ls (hd0,gpt1)/efi
./ ../ redhat/ boot/
grub> ls (hd0,gpt2)/
./ ../ efi/ grub2/ .vmlinuz-3.10.0-121.el7.x86_64.hmac
System.map-3.10.0-121.el7.x86_64 config-3.10.0-121.el7.x86_64
symvers-3.10.0-121.el7.x86_64.gz vmlinuz-3.10.0-121.el7.x86_64
initrd-plymouth.img initramfs-0-rescue-7099d5868c48473c993b85d60bafd021.img
vmlinuz-0-rescue-7099d5868c48473c993b85d60bafd021
initramfs-3.10.0-121.el7.x86_64.img initramfs-3.10.0-121.el7.x86_64kdump.img
```

GRUB 2 and UEFI firmware

GRUB 2 and UEFI together form an amazing combination. Just keep some things in mind:

1. GRUB 2 understands msdos and gpt partition tables.
2. UEFI firmware understands msdos and gpt partition tables.
3. Using UEFI, GRUB 2, and gpt together works like a charm and is recommended.
4. Using UEFI, GRUB 2, and msdos together is not recommended. It does not mean it does not work but it will make life very difficult.
5. Using BIOS, GRUB 2, and msdos together works like a charm.
6. Using BIOS, GRUB 2, and gpt works, but can make your life difficult.

Let's try to first understand the way UEFI firmware works. As we all know BIOS is deprecated. That means there will not be new BIOS development happening. BIOS had many limitations, such as:

1. You can only create 4 primary partitions. If you want to create 5th one then create the secondary/extended partition and inside that container you can create a fixed number of logical partition (16 is the maximum).
2. A partition size can not be more than 2.2 TB.
3. It takes a long time to boot an operating system.
4. It is dumb. It does not understand operating systems or boot loaders. It only knows to jump on a hard disk's first 512 bytes. This area of a disk is called the *boot sector*.
5. It does not have GUI or even a mouse support.

6. It struggles to initialize USB devices.
7. It has CPU and memory level limitations.

But UEFI is really smart and nicely designed. It has tremendous features like:

1. It is smart. It understands the operating system and boot loaders.
2. It is fast and robust. It handles USB devices smoothly.
3. It has maintenance tools.
4. A partition limit for UEFI is 8.2 zeta bytes, which is really huge.
5. It has very nice GUI implementation along with mouse support.
6. It can use the full CPU and all attached RAM.
7. It treats the bootloader as a simple application.
8. There are standards from the UEFI forum (uefi.org) that every operating system vendor has to follow.
9. The 'Secure Boot' feature can secure you from bootable viruses or, in other words, it can secure boot loaders from viruses.
10. It does not jump on the boot sector. Rather it jumps on a ESP partition which has much much bigger space than 512 bytes which BIOS used to use at the time of boot.
11. It provides a shell.

So as we have seen earlier, on a UEFI system GRUB 2 gets installed inside the **/etc/efi/EFI** directory. In the **/etc/efi/EFI** directory, the operating system creates its own sub-directory and there it will install the GRUB 2 bootloader. Red Hat installs its bootloader in **/boot/efi/EFI/redhat/**. Here's what the contents of that directory looks like:

```
# ls /boot/efi/EFI/redhat
BOOT.CSV    gcdx64.efi  grubx64.efi       shim.efi
fonts       grub.cfg    MokManager.efi    shim-redhat.efi
```

Notice the files that end with the **.efi** extension. These files are binaries/executables that are executed by UEFI firmware:

grubx64.efi:

This is the actual GRUB 2 file. UEFI firmware executes this file. The **grubx64.efi** file reads the **grub.cfg** file. The contents of the **grub.cfg** file determines the options displayed on the boot menu that let the user choose the appropriate kernel or operating system.

shim-redhat.efi and **shim.efi:**

This is a separate boot loader. You may wonder how you can have a boot loader inside a boot loader. Let me explain. UEFI has a *secure boot* feature. Every vendor has to lock their bootloaders with their private key and need to ship public key within the UEFI firmware. Whenever the UEFI-based system starts booting, it will check the environment variables and will go inside the ESP and run the respective boot loader's particular **.efi** file. But before running that file, it gets the public key of that bootloader. If the key matches, it will allow the bootloader to run. If the key does not match, the bootloader is considered to be malicious code and execution stops.

This sounds like a good approach, but the glitch is that the Microsoft bootloader key will not match Red Hat's **grubx64.efi** file. So UEFI firmware will consider it as malicious code and GRUB 2 will not be allowed to run. So, when it comes to Linux, there are the following options:

1. Each Linux distribution could create its own key pair and start shipping the public key with every hardware vendor. But as you may know, there are more than 250 Linux distributions available. Obviously it's not possible to put every distro's key in UEFI firmware.
2. All the Linux distributions could sign their boot loaders with only one key pair. But it will be very hard to secure such key pairs.
3. Vendors could provide a *secure boot disable* feature in their UEFI firmware implementation. But we do not have control of that.
4. For Microsoft, it's very easy to ship their key with every hardware vendor because of its business model. But for Linux community, it's difficult to ship the key with every vendor. So the option is to sign the GRUB 2 bootloader with Microsoft's key. The problem with relying on Microsoft to sign the GRUB 2 bootloader is that the GRUB 2 implementation becomes dependent on Microsoft. To make this work, a Linux vendor can build a smaller, dumb, initial bootloader which will be signed by Microsoft's key and it will call the original GRUB 2 bootloader.

The last choice is what Red Hat has done. The initial bootloader is called a **shim** bootloader. If the secure boot feature is enabled, then either **shim-redhat.efi** or **shim.efi** is called first and that shim bootloader will then call the original GRUB 2 bootloader (**grubx64.efi**).

MokManager.efi:

If you want to add your own custom key in UEFI firmware then you need to execute this binary to do that.

UEFI Shell

The UEFI shell is an amazing feature. As we discussed earlier, UEFI firmware considers the bootloader to be an application. In the same way, the UEFI shell is another application that is shipped with the firmware. Now why are we discussing the UEFI shell here? Because by understanding the UEFI shell we can understand the way GRUB 2 is called by the firmware. Another benefit of understanding the UEFI shell is if you are facing issues with your GRUB bootlooader at the time of boot. In that case, you can use the UEFI shell for debugging. Here's how you can call a GRUB 2 bootloader from a UEFI shell:

1. Boot the system, go to the boot menu, and select **EFI shell**, as shown in Figure 6:

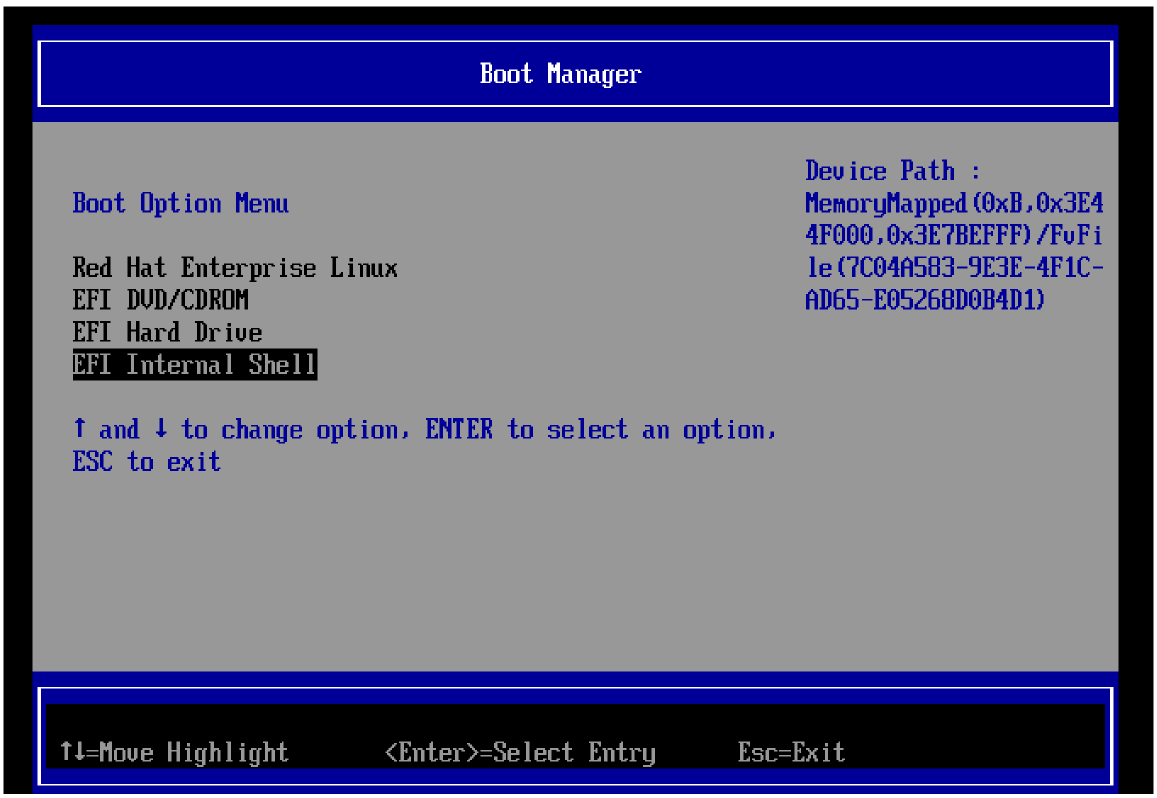


Figure 6: Getting to the UEFI shell

2. After the UEFI shell appears, run a **map** command. It will list filesystems and storage devices available on the system, as shown in Figure 7:

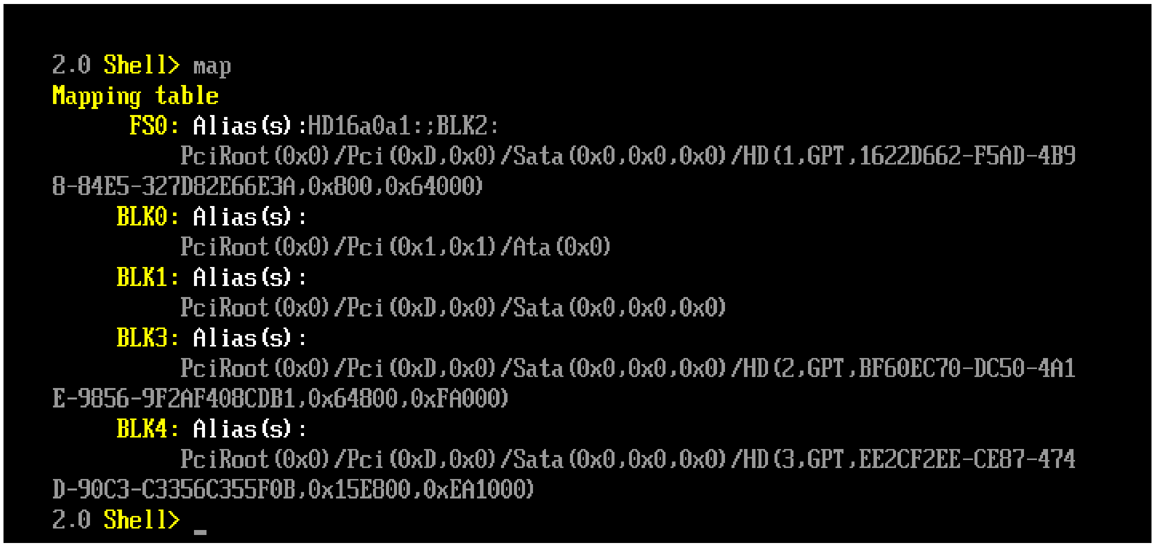


Figure 7: Run **map** to see storage devices from the UEFI shell

3. Choose the first filesystem, which is **fs0**. It's our ESP. That means if we list the contents of ESP, you will find the EFI directory in it. Inside that directory, you will find the directory named **redhat**, as shown in Figure 8:

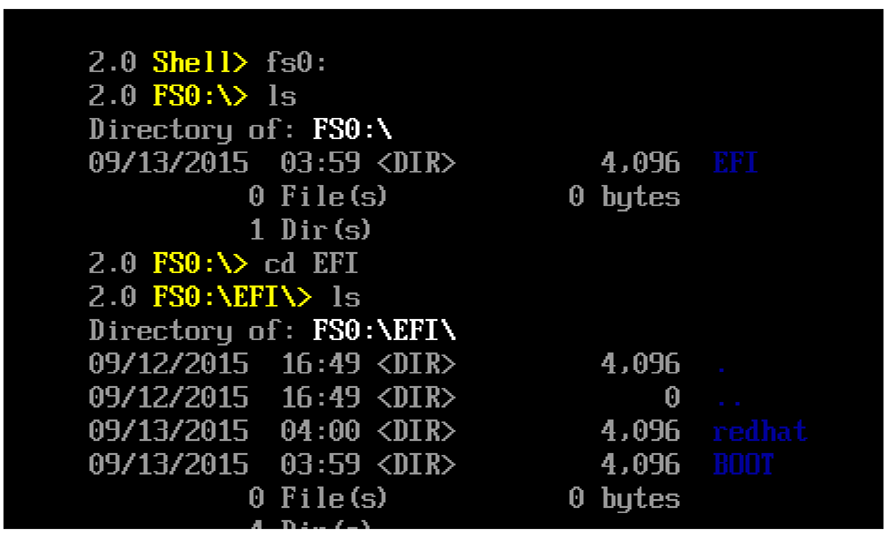


Figure 8: View the redhat directory inside the EFI filesystem

4. The **redhat** directory is where GRUB 2 has been installed, as you can see by typing **ls**. From there, you can run the **grubx64.efi** bootloader, as shown in Figure 9:

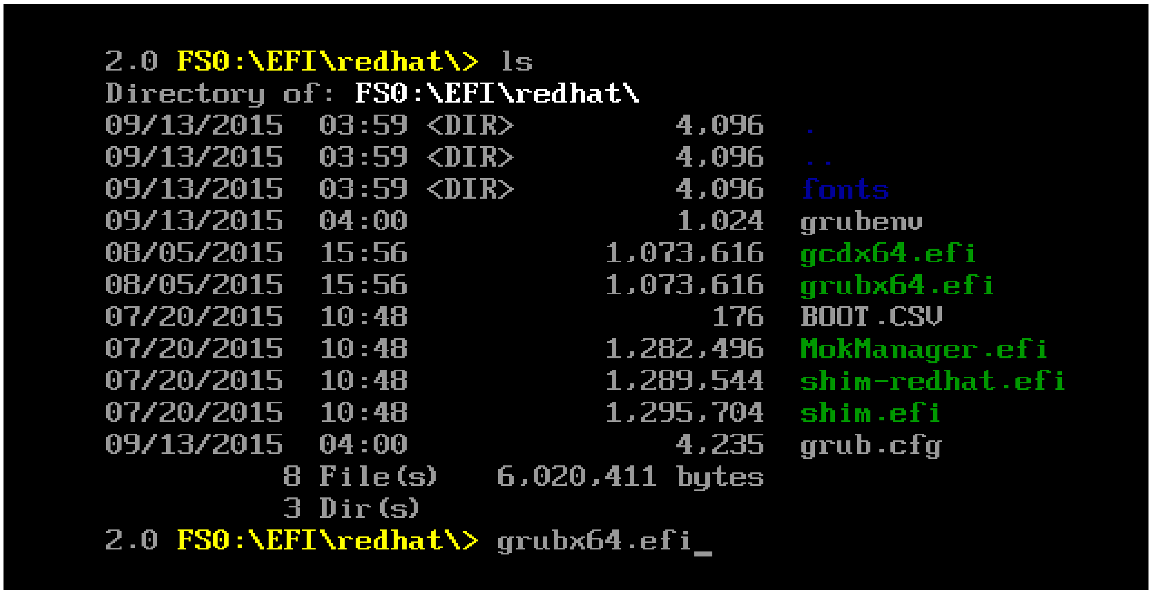


Figure 9: List GRUB 2 files and run grubx64.efi from the UEFI shell

5. When you run **grubx64.efi**, it will read the **grub.cfg** and display the contents as a splash screen, as shown in Figure 10:

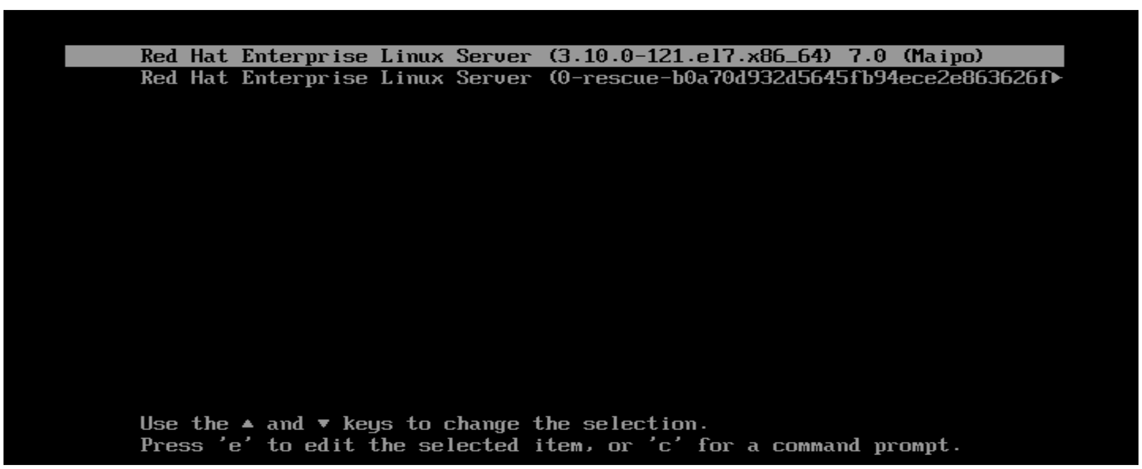


Figure 10: Display the GRUB 2 splash screenl

Understanding UEFI default boot behavior

Inside the EFI directory, there is a directory named BOOT as well. Let's look at the contents of that directory:

```
# ls /boot/efi/EFI/BOOT/
BOOTX64.EFI  fallback.efi
```

When the system starts up, UEFI firmware checks the environment variable called as *BootOrder* and tries to boot every **OS/bootloader/application/*.efi** from it until it runs out of options. Figure 11 shows the output of the efibootmgr command to list the BootOrder and the currently booted system (*BootCurrent*):

```
[root@localhost /]# efibootmgr -v
BootCurrent: 0005
BootOrder: 0005,0000,0001,0002,0003,0004
Boot0000* EFI VMware Virtual SCSI Hard Drive (0.0)      ACPI(a0341d0,0)PCI(10,0)SCSI(0,0)
Boot0001* EFI VMware Virtual SATA CDROM Drive (0.0)     ACPI(a0341d0,0)PCI(11,0)PCI(5,0)03120a00000000000000
Boot0002* EFI VMware Virtual SATA CDROM Drive (1.0)     ACPI(a0341d0,0)PCI(11,0)PCI(5,0)03120a00010000000000
Boot0003* EFI Network   ACPI(a0341d0,0)PCI(11,0)PCI(1,0)MAC(000c2927a8fd,0)
Boot0004* EFI Internal Shell (Unsupported option)       MM(b,e1a2000,e42ffff)
Boot0005* Red Hat Enterprise Linux      HD(1,800,64000,a17c521e-8435-4859-83c0-cb923874a846)File(\EFI\redhat\shi
m.efi)
```

Figure 11: Run efibootmgr to see BootOrder and BootCurrent

If every option provided by the BootOrder variable fails to boot, then UEFI firmware goes to its default boot behavior, in which it runs the **/EFI/Boot/BOOTX64.EFI** file. Now this is a default boot behavior of UEFI. Red Hat's default boot behavior is, if everything fails, then it runs **fallback.efi** from same directory. The **fallback.efi** file is a copy of **shim.efi**, with some small changes. The job of the **fallback.efi** executable is to go into UEFI firmware and set the proper values of environment variables. For example:

1. Add GRUB 2's entry in the BootOrder variable.
   BootOrder: 0005,0000,0001,0002,0003,0004
2. Make GRUB 2's entry in the appropriate Boot*XXXX* variable. By referring the 'BOOT.CSV' file which is at '/boot/efi/EFI/redhat/BOOT.CSV':

```
# cat /boot/efi/EFI/redhat/BOOT.CSV
��shim.efi,Red Hat Enterprise Linux,,This is the boot entry for Red Hat
Enterprise Linux
You will find the similar entries in 'efibootmgr -v' output which are made by
'fallback.efi' by referring the BOOT.CSV file:
Boot0005* Red Hat Enterprise Linux
HD(1,800,64000,a17c521e-8435-4859-83c0-cb923874a846)File(\EFI\redhat\shim.efi)
```

The **fallback.efi** file is not used for booting purpose. This file will try to fix the environment variables if the system is failing to boot.

Repairing/Fixing GRUB 2

This section covers what to do if GRUB 2 is not working on your system or if GRUB 2 itself is missing.

Issue 1: GRUB 2 itself is missing.

If GRUB 2 is missing, the first thing comes in mind is a **grub-install** command. This is what we used to do with GRUB legacy. But with GRUB 2 on a UEFI-based system, the **grub2-install** command is not really a good idea. The **grub2-install** command will only restore **grubx64.efi**. It will neither create a **grub.cfg** file nor fix any other ***.efi** file if they are missing. For example:

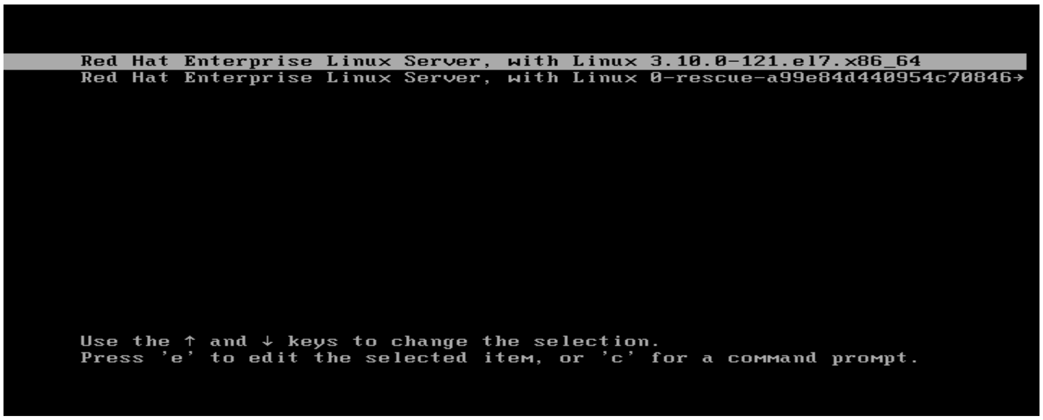
```
# ls /boot/efi/EFI/
BOOT redhat
# rm -rf /boot/efi/EFI/redhat/ boot/efi/EFI/BOOT/
# ls /boot/efi/EFI/
# grub2-install --efi-directory=/boot/efi
Installing for x86_64-efi platform.
Installation finished. No error reported.
# ls /boot/efi/EFI/
red
# ls /boot/efi/EFI/red/
grubx64.efi
```

Above, I deleted the **redhat** and **BOOT** directories from the ESP, then used the **grub2-install** command. But it does not really fix anything. If we reboot this system now it will not be able to boot. So the conclusion is that the **grub2-install** command will not be helpful on UEFI systems to fix GRUB 2. It might help on a BIOS based system though. First let’s try to understand the **grub2-install** command on a BIOS based system:

1. Remove the /boot/grub2 directory and try to restore it as follows:

```
# rm -rf /boot/grub2/
# grub2-install /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.
# ls /boot/grub2/
fonts grubenv i386-pc locale
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0.121.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0.121.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-a99e84d440954c708469fab5ac324a61
Found initrd image: /boot/initramfs-0-rescue-a99e84d440954c708469fab5ac324a61.img
done
# reboot
```

2. Above I deleted the grub2 home directory itself, which is **/boot/grub2** and I tried to fix it. The **grub2-install** command placed the missing root directory of grub2 but it did create a **grub.cfg** which we need to create with the help of **grub2-mkconfig** and after reboot grub2 came back:



3. We can also install GRUB 2 on a different disk. Below you can see that this system has another disk attached called sdb. I want to install GRUB 2 on that new disk:

```
# ls -l /dev/sd*
brw-rw----. 1 root disk 8,  0 Sep 28 05:39 /dev/sda
brw-rw----. 1 root disk 8,  1 Sep 28 05:39 /dev/sda1
brw-rw----. 1 root disk 8,  2 Sep 28 05:39 /dev/sda2
brw-rw----. 1 root disk 8, 16 Sep 28 05:39 /dev/sdb
brw-rw----. 1 root disk 8, 17 Sep 28 05:39 /dev/sdb1
# mount /dev/sdb1 /mnt/
# mkdir /mnt/boot
# grub2-install --boot-directory=/mnt/boot /dev/sdb
Installing for i386-pc platform.
Installation finished. No error reported.
# ls /mnt/boot/
grub2
# ls /mnt/boot/grub2
fonts  brubenv   i386-pc   locale
# dd if=/dev/sdb of=first.sector bs=512 count=1
1+0 records in
1+0 records outstanding512 bytes (512 B) copied, 0.00184271 s, 2.8 MB/s
# hexdump first.sector | less
```

So I needed to just use the **--boot-directory**' option. Basically, the **grub2-install** command on BIOS-based system fixes all the stages which we have seen earlier:

```
stage1      From the first 512 bytes
stage1.5    From the first 512 bytes
stage2      From grub2's root directory which is
            /boot/grub2. If you use --boot-directory, the directory
             you identify is used as the path to GRUB 2's root
```

Just to cross verify, If you open the first 512 bytes with the help of **dd** and **hexdump** commands, you will find it added the necessary data (stage1 and stage1.5) in first sector of the second hard drive (**/dev/sdb**). See Figure 11 for an example of the **hexdump** output:

```
0000000 63eb 0090 0000 0000 0000 0000 0000 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
*
0000050 0000 0000 0000 0000 0000 8000 0001 0000
0000060 0000 0000 faff 9090 c2f6 7480 f605 70c2
0000070 0274 80b2 79ea 007c 3100 8ec0 8ed8 bcd0
0000080 2000 a0fb 7c64 ff3c 0274 c288 be52 7c05
0000090 41b4 aabb cd55 5a13 7252 813d 55fb 75aa
00000a0 8337 01e1 3274 c031 4489 4004 4488 89ff
00000b0 0244 04c7 0010 8b66 5c1e 667c 5c89 6608
00000c0 1e8b 7c60 8966 0c5c 44c7 0006 b470 cd42
00000d0 7213 bb05 7000 76eb 08b4 13cd 0d73 845a
00000e0 0fd2 de83 be00 7d85 82e9 6600 b60f 88c6
00000f0 ff64 6640 4489 0f04 d1b6 e2c1 8802 88e8
0000100 40f4 4489 0f08 c2b6 e8c0 6602 0489 a166
0000110 7c60 0966 75c0 664e 5ca1 667c d231 f766
0000120 8834 31d1 66d2 74f7 3b04 0844 377d c1fe
0000130 c588 c030 e8c1 0802 88c1 5ad0 c688 00bb
0000140 8e70 31c3 b8db 0201 13cd 1e72 c38c 1e60
0000150 00b9 8e01 31db bff6 8000 c68e f3fc 1fa5
0000160 ff61 5a26 be7c 7d80 03eb 8fbe e87d 0034
0000170 94be e87d 002e 18cd feeb 5247 4255 0020
0000180 6547 6d6f 4800 7261 2064 6944 6b73 5200
0000190 6165 0064 4520 7272 726f 0a0d bb00 0001
00001a0 0eb4 10cd 3cac 7500 c3f4 0000 0000 0000
00001b0 0000 0000 0000 0000 2d79 b389 0000 2000
00001c0 0021 8a83 8208 0800 0000 f800 001f 0000
00001d0 0000 0000 0000 0000 0000 0000 0000 0000
*
00001f0 0000 0000 0000 0000 0000 0000 0000 aa55
:
```

Figure 11: Run hexdump to view boot sector contents

This is all about fixing GRUB 2 with the **grub2-install** command on a BIOS based system. But as we have seen earlier the same **grub2-install** command is not really helpful on a UEFI-based system. What could be done here, is that you can take another identical system and copy the **/boot/efi/** contents to your system by booting in rescue mode or you can install the binaries with the help of **yum** command. For example:

```
# yum install grub2-efi shim
```

The grub2-efi package provides the following files:

- fonts
- gcdx64.efi
- grubx64.efi

The shim package installs the below-mentioned binaries:

- BOOTX64.efi
- fallback.efi
- MockManager.efi
- shim.efi
- shim-redhat.efi

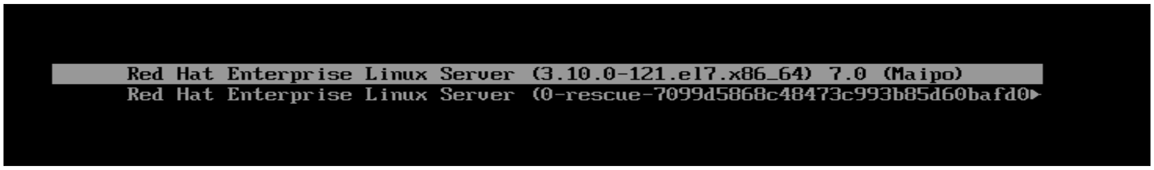
Here's how to see those files:

```
# ls /boot/efi/EFI/
BOOT redhat
# ls /boot/efi/EFI/BOOT/
BOOTX64.EFI   fallback.efi
# ls /boot/efi/EFI/redhat/
BOOT.CSV    MokManager.efi   shim.efi    shim-redhat.efi
```

You may have noticed there is no **grub.cfg** file and without **grub.cfg**, GRUB 2 will not be able to boot without manual intervention. So we need to recreate it with the help of the **grub2-mkconfig** command:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0.121.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0.121.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-7099d5868c48473c993b85d60bafd021
Found initrd image: /boot/initramfs-0-rescue-7099d5868c48473c993b85d60bafd021.img
done
# ls /boot/efi/EFI/redhat
BOOT.CSV  gcdx64.efi  grubenv     MokManager.efi   shim-redhat.efi
fonts     grub.cfg    grubx64.efi shim.efi
```

Now if we reboot system it should boot, as shown here:



Issue 2: grub2.cfg file does not contain all the entries.

Whenever the **grub2-mkconfig** command runs, it goes to the **/etc/grub.d** directory and it runs all the scripts that are available. Now if the scripts gets corrupted or misplaced or are not at their default locations, then obviously the resultant **grub.cfg** file will not have all the entries into it. In that case, we need to reinstall the grub2-tools package, which will copy the correct script files to the **/etc/grub.d** directory:

```
# rm -rf /etc/grub.d/*
# ls /etc/grub.d/
# yum reinstall grub2-tools
# ls /etc/grub.d/
00_header   10_linux      20_ppc_terminfo     40_custom   README
01_users   20_linux_xen   30_os-prober    41_custom
```

The grub2-tools package is an important package. Besides providing the above discussed scripts, the package also provides all the commands related to GRUB 2, including:

```
grub2-bios-setup       grub2-mkfont             grub2-reboot
grub2-editenv          grub2-mkimage            grub2-render-label
grub2-file             grub2-mklayout           grub2-rpm-sort
grub2-fstest           grub2-mknetdir           grub2-script-check
grub2-glue-efi         grub2-mkpasswd-pbkdf2    grub2-set-default
grub2-install          grub2-mkrelpath          grub2-setpassword
grub2-kbdcomp          grub2-mkrescue           grub2-sparc64-setup
grub2-macbless         grub2-mkstandalone       grub2-syslinux2cfg
grub2-menulst2cfg      grub2-ofpathname
grub2-mkconfig         grub2-probe
```

Tweaking GRUB 2

As we have seen, if one has multiple operating systems installed on a computer and runs the **grub2-mkconfig** command, GRUB 2 executes the scripts that are available from the

/etc/grub.d directory. Scripts including **10_linux** will find installed kernels (meaning Linux-based operating systems), **30_os-prober** will find non-Linux operating systems, such as Windows and it will add their entries into the **grub.cfg** file. This does not mean that the scripts will find every operating system from hard drive. They will try hard to find each operating system, but there is no guarantee. In case the scripts do not find a particular operating system, for example Oracle Solaris in this case, then you need to add its entry in either the **40_custom** or **41_custom** file as shown below:

1. Boot into the UEFI shell.
2. Go to fs0: -> EFI -> Oracle -> grubx64.efi and make sure Oracle Solaris is booting.
3. Reboot and select RHEL, Centos, or Fedora Linux. (I am considering RHEL in this case).
4. Open the **/etc/grub.d/40_custom** file for editing (using vi in this example) then add an entry for the operating system you want to boot (highlighted in this example):

```
# vi /etc/grub.d/40_custom
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries. Simply type
# menu entries you want to add after this comment. Be careful not to change
# the 'exec tail' line above.
menuentry "Solaris" {

set root=(hd0,gpt2)
chainloader /efi/oracle/grubx64.efi
}
```

Here is a description of the entry for Solaris:

menuentry "solaris" {

In the GRUB Legacy configuration file, it uses the word '**Title**' for displayting an operating system name. GRUB 2 uses the word '**menuentry**' instead. After the menu name, the operating system name has to be passed in quotes. After that an open curly bracket is needed. Note that the open curly bracket *has to be* on the same menuentry line otherwise it will result in an error.

set root=(hd0,gpt2)

This identifies the location of the disk partition that contains the bootable operating system (**hd0**) and the type of file system (**gpt2**).

chainloader /efi/oracle/grubx64.efi

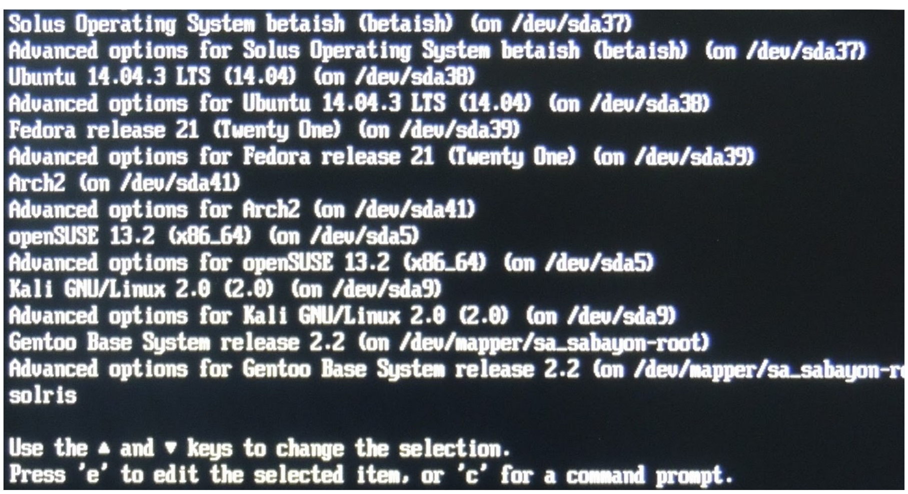
To boot any operating system we need to pass the location of the kernel and the file containing its drivers (usually the initial RAM disk) to the bootloader. In this case we do not know the Solaris filenames, but its bootloader knows the files. So the trick is let GRUB 2 call the Solaris boot loader (GRUB again). Once GRUB 2 runs the Solaris executable, it knows what to do next.

In GRUB Legacy, we used to call chainloader as **chainloader +1**. It used to go to the VBR (virtual boot record) of a particular partition to run that particular operating system's bootloader. In UEFI it's even easier. We just need to reach to the bootloader's **.efi** file in ESP.

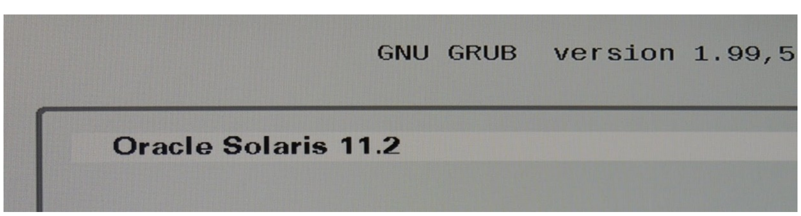
5. After the new entry has been added, rebuild the **grub.cfg** file as follows:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
Generating gurb configuration file...
Found linux image: /boot/vmlinuz-3.10.0-121.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0.121.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-7099d5868c48473c993b85d60bafd021
Found initrd image: /boot/initramfs-0-rescue-7099d5868c48473c993b85d60bafd021.img
```

7. After reboot, we can see 'solaris' entry has been added:



When we select that entry it calls/chainloads the Solaris bootloader, which is also a GRUB boot loader:



Passing parameters to user space and the init process

The topic title is somewhat confusing. We have seen so far that GRUB passes parameters to the kernel and that the moment the kernel takes control, GRUB goes away. So, in other words, when user space processes start, GRUB is not even available.

If GRUB has exited, how will it pass parameters to user space. Let me clear the confusion. Suppose you want to boot in single user mode. With the latest Linux distributions, such as RHEL 7. The ideal way is to reboot the system into single user mode, emergency mode or rescue mode with the help of systemd. For example. run either of the two following commands:

systemctl rescue

systemctl emergency

If you want to boot into single user mode at the time of the boot itself, with GRUB Legacy, you would pass 1 or s to the kernel stanza from GRUB interface. This is irrespective of GRUB 1 or GRUB 2. In GRUB, this **1** or **s** parameter is not a kernel parameter so the kernel does not understand it. However, the kernel saves it and passes it to **init**/systemd (or other initialization process) when it forks that process. The init/systemd process is a first user space process started. The **1** or **s** tells **init/systemd** to go into single user mode. In case of a GRUB 2 and the latest linux systems, single user mode can be considered as rescue mode or emergency mode. Here is an example of going into single user mode with GRUB 2:

1. Interrupt GRUB 2 when the splash screen appears:

```
      Red Hat Enterprise Linux Server (3.10.0-121.el7.x86_64) 7.0 (Maipo)
      Red Hat Enterprise Linux Server (0-rescue-7099d5868c48473c993b85d60bafd0▸
      Solaris




      Use the ▲ and ▼ keys to change the selection.
      Press 'e' to edit the selected item, or 'c' for a command prompt.
```

2. Press **e** (stands for edit), search for the kernel stanza (vmlinuz) and attach **1** at the end of it:

```
        linuxefi /vmlinuz-3.10.0-121.el7.x86_64 root=/dev/mapper/rhel-root ro \
rd.lvm.lv=rhel/root crashkernel=auto  rd.lvm.lv=rhel/swap vconsole.font=latarc\
yrheb-sun16 vconsole.keymap=us 1_
        initrdefi /initramfs-3.10.0-121.el7.x86_64.img
```

3. Press Ctrl + X to execute. It will drop into rescue or a single user mode: