The GRUB Switch
General information

Ruediger Willenberg

Version 1.3
June 28, 2022

Note: The GRUB Switch only works with GRUB versions 2.x, not with the older versions also known as “GRUB Legacy”. This shouldn’t be a problem as all major Linux/Unix distributions have upgraded to GRUB 2 a decade ago.

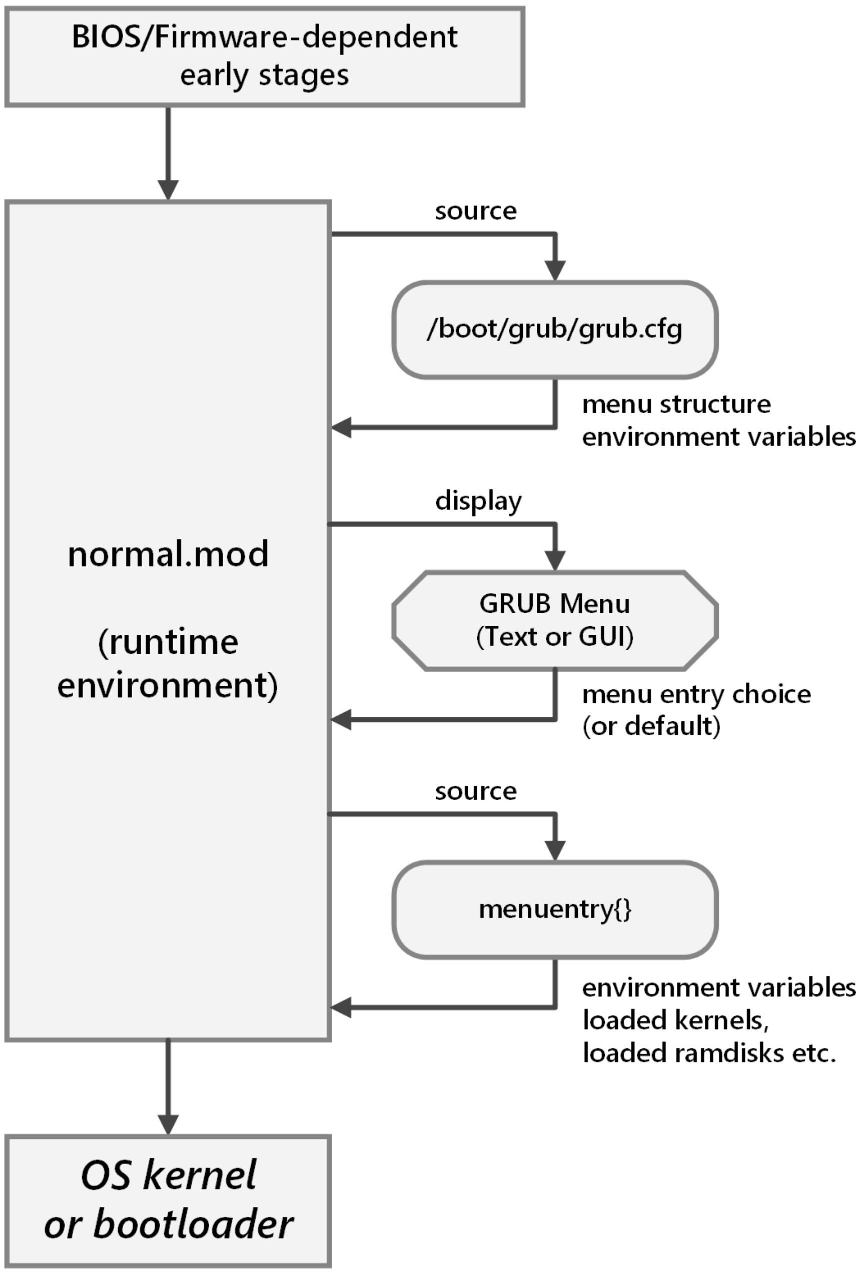
Table of Contents

1 How GRUB2 works

Disclaimer: This information is likely oversimplified and possibly inaccurate, but it should explain how what we do works.

1.1 The GRUB2 boot process

The following image gives a rough overview how GRUB2 boots one of multiple installed operating systems.



How and from where the boot process starts is dependent on the type of BIOS or (U)EFI firmware that is running on the hardware but they will all lead to executing a central GRUB binary module called **normal.mod**, which can be considered a runtime environment for a POSIX-like shell with syntax, variable handling, control structures etc. very similar to what a common Unix shell like **bash** offers. One major difference is that there is no support to write files, so GRUB cannot change persistent storage. Another difference is a large set of commands and environment variables related to how GRUB supports booting various operating systems. See the GRUB2 documentation for detailed information on all features: https://www.gnu.org/software/grub/manual/grub/grub.html

The environment will by default source a script named **`grub.cfg`** located in the GRUB installation directory. This is usually located in **`/boot/grub`** or **`/boot/grub2`** on the root drive of the Linux installation that installed GRUB.

When sourced, **`grub.cfg`** will usually set a few environment variables, for example settings related to the GRUB menu like a timeout, a default boot entry and graphics settings if a GUI menu instead of a text menu is desired.

`grub.cfg`'s other purpose is to build the structure of the menu, which involves adding the menu entries in text form as well as the actions to be taken when a specific menu entry is chosen. The central shell command to do this is called **`menuentry`**. Here is a (simplified) version of what a **`menuentry`** call could look like:

```
menuentry 'Windows 10' --class windows --class os {
     insmod part_gpt
     insmod fat

     search --no-floppy --fs-uuid --set=root  2E9D-AF3F

     chainloader /EFI/Microsoft/Boot/bootmgfw.efi
}
```

This would generate a menu entry named “Windows 10” which, when chosen, loads two additional GRUB code modules for reading GPT partition tables and FAT file systems, identifies a partition by its UUID and sets it as the Windows root partition and then starts Windows’ own proprietary bootloader.

It is important to understand that this code is not executed when the **`grub.cfg`** reaches the **`menuentry`** statement. Rather, with its curly braces, it can be understood like a C function definition, and that function is called later when the menu entry has been chosen by the user or by default.

Another important command is the **`submenu`** command which can enclose a group of **`menuentry`** blocks inside curly braces and results in a **`submenu`** grouping multiple choices in the GRUB menu. For example, Ubuntu uses this to group older kernel versions and safe mode boot options in a submenu.

When the execution of **`grub.cfg`** has finished, **normal.mod** will display the GRUB menu, based on the environment parameters for graphics, timeout and default choice that have been set in **`grub.cfg`**. Menu entries and submenus will be displayed in the order in which they have been registered in **`grub.cfg`**.

If the user chooses an entry by pressing enter or if the default is chosen by timeout, the runtime environment will execute the commands listed in the corresponding **`menuentry`** definition and will start executing the chosen OS's kernel or proprietary bootloader.

1.2 How *grub.cfg* is generated

`grub.cfg` is not designed to be edited by hand to change GRUB menu and boot behaviour, but rather to be auto-generated based on a series of Linux shell scripts. In fact, distributions like Ubuntu re-generate **`grub.cfg`** every time they update the kernel version, so any changes made directly to it by hand would be overwritten again. To understand how to influence GRUB behaviour, we need to take a look at that generation process. It is illustrated in the image on the next page.

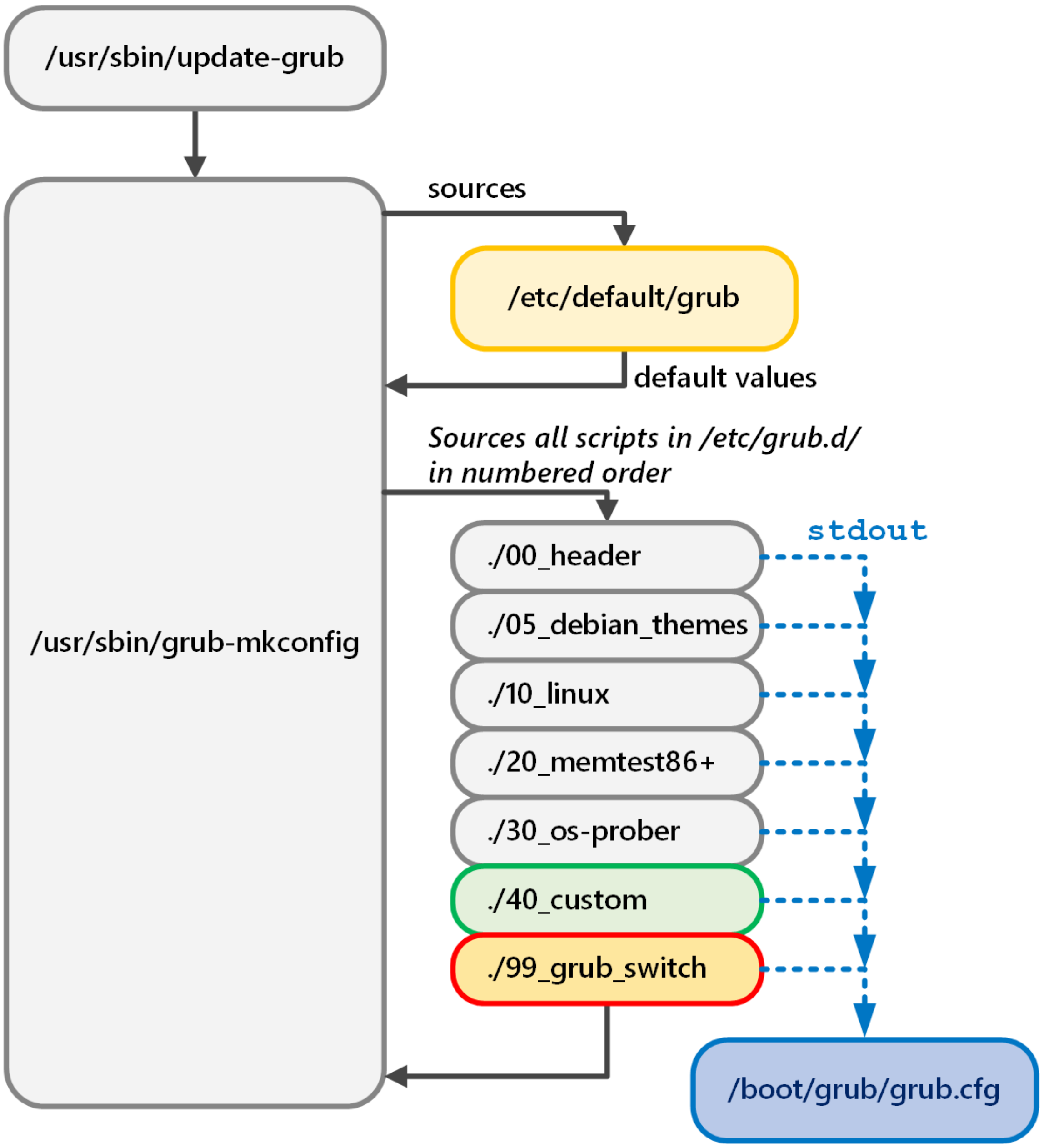
To update **`grub.cfg`**, one has to call the **`grub-mkconfig`** script. However, most distributions wrap this call into another script called **`update-grub`** so that extra call parameters can be included whenever the user calls the script. To trigger the update explicitly, call

```
sudo update-grub
```

This should usually not lead to any changes in **`grub.cfg`** if your OS installation configuration has not changed, as the script has been called with the identical data before. When **`grub-mkconfig`** executes, it first sources the file **`/etc/default/grub`**, which only holds variable definitions like these, for example:

```
GRUB_DEFAULT=0
GRUB_TIMEOUT=10
# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console
```

`/etc/default/grub` is in fact the file you are supposed to change by hand, because it will never be overwritten by others. However, the variables set here will be used to generate **`grub.cfg`**, which will then set the GRUB environment variables. Based on the example, the menu timeout is set to 10 seconds, and the first menu entry (the count starts at 0) is made the default, while a GUI menu will not be suppressed in favor of a text menu.



After sourcing **`/etc/default/grub`**, the **`grub-mkconfig`** searches the directory **`/etc/grub.d`** for shell script files starting with a number and executes all of them in numerical order (which will later influence the menu entry order). These scripts are not supposed to write directly to **`grub.cfg`**, however. Instead, their **`stdout`** output will be redirected to be written into it, therefore assuring that all outputs are concatenated in sequence, resulting in the corresponding menuentry order.

Our image shows a simplified example list of scripts that can be found in **`/etc/grub.d`**:

- **00_header** produces a few initial settings for **grub.cfg** that relate to menu appearance, like GUI or text display, timeout and default entry. For that purpose it reads the values defined in **/etc/default/grub** but also checks the for validity before setting corresponding environment variables in **grub.cfg**.
- **05_debian_themes** probably sets a few style (font, font size, background image, etc.) for GUI menu mode.
- **10_linux** probes for Linux installations, older kernel versions and safe mode boot options and generates the corresponding **menuentry** blocks for **grub.cfg**.
- **20_memtest86**+ generates a menu entry for the **memtest86+** tool that can be executed right from GRUB.
- **30_os-prober** probes for other non-Linux operating systems, most prominently Windows, and produces corresponding **menuentry** blocks

  Note: Starting with GRUB v2.06 (released 2022), **30_os-prober** *is not called anymore by default, therefore Windows might be missing in the GRUB menu. This might apply to all distributions updating to the new GRUB (e.g. Ubuntu 22.04 LTS, though it still runs* os-prober *under certain conditions). To fix this, make sure the os-prober package is installed (it likely already is)*

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

  Then, add the line

  ```
  GRUB_DISABLE_OS_PROBER=false
  ```

  to **/etc/default/grub***, for example with*

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

  and update the **grub.cfg** *again with*

  ```
  sudo update-grub
  ```

- **40_custom** is a mostly empty script in which the user can enter desired **grub.cfg** code. It will be automatically redirected to **stdout** to be concatenated.
- **99_grub_switch** is shown here somewhat prematurely for illustration purposes, as it is the script with which we add our GRUB Switch modifications. The **99_** prefix assures that this is the very last code executed in **grub.cfg**. We are providing configuration scripts that will automatically copy our script to the **/etc/grub.d** directory and call **update-grub** respectively **grub-mkconfig** so that the new code is installed in GRUB. We also provide a script to remove it again cleanly.

2 How The GRUB Switch works

2.1 Basic idea

The basic operating principle of GRUB Switch is very simple:

- A block of extra commands is added at the end of **`grub.cfg`** through the **`update-grub`**/ **`grub-mkconfig`** mechanism. This means that our commands are executed *after* the menu structure has been constructed, but *before* the menu is being displayed.
- The extra commands are searching all readable partitions for a FAT partition that holds a file called **`SWITCH.GRB`**. If that file is found, it is sourced as an extra script from **`grub.cfg`** (the execution of any random, unsafe script with that name can be avoided through checking the file's hash signature).
- The main purpose of a **`SWITCH.GRB`** script is to
  - set the GRUB environment variable **`$default`** to a specific menu entry. Because **`$default`** is not set to a *number* but to the entry's displayed *name*, the menu entry order in **`grub.cfg`** can change and the correct OS is still being chosen.
  - The environment variable **`$timeout`** is set to **`0`** and therefore the menu will not even be displayed; when execution of **`grub.cfg`** is finished, the new default choice is booted right away.
- As a result, offering a different **`SWITCH.GRB`** file can lead to a different boot choice. There are two ways to accomplish that, both of which GRUB Switch facilitates:
  - By using multiple, plain USB flash drives that hold different **`SWITCH.GRB`** files, the boot choice can be made by plugging in the corresponding drive. A **`SWITCH.GRB`** file only has a size of a few hundred bytes, so any old flash drive will do.
  - A custom USB hardware that can change the file content, based on an electrical switch. We provide a USB firmware for corresponding devices.
- Two final comments on the **`SWITCH.GRB`** files:
  - The scripts do NOT execute the complicated sequences of commands that are necessary to boot the various OS options. By setting **`$default`**, it only points to the **`grub.cfg`** **`menuentry`** block in which these actions have been defined.
  - The scripts also include a mechanism to display the chosen boot option for a defined number of seconds; during that time, pressing ESC cancels the choice and leads back to the GRUB menu with its original timeout.

2.2 GRUB Switch directory and file structure

This shows a partial file structure of the GRUB Switch repository to illustrate how GRUB Switch is used and configured:

```
grub-switch/
|
+---1_config_scripts/
|   +---CONFIGURE_GRUBswitch.sh
|   +---[ various support scripts ]
|   +---modifier_script/
|       +---99_grub_switch
|
+---2_usb_device/
|   +---[ various source and build directories and makefiles ]
|
+---3_custom_hardware/
|   +---[ project files for PCB design/manufacturing/assembly ]
|
+---bootfiles/
|   +---template
|   +---grubmenu_all_entries.lst
|   +---.entries.txt
|   +---boot.1/
|   |   +---SWITCH.GRB
|   +---boot.2/
|   |   +---SWITCH.GRB
|   +---boot.3/
|   |   +---SWITCH.GRB
|   +---[ can continue up to boot.15/ ]
|   +---grub_switch_hashes
|       +---1.sha512
|       +---2.sha512
|       +---3.sha512
|       +---[ can continue up to 15.sha512 ]
|
+---documentation/
|   +---[ various documentation and documentation sources ]
|
+---README
+---LICENSE
+---quickstart.pdf
```

The **1_config_scripts/** directory holds all bash scripts that are necessary to deploy and configure GRUB Switch. The only script that needs to be called directly is **CONFIGURE_GRUBswitch.sh**, but there are multiple support scripts that will be called by it to execute various operations. In principle, these scripts can be called separately, although there is really no need for that. They and their command-line arguments are documented in more detail in **grub-switch/documentation/1_config_scripts.pdf**.

The subdirectory **modifier_script/** holds **99_grub_switch**, which will be copied to **/etc/grub.d/** before updating **grub.cfg**, thereby activating GRUB Switch.

The **2_usb_device/** directory holds all files to build and program firmware images for custom USB devices that can offer a **SWITCH.GRB** based on hardware input. Detailed documentation can be found in **grub-switch/documentation/2_usb_device.pdf**

The **3_custom_hardware/** directory holds all project files to modify, manufacture or assemble our own PCB design for a USB device. Detailed documentation can be found in **grub-switch/documentation/3_custom_hardware.pdf**

The **bootfiles/** directory is central to generating the files for a specific GRUB Switch configuration:

The text file **template** holds almost the entire script that is placed into the **SWITCH.GRB** files; it is only prepended by the specific variable assignments for the GRUB default and the choice display.

grubmenu_all_entries.lst is a list of all GRUB menu entries by name. It is extracted from **grub.cfg** by an action available in our configuration script. In another action, our configuration can be used to pick and order the entries you need for your particular GRUB Switch setup. This action will subsequently generate a number of subdirectories and files.

.entries.txt is the first file generated by our configuration script. It is intended to be written onto the custom USB storage device if that method is used, and holds all menu entries available for boot choice in the requested order, as well as display settings for the boot choice.

The **boot.*/** subdirectories (at maximum, **boot.1/** to **boot.15/**) each hold the specific **SWITCH.GRB** corresponding to that numbered choice. They should be copied onto separate, regular flash drives. If that method is used, only the flash drive corresponding to your next boot choice should be plugged in.

The `grub_switch_hashes/` directory holds a numbered file with a SHA512 hash for each `SWITCH.GRB` file that has been placed in a `boot.*/` directory with the corresponding choice number. If `grub_switch_hashes/` is copied to `/boot/grub` (this can be accomplished with our configuration script) and GRUB Switch is installed, every `SWITCH.GRB` file found is checked against the hashes and any file that does not match will not be executed, thereby ignoring manipulated `SWITCH.GRB` files.

3 Downloading and installing GRUB Switch repository

The GRUB Switch scripts and files are designed to be installed and used under Linux. There are two ways to obtain the files:

- Via **git clone**:
  - Go to your preferred working directory
  - Clone the repository with

    ```
    git clone  https://github.com/rw-hsma-fpga/grub-switch.git
    ```

  - All GRUB Switch files are now available in the path

    ```
    ./grub-switch
    ```

- Via **download**:
  - Download to your preferred directory: https://github.com/rw-hsma-fpga/grub-switch/archive/refs/heads/master.zip
  - Unzip with GUI-tool or on the command line with

    ```
    unzip grub-switch-master.zip
    ```

  - All GRUB Switch files are now available in the path

    ```
    ./grub-switch-master
    ```

All further examples will assume you are starting in the parent directory above **`grub-switch`**

4 Steps to install and configure GRUB Switch

Almost all actions to configure your system with GRUB Switch can be accomplished through the `CONFIGURE_GRUBswitch.sh` script. For a How-To guide with screenshots, see `grub-switch/quickstart.pdf`.

1. Go to the configuration script directory with

   ```
   cd grub_switch/1_config_scripts/
   ```

   and execute the script with

   ```
   ./CONFIGURE_GRUBswitch.sh
   ```

2. Install GRUB Switch into `grub.cfg` with action $\boxed{7}$ (this only needs to be done once). You will need *super user* rights, so the script will ask you for your **sudo** password. You can use action $\boxed{8}$ to cleanly remove GRUB Switch again from the boot process.

3. Extract the current menu entries from `grub.cfg` with action $\boxed{1}$. If you want to change your GRUB Switch boot choices (or if GRUB Menu entries have been renamed), you will have to repeat steps from here.

4. By picking action $\boxed{2}$, you will be able to pick the GRUB menu entries you want to be able to boot through GRUB Switch. This includes assigning them an order with numbers; this ordering is mostly relevant for the custom USB device option with `.entries.txt` (choice by hardware switch), but the numbers also correspond to the naming of the `bootfiles/boot.*` directories and hashes.

   You will next be asked for how long the GRUB Switch choice should be displayed. A three-digit number of seconds can be adjusted. If you set 0 seconds, booting will continue instantly. However, this precludes the chance to go back to the GRUB menu by pressing the ESC key.

   You can further pick a background/foreground colour combination that highlights the name of your chosen menu entry during the display time.

   After all settings have been chosen, all the files in `bootfiles/` will be generated. They can be removed again with action $\boxed{3}$.

5. If you want any **`SWITCH.GRB`** file to be checked for authenticity, you have to install the hashes into **`/boot/grub/`** with action $\boxed{5}$.
   Note: Some improper updates of GRUB itself on systems with Secure Boot can result in the GRUB hash checking module to be rejected. In that case, GRUB Switch will yield to the GRUB menu as if the hash check itself has failed.

   One fix for this issue is to remove the hashes again, which can be done under any circumstances with action $\boxed{6}$. With careful consideration, other fixes like de-activating Secure Boot itself can be used.
6. If you are using the custom USB device solution, menu action $\boxed{4}$ can automatically mount the device, copy the generated **`bootfiles/.entries.txt`** onto it and unmount the drive again. This will again require *sudo* access.
   Of course the file can also be copied on the mounted device manually.
7. If you are using the solution with multiple regular USB drives, you have to copy the various generated **`bootfiles/boot.*/SWITCH.GRB`** files to the corresponding drives manually.
8. On the next boot, GRUB will look for a **`SWITCH.GRB`** file on all drives and, if it finds it, react with the corresponding boot action.

5 Example files

5.1 grubmenu_all_entries.lst *(extracted from grub.cfg)*

```
### All entries, extracted from current GRUB menu
Ubuntu
Advanced options for Ubuntu>Ubuntu, with Linux 5.4.0-74-generic
Advanced options for Ubuntu>Ubuntu, with Linux 5.4.0-74-generic (recovery mode)
Advanced options for Ubuntu>Ubuntu, with Linux 5.4.0-73-generic
Advanced options for Ubuntu>Ubuntu, with Linux 5.4.0-73-generic (recovery mode)
Windows Boot Manager (on /dev/nvme0n1p1)
System setup
```

5.2 .entries.txt *(generated by configuration script)*

```
#1 005
#2 white/blue
Windows Boot Manager (on /dev/nvme0n1p1)
Ubuntu
System setup










### These comments may be cut off (without harm) if the file gets too large
### (ATmega32u4 has 960 bytes of available space, ATmega16u4 has only 448 bytes)
###
### #1 specifies display time for subsequent entries
### #2 specifies display colors for subsequent entries
###
### Uncommented lines above are GRUB switch choices 1..15 (0x1..0xF)
### An empty line leads to the GRUB menu, as does choice 0
```

5.3 SWITCH.GRB *(generated by configuration script or USB device)*

***Red** part is generated specifically for a boot choice, **blue** part is from the fixed template file*

```
grubswitch_sleep_secs='005'
grubswitch_choice_color='white/blue'
grubswitch_choice='Ubuntu'
### ONLY CHANGE BELOW IF YOU KNOW WHAT YOU'RE DOING ###

# make boot default
set default="$grubswitch_choice"

# suppress menu display
set timeout_style=menu
set timeout=0

# print boot choice
clear

echo 'Press ESC for boot menu'
echo
echo -n 'Booting '

old_color="$color_normal"                        # save normal color
if [ -n "${grubswitch_choice_color}" ]; then
   set color_normal="$grubswitch_choice_color"   # highlight if defined
fi

echo $grubswitch_choice

set color_normal="$old_color"                    # restore normal color
echo -n 'continues in '

# wait for specified num of seconds
if sleep --interruptible --verbose $grubswitch_sleep_secs
then
   clear
else
   unset timeout
fi

# return to grub.cfg and boot
```

5.4 99_grub_switch *(appended to **grub.cfg**)*

```
### GRUBswitch_script v1.0 ###
### FINDING, HASH-CHECKING AND EXECUTING 'SWITCH.GRB' FILE ###

## Try to find SWITCH.GRB file in drives
## extend for loop numbers if your system might have
## more drives, including plugged-in USB drives
SWITCH_GRB_FOLDER=
for i in 0 1 2 3 4 5 6 7 8 9
do
   if [ -e (hd${i})/SWITCH.GRB ]
   then     SWITCH_GRB_FOLDER="(hd${i})"
   fi

   if [ -e (hd${i},1)/SWITCH.GRB ]
   then     SWITCH_GRB_FOLDER="(hd${i},1)"
   fi
done

## check SWITCH.GRB against hashes
if [ -n "${SWITCH_GRB_FOLDER}" ]
then
   MATCHED=true        ## if the hashes folder does not exist, don't check
   if [ -e ${prefix}/grub_switch_hashes ]
   then
      echo "Found hash folder. Continuing…"
      MATCHED=false
      for i in 0 1 2 3 4 5 6 7 8 9 a b c d e f
      do
         filename=${prefix}/grub_switch_hashes/${i}.sha512
         if [ -e $filename ]
         then
            echo -n "checking against hash #${i}: "
            if sha512sum  --check ${filename} --prefix ${SWITCH_GRB_FOLDER}
            then
               MATCHED=true
            fi
         fi
      done
      if [ false = $MATCHED ]
      then
         echo "SWITCH.GRB didn't match any of the stored hashes and won't be run."
         sleep 5
      fi
   else
      echo "Didn't find hash folder."
   fi
else
   MATCHED=false       ## if SWITCH.GRB not found, don't execute either
fi

## execute SWITCH.GRB file if it exists and did not fail hash check
if [ true = $MATCHED ]
then
      source ${SWITCH_GRB_FOLDER}/SWITCH.GRB
fi
```