LINUX VOICE TUTORIAL

GRUB 2: HEAL YOUR BOOTLOADER

MAYANK SHARMA

There are few things as irritating as a broken bootloader. Get the best out of Grub 2 and keep it shipshape.

WHY DO THIS?

- *Grub 2* is the most popular bootloader that's used by almost every Linux distribution.
- A bootloader is a vital piece of software, but they are susceptible to damage.
- *Grub 2* is an expansive and flexible boot loader that offers various customisable options.

The *Grub 2* Linux bootloader is a wonderful and versatile piece of software. While it isn't the only bootloader out there, it's the most popular and almost all the leading desktop distros use it. The job of the *Grub* bootloader is twofold. First, it displays a menu of all installed operating systems on a computer and invites you to pick one. Second, *Grub* loads the Linux kernel if you choose a Linux operating system from the boot menu.

As you can see, if you use Linux, you can't escape the bootloader. Yet it's one the least understood components inside a Linux distro. In this tutorial we'll familiarise you with some of *Grub 2*'s famed versatility and equip you with the skills to help yourself when you have a misbehaving bootloader.

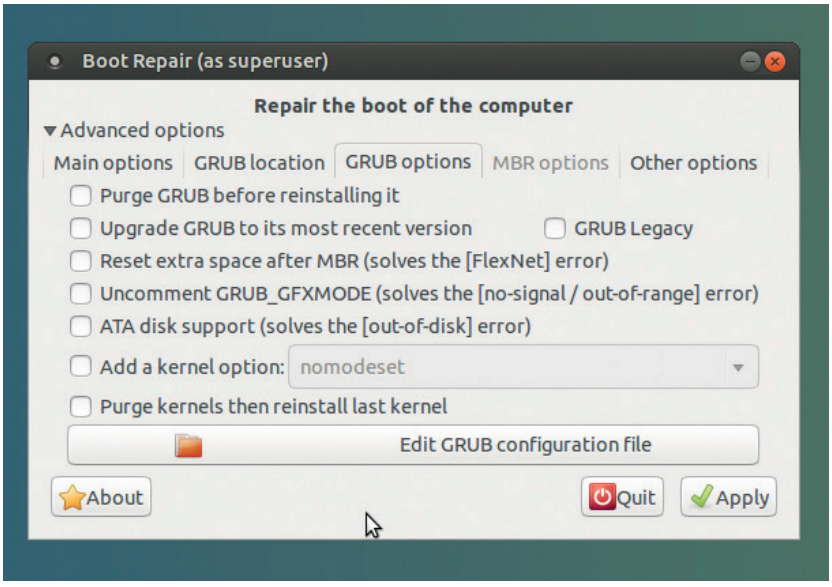
The most important parts of *Grub 2* are a bunch of text files and a couple of scripts. The first piece to know is **/etc/default/grub**. This is the text file in which you can set the general configuration variables and other characteristics of the *Grub 2* menu (see box titled "Common user settings").

"The Grub 2 Linux bootloader is a wonderful and versatile piece of software."

The other important aspect of *Grub 2* is the **/etc/grub.d** folder. All the scripts that define each menu entry are housed there. The names of these scripts must have a two-digit numeric prefix. Its purpose is to define the order in which the scripts are executed and the order of the corresponding entries when the *Grub 2* menu is built. The **00_header** file is read first, which parses the **/etc/default/grub** configuration file. Then come the entries for the Linux kernels in the **10_linux** file. This script creates one regular and one recovery menu entry for each kernel in the default **/boot** partition.

This script is followed by others for third-party apps such as **30_os-prober** and **40_custom**. The **os-prober** script creates entries for kernels and other operating systems found on other partitions. It can recognise Linux, Windows, BSD and Mac OS X installations. If your hard disk layout is too exotic for the **os-prober** script to pick up an installed distro, you can add it to the **40_custom** file (see the "Add custom entries" box).

Grub 2 does not require you to manually maintain your boot options' configuration file: instead it generates the **/boot/grub/grub.cfg** file with the



Boot Repair also lets you customise *Grub 2*'s options.

Graphical boot repair

A vast majority of *Grub 2* issues can easily be resolved with the touch of a button thanks to the *Boot Repair* app. This nifty little application has an intuitive user interface and can scan and comprehend various kinds of disk layouts and partitioning schemes, and can sniff out and correctly identify operating system installations inside them. The utility works on traditional computers with a Master Boot Record (MBR) as well as the newer UEFI computers with the UID Partition Table (GPT) layout.

The easiest way to use *Boot Repair* is to install it inside a Live Ubuntu session. Fire up an Ubuntu Live distro on a machine with a broken bootloader and install *Boot Repair* by first adding its PPA repository with the

```
sudo add-apt-repository ppa:yannubuntu/Boot Repair
```

command. Then refresh the list of repositories with

```
sudo apt-get update
```

before installing the app with

```
sudo apt-get install -y Boot Repair
```

Fire up the tool once it's installed. The app will scan your hard disk before displaying its interface, which is made up of a couple of buttons. To follow the advice of the tool, simply press the Recommended Repair button, which should fix most broken bootloaders. After it's restored your bootloader, the tool also spits out a small URL which you should note. The URL contains a detailed summary of your disks, including your partitions along with the contents of important Grub 2 files including **/etc/default/grub** and **boot/grub/grub.cfg**. If the tool hasn't been able to fix your bootloader, you can share the URL on your distro's forum boards to allow others to understand your disk layout and offer suggestions.

grub2-mkconfig command. This utility will parse the scripts in the **/etc/grub.d** directory and the **/etc/default/grub** settings file to define your setup.

Bootloader bailout

Grub 2 boot problems can leave the system in several states. The text on the display where you'd expect the bootloader menu gives an indication of the current state of the system. If the system stops booting at the **grub>** prompt, it means the *Grub 2* modules were loaded but it couldn't find the **grub.cfg** file. This is the full *Grub 2* command shell and you can do quite a bit here to help yourself. If you see the **grub rescue>** prompt, it means that the bootloader couldn't find the *Grub 2* modules nor could it find any of your boot files. However, if your screen just displays the word 'GRUB', it means the bootloader has failed to find even the most basic information that's usually contained in the Master Boot Record.

You can correct these *Grub* failures either by using a live CD or from *Grub 2*'s command shell. If you're lucky and your bootloader drops you at the **grub>** prompt, you have the power of the *Grub 2* shell at your disposal to correct any errors.

The next few commands work with both **grub>** and **grub rescue>**. The **set pager=1** command invokes the pager, which prevents text from scrolling off the screen. You can also use the **ls** command which lists all partitions that *Grub* sees, like this:

```
grub> ls
(hd0) (hd0,msdos5) (hd0,msdos6) (hd1,msdos1)
```

As you can see, the command also lists the partition table scheme along with the partitions.

You can also use the **ls** command on each partition to find your root filesystem:

```
grub> ls (hd0,5)/
lost+found/ var/ etc/ media/ bin/ initrd.gz
boot/ dev/ home/ selinux/ srv/ tmp/ vmlinuz
```

You can drop the **msdos** bit from the name of the partition. Also, if you miss the trailing slash and instead say **ls (hd0,5)** you'll get information about the partition including its filesystem type, total size, and last modification time. If you have multiple partitions, read the contents of the **/etc/issue** file with the **cat** command to identify the distro, such as **cat (hd0,5)/etc/issue**.

Assuming you find the root filesystem you're looking for inside **(hd0,5)**, make sure that it contains the **/boot/grub** directory and the Linux kernel image you wish to boot into, such as **vmlinuz-3.13.0-24-generic**. Now type the following:

```
grub> set root=(hd0,5)
grub> linux /boot/vmlinuz-3.13.0-24-generic root=/dev/sda5
grub> initrd /boot/initrd.img-3.13.0-24-generic
```

The first command points *Grub* to the partition housing the distro we wish to boot into. The second command then tells *Grub* the location of the kernel image inside the partition as well as the location of the root filesystem. The final line sets the location of the initial ramdisk file. You can use tab autocompletion to fill in the name of the kernel and the initrd, which will save you a lot of time and effort.

Once you've keyed these in, type **boot** at the next **grub>** prompt and *Grub* will boot into the specified operating system.

Things are a little different if you're at the **grub rescue>** prompt. Since the bootloader hasn't been able to find and load any of the required modules, you'll have to insert them manually:

```
grub rescue> set root=(hd0,5)
grub rescue> insmod (hd0,5)/boot/grub/normal.mod
grub rescue> normal
grub> insmod linux
```

As you can see, just like before, after we use the **ls** command to hunt down the Linux partition, we mark it with the **set** command. We then insert the **normal** module, which when activated will return us to the

Grub 2 has a command line, which you can invoke by pressing C at the bootloader menu.

```
                    GNU GRUB  version 1.99-27+deb7u2

   Minimal BASH-like line editing is supported. For the first word,
   TAB lists possible command completions. Anywhere else TAB lists
   possible device or file completions. ESC at any time exits.

grub> ls
(hd0) (hd0,msdos5) (hd0,msdos1) (hd1) (hd2) (hd3)
grub> ls (hd0,msdos5)
        Partition hd0,msdos5: Not a known filesystem - Partition start at
5928960 - Total size 360448 sectors
grub> ls (hd0,msdos1)
        Partition hd0,msdos1: Filesystem type ext2 - Last modification time
2014-09-26 14:53:40 Friday, UUID 7a53ccc5-963f-4628-80ba-3696bbc641a9 -
Partition start at 2048 - Total size 5924864 sectors
grub> ls (hd0,msdos1)/
lost+found/ var/ etc/ media/ bin/ boot/ dev/ export/ home/ initrd.img lib/
lib64/ mnt/ opt/ proc/ root/ run/ sbin/ selinux/ srv/ sys/ tmp/ usr/ vmlinuz

grub> _
```

Grub 2 and UEFI

UEFI-enabled machines (more or less, any machine sold in the last couple of years) have added another layer of complexity to debugging a broken *Grub 2* bootloader. While the procedure for restoring a *Grub 2* install on a UEFI machine isn't much different than it is on a non-UEFI machine, the newer firmware handles things differently, which results in mixed restoration results.

On a UEFI-based system, you do not install anything in the MBR. Instead you install a Linux EFI bootloader in the EFI System Partition (ESP) and set it as the EFI's default boot program using a tool such as **efibootmgr** for Linux, or **bcdedit** for Windows.

As things stand now, the *Grub 2* bootloader should be installed properly when installing any major desktop Linux distro, which will happily coexist with Windows 8. However, if you end up with a broken bootloader, you can restore the machine with a live distro. When you boot the live medium, make sure you boot it in the UEFI mode. The computer's boot menu will have two boot options for each removable drive – a vanilla option and an option tagged with UEFI. Use the latter to expose the EFI variables in **/sys/firmware/efi/**.

From the live environment, mount the root filesystem of the broken installation as mentioned in the tutorial. You'll also have to mount the ESP partition. Assuming it's **/dev/sda1**, you can mount it with

```
sudo mount /dev/sda1 /mnt/boot/efi
```

Then load the **efivars** module with **modprobe efivars** before chrooting into the installed distribution as shown in the tutorial.

Here on, if you're using Fedora, reinstall the bootloader with the

```
yum reinstall grub2-efi shim
```

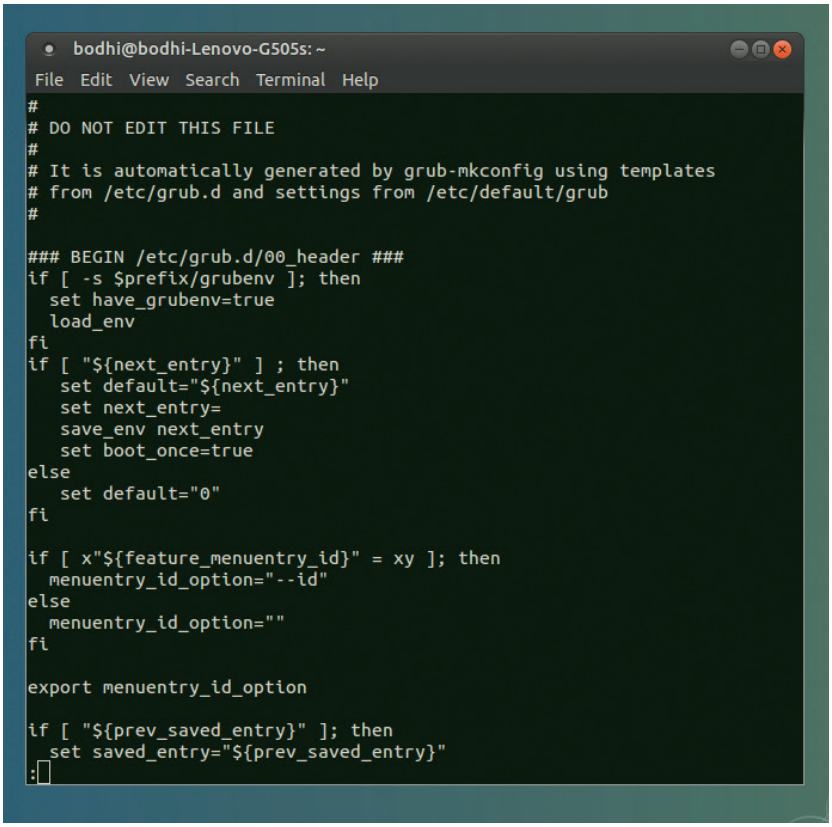
command followed by

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

to generate the new configuration file. Ubuntu users can do this with

```
apt-get install --reinstall grub-efi-amd64
```

With the bootloader in place, exit chroot, unmount all partitions and reboot to the *Grub 2* menu.



To disable a script under the **/etc/grub.d**, all you need to do is remove the executable bit, for example with **chmod -x /etc/grub.d/20_memtest86+** which will remove the 'Memory Test' option from the menu.

standard **grub>** mode. The next command then inserts the linux module in case it hasn't been loaded. Once this module has been loaded you can proceed to point the boot loader to the kernel image and initrd files just as before and round off the procedure with the **boot** command to bring up the distro.

Once you've successfully booted into the distro, don't forget to regenerate a new configuration file for *Grub* with the

```
grub-mkconfig -o /boot/grub/grub.cfg
```

command. You'll also have to install a copy of the bootloader into the MBR with the

```
sudo grub2-install /dev/sda
```

command.

Dude, where's my Grub?

The best thing about *Grub 2* is that you can reinstall it whenever you want. So if you lose the *Grub 2* bootloader, say when another OS like Windows replaces it with its own bootloader, you can restore *Grub* within a few steps with the help of a live distro. Assuming you've installed a distro on **/dev/sda5**, you can reinstall *Grub* by first creating a mount directory for the distro with

"The best thing about Grub 2 is that you can reinstall it whenever you want."

```
sudo mkdir -p /mnt/distro
```

and then mounting the partition with

```
mount /dev/sda5 /mnt/distro
```

You can then reinstall *Grub* with

```
grub2-install --root-directory=/mnt/distro /dev/sda
```

This command will rewrite the MBR information on the **/dev/sda** device, point to the current Linux installation and rewrite some *Grub 2* files such as **grubenv** and **device.map**.

Another common issue pops up on computers with multiple distros. When you install a new Linux distro, its bootloader should pick up the already installed distros. In case it doesn't, just boot into the newly installed distro and run

```
grub2-mkconfig
```

Before running the command, make sure that the root partitions of the distros missing from the boot menu are mounted. If the distro you wish to add has **/root** and **/home** on separate partitions, only mount the partition that contains **/root**, before running the **grub2-mkconfig** command.

While *Grub 2* will be able to pick most distros, trying to add a Fedora installation from within Ubuntu requires one extra step. If you've installed Fedora with its default settings, the distro's installer would have created LVM partitions. In this case, you'll first have to install the **lvm2** driver using the distro's package management system, such as with

```
sudo apt-get install lvm2
```

before *Grub 2*'s **os-prober** script can find and add Fedora to the boot menu.

Thorough fix

If the **grub2-install** command didn't work for you, and you still can't boot into Linux, you'll need to completely reinstall and reconfigure the bootloader. For this task, we'll use the venerable **chroot** utility to change the run environment from that of the live CD to the Linux install we want to recover. You can use any Linux live CD for this purpose as long as it has the **chroot** tool. However, make sure the live medium is for the same architecture as the architecture of the installation on the hard disk. So if you wish to **chroot** to a 64-bit installation you must use an amd64 live distro.

After you've booted the live distro, the first order of business is to check the partitions on the machine. Use **fdisk -l** to list all the partitions on the disk and

Common user settings

Grub 2 has lots of configuration variables. Here are some of the common ones that you're most likely to modify in the **/etc/default/grub** file. The **GRUB_DEFAULT** variable specifies the default boot entry. It will accept a numeric value such as 0, which denotes the first entry, or "saved" which will point it to the selected option from the previous boot. The **GRUB_TIMEOUT** variable specifies the delay before booting the default menu entry and the **GRUB_CMDLINE_LINUX** variable lists the parameters that are passed on the kernel command line for all Linux menu entries.

If the **GRUB_DISABLE_RECOVERY** variable is set to **true**, the recovery mode menu entries will not be generated. These entries boot the distro into single-user mode from where you can repair your system with command line tools. Also useful is the **GRUB_GFXMODE** variable, which specifies the resolution of the text shown in the menu. The variable can take any value supported by your graphics card.

make note of the partition that holds the *Grub 2* installation that you want to fix.

Let's assume we wish to restore the bootloader from the distro installed in **/dev/sda5**. Fire up a terminal and mount it with:

```
sudo mount /dev/sda5 /mnt
```

Now you'll have to bind the directories that the *Grub 2* bootloader needs access to in order to detect other operating systems:

```
$ sudo mount --bind /dev /mnt/dev
$ sudo mount --bind /dev/pts /mnt/dev/pts
$ sudo mount --bind /proc /mnt/proc
$ sudo mount --bind /sys /mnt/sys
```

We're now all set to leave the live environment and enter into the distro installed inside the **/dev/sda5** partition via **chroot**:

```
$ sudo chroot /mnt /bin/bash
```

You're now all set to install, check, and update *Grub*. Just like before, use the

```
sudo grub2-install /dev/sda
```

command to reinstall the bootloader. Since the **grub2-install** command doesn't touch the **grub.cfg** file, we'll have to create it manually with

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

That should do the trick. You now have a fresh copy of *Grub 2* with a list of all the operating systems and distros installed on your machine. Before you can restart the computer, you'll have to exit the chrooted system and unmount all the partitions in the following order:

```
$ exit
$ sudo umount /mnt/sys
$ sudo umount /mnt/proc
$ sudo umount /mnt/dev/pts
$ sudo umount /mnt/dev
$ sudo umount /mnt
```

You can now safely reboot the machine, which should be back under *Grub 2*'s control, and the bootloader under yours!

Mayank Sharma has been tinkering with Linux since the 90s and contributes to a variety of technical publications on both sides of the pond.

Add custom entries

If you wish to add an entry to the bootloader menu, you should add a boot stanza to the **40_custom** script. You can, for example, use it to display an entry to boot a Linux distro installed on a removable USB drive. Assuming your USB drive is **sdb1,** and the vmlinuz kernel image and the initrd files are under the root (/) directory, add the following to the **40_custom** file:

```
menuentry "Linux on USB" {
set root=(hd1,1)
linux /vmlinuz root=/dev/sdb1 ro quiet splash
initrd /initrd.img
}
```

For more accurate results, instead of device and partition names you can use their UUIDs, such as

```
set root=UUID=54f22dd7-eabe
```

Use

```
sudo blkid
```

to find the UUIDs of all the connected drives and partitions. You can also add entries for any distros on your disk that weren't picked up by the os-prober script, as long as you know where the distro's installed and the location of its kernel and initrd image files.