The GRUB Switch Configuration scripts

Ruediger Willenberg
Version 1.4
April 30, 2023

Table of Contents

1 Prerequisite information....3
1.1 GRUB Switch only works with GRUB2....3
1.2 Changes with GRUB version 2.06 and how to fix them....3
1.3 Conditions for script execution....4
1.4 Don’t execute with *sudo* - we’ll ask for it....4
2 CONFIGURE_GRUBswitch.sh....5
3 Support scripts for specific actions....7
3.1 _1_extract_menuentries.sh....7
3.2 _2_configure_and_generate_bootfiles.sh....7
3.3 _3_remove_generated_files.sh....9
3.4 _4_write_entries_to_usb_device.sh....9
3.5 _5_install_uptodate_hashes.sh....10
3.6 _6_remove_all_hashes.sh....10
3.7 _7_install_grubswitch.sh....11
3.8 _8_remove_grubswitch.sh....11
4 Support scripts....12
4.1 _shared_objects.h....12
4.2 update-grub.sh update-grub2.sh....12

5 Problems with specific Linux distributions....13
5.1 Linux Mint....13
5.2 Elementary OS....13
5.3 Fedora Linux....13
5.4 openSUSE....13

1 Prerequisite information

1.1 GRUB Switch only works with GRUB2

All of GRUB Switch's functionality is built on the entirely redesigned GRUB 2.x versions. It will not work with GRUB 0.9x also known as "GRUB Legacy".

1.2 Changes with GRUB version 2.06 and how to fix them

Starting with GRUB v2.06 released in 2022, calling **`update-grub/grub-mkconfig`**, which build the **`/boot/grub/grub.cfg`** script and therefore the GRUB boot menu, don't run a helper script called **`*_os-prober`** anymore; this script was responsible for adding menu entries for *Windows* and *Linux* installations on other partitions to the GRUB boot menu. This is ostensibly for security reasons. I can't speak to that, but on private multi-boot PCs, it surely is a damn nuisance.

Ubuntu 22.04 LTS and other current Linuxes have, reasonably, updated to GRUB 2.06, originally following the policy of not running **`os-prober`**. With a lot of confused people not finding Windows anymore in their boot menu, Ubuntu has, for now, implemented a workaround. **`os-prober`** is run during the first post-installation boot, and if it finds a Windows installation, it's called on subsequent runs. Since this seems unreliable and maybe prone to change, I recommend the following steps for **any installation** based on GRUB 2.06:

Make sure that **`os-prober`** is installed (it probably already is) with

```
sudo apt-get install os-prober
```

Open the *root*-owned **`/etc/default/grub`** file and add the line

```
GRUB_DISABLE_OS_PROBER=false
```

for example with

```
sudo nano /etc/default/grub
```

And afterwards run **`update-grub`** at least once with

```
sudo update-grub
```

before using our *GRUB Switch* command-line tools.

1.3 Conditions for script execution

To provide stable, reproducible behaviour, the GRUB Switch configuration scripts described in this document need to be run under the following conditions. Scripts will check these right away and quit with an error message if not fulfilled.

- Scripts need to be executed from the **`grub-switch/1_config_scripts/`** working directory as to properly read and write the the correct local and neighbouring paths and files. Make sure you're in the right place, for example with

  ```
  cd grub-switch/1_config_scripts/
  ```

- Scripts need to be called as an executable, i.e.

  ```
  ./SCRIPTNAME.sh  [parameters]
  ```

  Sourcing them with

  ```
  . SCRIPTNAME.sh
  ```
  or
  ```
  source SCRIPTNAME.sh
  ```

  would pollute the shell environment with variables and functions, and is therefore blocked.
  If your repository clone has somehow not maintained executable flags, you can restore them in the **`1_config_scripts/`** directory with

  ```
  chmod +x *.sh
  ```

1.4 Don't execute with *sudo* - we'll ask for it.

Unless you are doing all GRUB switch configuration as **root** anyway, start the scripts as a *local user* with *sudo* rights. The main configuration script asks for *sudo* right after starting, and any script requiring it will ask for it again when the *sudo* timer runs out. Starting prematurely with *sudo* will just make all generated files **root**-owned and harder to access.

If you don't know if your user has *sudo* rights, type

```
sudo whoami
```

After entering your user password, it will either reply with **root** or give you a slap on the wrist for not being a *sudoer*. In that case, google how to gain sudo rights.

2 CONFIGURE_GRUBswitch.sh

CONFIGURE_GRUBswitch.sh is the main configuration script and in most cases the only script that needs to be started explicitly by the user. It checks presence of the required directory paths and files and acts as the main text UI status display and menu for all GRUB Switch functionality.

The complete call syntax for the script is

```
./CONFIGURE_GRUBswitch.sh  -g GRUB_CFG_DIR  -s CFG_SCRIPTS_DIR  -b BLS_DIR
```

In most cases, the extra arguments are not required because of automatic detection:

The **-g** switch specifies the boot directory in which **grub.cfg** (and other GRUB files and directories required during boot). If this argument is not specified, the script will check for presence of
/boot/grub/ as well as **/boot/grub2/**
as these are the default paths in all major Linux distributions.

The script will then check for the presence of **grub.cfg** in either the specified or the automatically detected path. If no path or no **grub.cfg** can be found, the scripts quits with an error.

The **-s** switch specifies the path of the scripts that **update-grub**/**grub-mkconfig** call to produce a new **grub.cfg** from their standard output. We need this to place our own extension **99_grub_switch** there. In Ubuntu installations, this is usually **/etc/grub.d/**. However, as this location is included in the comments of **grub.cfg**, the script will by default just extract this location from there automatically. If the specified or extracted directory doesn't exist, the scripts quits with an error.

The **-b** switch points to a directory with ***.conf** files that specify boot menu entries according to the **Boot Loader Specification** (BLS). Currently only Fedora Linux, whose maintainers proposed BLS, uses this. By default, GRUB Switch already looks for such files in **/boot/loader/entries**. The switch is only required if the files are in a different path.

With all collected data, the script will display its main screen, which holds information about the existence and date of important files, and the actions that can be taken from here:

```
GRUBswitch Configuration Menu
=============================

Status of files:
----------------

Extracted list of GRUB menu entries  (../bootfiles/grubmenu_all_entries.lst):
-> last extracted at 22 Apr 2023 - 17:22:08

Generated boot files for regular flash drives (../bootfiles/boot.[1..f]):
-> last updated at 22 Apr 2023 - 23:11:24
Generated boot file for GRUBswitch USB device (../bootfiles/.entries.txt):
-> last updated at 22 Apr 2023 - 23:11:24

Permitted SWITCH.GRB file hashes  (/boot/grub/grub_switch_hashes/*):
-> last updated at 21 Apr 2023 - 19:14:41

GRUB menu config file  (/boot/grub/grub.cfg):
-> last modified at 28 Apr 2023 - 21:07:37
   contains up-to-date GRUBswitch code


ACTIONS:
--------

1 - Extract all menu entries from grub.cfg
2 - Configure GRUBswitch order and generate bootfiles and hashes
3 -   Remove generated files
4 - Write GRUBswitch bootfile to GRUBswitch USB device

5 - Install up-to-date hashes for permitted SWITCH.GRB files
6 -   Remove all hashes, no permission checking

7 - Install GRUBswitch into grub.cfg
8 -   Remove GRUBswitch from grub.cfg

q - Quit
```

All actions to be taken can be chosen by number and will call the corresponding function script (see next section).

Configuration can be concluded by pressing **q**.

3 Support scripts for specific actions

All scripts listed here should in general be called through CONFIGURE_GRUBswitch. You can call them directly, but there will be no autodetection of required directory path, they will have to be explicitly specified through arguments or the script will exit with an error.

3.1 _1_extract_menuentries.sh

This script will extract the complete list of GRUB menu entries from **`grub.cfg`** (and *Boot Loader Specification* configuration files if **`/boot/loader/entries`** exists) and write them into **`grub-switch/bootfiles/grubmenu_all_entries.lst`**. It uses regular expressions and the **`sed`** command-line tool to extract the entries' displayed names from the **`menuentry`** and **`submenu`** structures. Entries in submenus will be stored in the format

```
SUBMENU_DISPLAY_NAME>MENUENTRY_DISPLAY_NAME
```

which is the string format required later to specify this choice in the GRUB shell's **`$default`** variable for booting.

Before extracting menu entries, the script suggest to optionally re-generate the **`grub.cfg`** with **`update-grub`**. This is recommended so that it really extracts the most recent boot choices (kernel versions etc.), with the exception of Linux Mint (see section on specifics of Linux distributions).

The required syntax for directly executing the script is

```
./_1_extract_menuentries.sh  -g GRUB_CFG_DIR  -b BLS_DIR
```

where **`GRUB_CFG_DIR`** is the directory in which **`grub.cfg`** is located and **BLS_DIR** is the directory where *Boot Loader Specification* **`*.conf`** files can be found (Fedora Linux only).

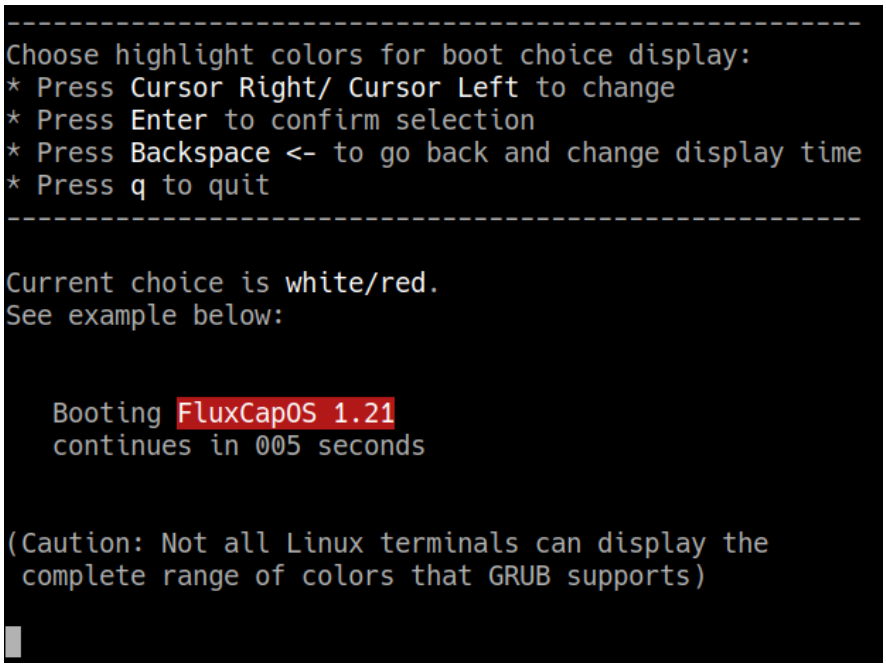
This script will require *sudo* status to read and write **`grub.cfg`**.

3.2 _2_configure_and_generate_bootfiles.sh

This action script has the most extensive functionality:

- It reads in all GRUB menu entries from **`bootfiles/grubmenu_all_entries.lst`**
- On its first screen, it enables the user to assign hexadecimal digits from **1..f** (decimal **0..15**) to entries, thereby both picking
  - the entries available for direct boot by GRUB Switch
  - the order of these entries

- On the second screen, the chosen entries are shown in the configured; the user can confirm the order or return to the previous screen to reconfigure.
- The third screen asks the user to specify for how many seconds the detected boot choice's name is displayed. This timeout allows a later GRUB Switch user to press **ESC** and return to the full GRUB menu, the minimum settings is **002** seconds so that there is always a return option to the GRUB menu.
- The fourth screen lets the user specify a foreground/background to highlight the boot choice during the specified timeout (see below).



- When all setting are confirmed, the script will generated
  - **`bootfiles/.entries.txt`** for the configuration of the custom USB device.
  - the **`bootfiles/boot.*/`** directories for the specified entry numbers, which each hold the corresponding **`SWITCH.GRB`** to copy on a flash drive that represents that choice.
  - corresponding **SHA512**-hashes for all generated **`SWITCH.GRB`** files in **`bootfiles/grub-switch-hashes/`**. This directory can later be copied to the directory that holds **`grub.cfg`** and thereby enable that every **`SWITCH.GRB`** will be checked for a matching hash before execution.
- At any stage, the script can be exit without generating new files by pressing **`q`**

The required syntax for directly executing the script is

```
./_2_configure_and_generate_bootfiles.sh
```

This script does not require *sudo*.

3.3 _3_remove_generated_files.sh

This scripts removes the generated files/directories

bootfiles/.entries.txt
bootfiles/boot.*/SWITCH.GRB
bootfiles/grub-switch-hashes/

that the previous script generated. It does not remove the list of extracted GRUB menu entries **bootfiles/grubmenu_all_entries.lst**

The required syntax for directly executing the script is

```
./_3_remove_generated_files.sh
```

This script does not require *sudo*.

3.4 _4_write_entries_to_usb_device.sh

This script's purpose is to automate the installation of a generated GRUB Switch configuration to a custom USB device through these four steps:

- The script tries to find our custom USB device through its FAT12 drive's 4-byte UUID, which is fixed in the FAT boot sector to the hexadecimal numbers **1985-1955**.
- It will try to create a temporary mount path **bootfiles/grub-switch-mount/** and mount the USB drive there.
- The configuration file **bootfiles/.entries.txt** will be copied to the mounted drive.
- The drive will be unmounted and the temporary folder will be removed. (unmounting/removing can fail if a previous step was unsuccessful)

All the steps can be also be manually accomplished by the user.

The required syntax for directly executing the script is

```
./_4_write_entries_to_usb_device.sh
```

The boot directory will not be detected automatically when calling the script directly, it has to be specified.

3.5 _5_install_uptodate_hashes.sh

This script will copy the directory **bootfiles/grub-switch-hashes/** with SHA512 hashes of **SWITCH.GRB** files to the same location as **grub.cfg**., thereby enabling hash checking for all offered **SWITCH.GRB** files on boot. On most systems, the destination folder will be **/boot/grub/** or **/boot/grub2/**.

The required syntax for directly executing the script is

```
./_5_install_uptodate_hashes.sh -g $GRUB_CFG_DIR
```

The boot directory will not be detected automatically when calling the script directly, it has to be specified.

This script will ask for *sudo* status because it is required for writing to the boot directory.

IMPORTANT NOTES:

*GRUB hash checking does not currently work with **UEFI Secure Boot** activated.*

*Hash checking also does not work with any current Fedora Linux installations and with openSUSE installations that use the default **btrfs** file format (on openSUSE, it does work on an **ext4** partition). Beware of this, as GRUB hash checking under Fedora an openSUSE(btrfs) returns with a misleading SUCCESS, implying any* SWITCH.GRB *script is ok.*

3.6 _6_remove_all_hashes.sh

This script will remove the directory of **SHA512** hashes from the GRUB boot directory. This action can both be used to remove outdated hashes as well as to disable all hash checking.

The required syntax for directly executing the script is

```
./_6_remove_all_hashes.sh -g $GRUB_CFG_DIR
```

The boot directory will not be detected automatically when calling the script directly, it has to be specified.

This script will ask for *sudo* status because it is required for removing the directory.

3.7 _7_install_grubswitch.sh

Calling this script will install the GRUB Switch functionality itself through these two steps:

- The script **`99_grub_switch`** will be copied into the directory that holds all scripts called for the regeneration of **`grub.cfg`**. On Ubuntu systems, this is **`/etc/grub.d`**.
- The scripts **`update-grub`**/**`grub-mkconfig`** will be executed to regenerate **`grub.cfg`**. As our own generator script is now included with the others, the code to detect, hash-check and source **`SWITCH.GRB`** from an attached drive will be appended to **`grub.cfg`** and executed on each boot *after* menu construction but *before* menu display.

The required syntax for directly executing the script is

```
./_7_install_grubswitch.sh -g $GRUB_CFG_DIR -s $CFG_SCRIPTS_DIR
```

The boot and generator script directories will not be detected automatically when calling the script directly, they have to be specified.

This script will ask for *sudo* status because it is required for both actions.

3.8 _8_remove_grubswitch.sh

Calling this script will fully remove the GRUB Switch functionality through these two steps:

- The script **`99_grub_switch`** will be removed from the directory that holds all scripts called for the regeneration of **`grub.cfg`**. On Ubuntu systems, this is **`/etc/grub.d`**.
- The scripts **`update-grub`**/**`grub-mkconfig`** will be executed to regenerate **`grub.cfg`**. As our own generator script is now removed, no GRUB Switch functionality will be executed.

The required syntax for directly executing the script is

```
./_8_remove_grubswitch.sh -g $GRUB_CFG_DIR -s $CFG_SCRIPTS_DIR
```

The boot and generator script directories will not be detected automatically when calling the script directly, they have to be specified.

This script will ask for *sudo* status because it is required for both actions.

4 Support scripts

4.1 _shared_objects.h

This script defines variables and helper functions used by the other scripts. It is sourced to include them into the executed scripts' environment.

4.2 update-grub.sh
update-grub2.sh

These two scripts are defined for compatibility reasons in case a Linux installation only has the **`grub-mkconfig`** script and not its **`update-grub`** wrapper.

5 Problems with specific Linux distributions

5.1 Linux Mint

Linux Mint is based on Ubuntu and uses a somewhat problematic mechanism to generate its individualized boot entries. When **`update-grub`** is called, it actually generates a **`grub.cfg`** with menu entries named *Ubuntu*. Then, every time the PC shuts down or reboots,a script is called that substitutes every mention of *Ubuntu* with *Linux Mint*.

As a consequence, you should extract menu entries right after rebooting and without calling **`update-grub`** before (which is optional). This way, the extracted entries contain *Linux Mint* and will be recognized on the next boot.

5.2 Elementary OS

Elementary OS puts its grub.cfg into a different location; this location needs to be specified when starting the GRUB Switch configuration:

```
./CONFIGURE_GRUBswitch.sh  -g /boot/efi/EFI/ubuntu/grub/
```

(Note: To heighten confusion, the regular location /boot/grub/grub.cfg is also maintained and updated by Elementary OS, however it is not actually used when GRUB is booting)

5.3 Fedora Linux

Fedora Linux defines and uses the *Boot Level Specification* to put its own boot entries into **`/boot/loader/entries`** instead of **`grub.cfg`**. GRUB Switch parses this location so no problems should result from that change.

Fedora uses entry titles that include the kernel version. Therefore, regularly update and re-extract boot choices to boot the most current kernel version with The GRUB Switch.

Lastly, checking hashes with GRUB Switch does not currently work in Fedora, and unfortunately returns a *success* value, implying the hash check recognized the file. Please do not use hash checking with Fedora!

5.4 openSUSE

openSUSE's hash checking also fails by returning *success* values everytime if it is installed with the default **btrfs** file system. If you install openSUSE on an **ext4** partition instead, hash checking works.