# GRUB/Tips and tricks

< GRUB

## Contents

## Alternative installation methods

## Install to external USB stick

### BIOS

Assume your USB stick's first partition is FAT32 and its partition is /dev/sdy1

```
# mkdir -p /mnt/usb
# mount /dev/sdy1 /mnt/usb
# grub-install --target=i386-pc --debug --boot-directory=/mnt/usb/boot /dev/sdy
# grub-mkconfig -o /mnt/usb/boot/grub/grub.cfg
```

Optionally backup configuration files of `grub.cfg`:

```
# mkdir -p /mnt/usb/etc/default
# cp /etc/default/grub /mnt/usb/etc/default
# cp -a /etc/grub.d /mnt/usb/etc
```

```
# sync; umount /mnt/usb
```

### EFI

To have grub write its EFI image to `/boot/efi/EFI/BOOT/BOOTX64.efi`, which the boot firmware will be able to find without any UEFI boot entry, use `--removable` when you run `grub-install`.

## Install to partition or partitionless disk

**Warning:** GRUB **strongly discourages** installation to a partition boot sector or a partitionless disk as GRUB Legacy or Syslinux does. This setup is prone to breakage, especially during updates, and is **not supported** by the Arch developers.

To set up grub to a partition boot sector, to a partitionless disk (also called superfloppy) or to a floppy disk, run (using for example `/dev/sdaX` as the `/boot` partition):

```
# chattr -i /boot/grub/i386-pc/core.img
# grub-install --target=i386-pc --debug --force /dev/sdaX
# chattr +i /boot/grub/i386-pc/core.img
```

**Note:**

- `/dev/sdaX` used for example only.
- `--target=i386-pc` instructs `grub-install` to install for BIOS systems only. It is recommended to always use this option to remove ambiguity in *grub-install*.

You need to use the `--force` option to allow usage of blocklists and should not use `--grub-setup=/bin/true` (which is similar to simply generating `core.img`).

`grub-install` will give out warnings like which should give you the idea of what might go wrong with this approach:

```
/sbin/grub-setup: warn: Attempting to install GRUB to a partitionless disk or to a partition. This is a BAD idea.
/sbin/grub-setup: warn: Embedding is not possible. GRUB can only be installed in this setup by using blocklists.
                        However, blocklists are UNRELIABLE and their use is discouraged.
```

Without `--force` you may get the below error and `grub-setup` will not setup its boot code in the partition boot sector:

```
/sbin/grub-setup: error: will not proceed with blocklists
```

With `--force` you should get:

```
Installation finished. No error reported.
```

The reason why `grub-setup` does not by default allow this is because in case of partition or a partitionless disk is that GRUB relies on embedded blocklists in the partition bootsector to locate the `/boot/grub/i386-pc/core.img` file and the prefix directory `/boot/grub`. The sector locations of `core.img` may change whenever the file system in the partition is being altered (files copied, deleted etc.). For more info, see **https://bugzilla.redhat.com/show_bug.cgi?id=728742** and **https://bugzilla.redhat.com/show_bug.cgi?id=730915**.

The workaround for this is to set the immutable flag on `/boot/grub/i386-pc/core.img` (using `chattr` command as mentioned above) so that the sector locations of the `core.img` file in the disk is not altered. The immutable flag on `/boot/grub/i386-pc/core.img` needs to be set only if GRUB is installed to a partition boot sector or a partitionless disk, not in case of installation to MBR or simple generation of `core.img` without embedding any bootsector (mentioned above).

Unfortunately, the `grub.cfg` file that is created will not contain the proper UUID in order to boot, even if it reports no errors. see **https://bbs.archlinux.org/viewtopic.php?pid=1294604#p1294604**. In order to fix this issue the following commands:

```
# mount /dev/sdxY /mnt        #Your root partition.
# mount /dev/sdxZ /mnt/boot   #Your boot partition (if you have one).
# arch-chroot /mnt
# pacman -S linux
# grub-mkconfig -o /boot/grub/grub.cfg
```

## Generate core.img alone

To populate the `/boot/grub` directory and generate a `/boot/grub/i386-pc/core.img` file **without** embedding any GRUB bootsector code in the MBR, post-MBR region, or the partition bootsector, add `--grub-setup=/bin/true` to `grub-install`:

```
# grub-install --target=i386-pc --grub-setup=/bin/true --debug /dev/sda
```

**Note:**

- `/dev/sda` used for example only.

- `--target=i386-pc` instructs `grub-install` to install for BIOS systems only. It is recommended to always use this option to remove ambiguity in grub-install.

You can then chainload GRUB's `core.img` from GRUB Legacy or syslinux as a Linux kernel or as a multiboot kernel (see also **Syslinux#Chainloading**).

## GUI configuration tools

Following package may be installed:

- **grub-customizer** — GTK+ customizer for GRUB or BURG

  **https://launchpad.net/grub-customizer** || **`grub-customizer (https://aur.archlinux.org/packages/grub-customizer/)`**AUR

- **grub2-editor-frameworks** — Unofficial KF5 port of grub2-editor

  **https://github.com/maz-1/grub2-editor** || **`grub2-editor-frameworks (https://aur.archlinux.org/packages/grub2-editor-frameworks/)`**AUR

- **startupmanager** — GUI app for changing the settings of GRUB Legacy, GRUB, Usplash and Splashy (**abandonned (https://launchpad.net/startup-manager/+announcement/8300)**)

**http://sourceforge.net/projects/startup-manager/** || **startupmanager (https://aur.archlinux.org/packages/startupmanager/)**AUR

## Visual configuration

In GRUB it is possible, by default, to change the look of the menu. Make sure to initialize, if not done already, GRUB graphical terminal, gfxterm, with proper video mode, gfxmode, in GRUB. This can be seen in the section **GRUB#"No suitable mode found" error**. This video mode is passed by GRUB to the linux kernel via 'gfxpayload' so any visual configurations need this mode in order to be in effect.

### Setting the framebuffer resolution

GRUB can set the framebuffer for both GRUB itself and the kernel. The old `vga=` way is deprecated. The preferred method is editing `/etc/default/grub` as the following sample:

```
GRUB_GFXMODE=1024x768x32
GRUB_GFXPAYLOAD_LINUX=keep
```

Multiple resolutions can be specified, including the default `auto`, so it is recommended that you edit the line to resemble `GRUB_GFXMODE=<desired resolution>,<fallback such as 1024x768>,auto`. For more information, refer to **the GRUB gfxmode documentation (https://www.gnu.org/software/g** rub/manual/grub/html_node/gfxmode.html). The gfxpayload (https://www.gnu.org/software/grub/manual/html_node/gfxpayload.html#gfxpayload) property will make sure the kernel keeps the resolution.

**Note:**

- Only the modes supported by the graphics card via **VESA BIOS Extensions** can be used. To view the list of supported modes, install `hwinfo` (https://www.archlinux.org/packages/?name=hwinfo) and run `hwinfo --framebuffer` as root. Alternatively, enter the GRUB command line and run the command `videoinfo`.
- Earlier versions of the **NVIDIA** proprietary driver (tested with GeForce GTX 970, driver: nvidia 370) accepts `GRUB_GFXMODE` in format `<width>x<height>-<depth>` (e.g. `1920x1200-24`, but not `1920x1200x24`). This does not appear to apply to newer cards and drivers. Pascal cards with more recent drivers (tested with GeForce GTX 1060 and nvidia 381.22) will not work with the suggested format and attempting to use it results in serious issues, including but not limited to system crashes and hard locks. The current driver and cards are best configured with `GRUB_GFXMODE` in the standard `<width>x<height>x<depth>` format.
- Make sure to run `grub-mkconfig -o /boot/grub/grub.cfg` after making changes.

If this method does not work for you, the deprecated `vga=` method will still work. Just add it next to the `"GRUB_CMDLINE_LINUX_DEFAULT="` line in `/etc/default/grub` for example: `"GRUB_CMDLINE_LINUX_DEFAULT="quiet splash vga=792"` will give you a `1024x768` resolution.

## 915resolution hack

Sometimes for Intel graphic adapters neither `# hwinfo --framebuffer` nor `videoinfo` will show you the desired resolution. In this case you can use the `915resolution` hack. This hack will temporarily modify video BIOS and add needed resolution. See **915resolution's home page (http://915resolution.mango-lang.org/)**. The package can be found here: **`915resolution (https://aur.archlinux.org/packages/915resolution/)`** AUR

First you need to find a video mode which will be modified later. For that we need the GRUB command shell:

```
sh:grub> 915resolution -l
```

```
Intel 800/900 Series VBIOS Hack : version 0.5.3
[...]
Mode 30 : 640x480, 8 bits/pixel
[...]
```

Next, we overwrite the `Mode 30` with `1440x900` resolution:

```
/etc/grub.d/00_header
```

```
[...]
915resolution 30 1440 900  # Inserted line
set gfxmode=${GRUB_GFXMODE}
[...]
```

Lastly we need to set `GRUB_GFXMODE` as described earlier, regenerate `grub.cfg` and reboot to test changes.

## Background image and bitmap fonts

GRUB comes with support for background images and bitmap fonts in `pf2` format. The unifont font is included in the **grub (https://www.archlinux.org/packages/?name=grub)** package under the filename `unicode.pf2`, or, as only ASCII characters under the name `ascii.pf2`.

Image formats supported include tga, png and jpeg, providing the correct modules are loaded. The maximum supported resolution depends on your hardware.

Make sure you have set up the proper **framebuffer resolution**.

Edit `/etc/default/grub` like this:

```
GRUB_BACKGROUND="/boot/grub/myimage"
#GRUB_THEME="/path/to/gfxtheme"
GRUB_FONT="/path/to/font.pf2"
```

**Note:** If you have installed GRUB on a separate partition, `/boot/grub/myimage` automatically becomes `/grub/myimage` in `grub.cfg`.

Re-generate `grub.cfg` to apply the changes. If adding the splash image was successful, the user will see `"Found background image..."` in the terminal as the command is executed. If this phrase is not seen, the image information was probably not incorporated into the `grub.cfg` file.

If the image is not displayed, check:

- The path and the filename in `/etc/default/grub` are correct
- The image is of the proper size and format (tga, png, 8-bit jpg)
- The image was saved in the RGB mode, and is not indexed
- The console mode is not enabled in `/etc/default/grub`
- The command `grub-mkconfig` must be executed to place the background image information into the `/boot/grub/grub.cfg` file
- The `grub-mkconfig` scripts won't quote the file name in `grub.cfg` so make sure it does not contain spaces

## Theme

Here is an example for configuring Starfield theme which was included in GRUB package.

Edit `/etc/default/grub`

```
GRUB_THEME="/usr/share/grub/themes/starfield/theme.txt"
```

Re-generate `grub.cfg` to apply the changes. If configuring the theme was successful, you will see `Found theme: /usr/share/grub/themes/starfield/theme.txt` in the terminal.

Your splash image will usually not be displayed when using a theme.

## Menu colors

You can set the menu colors in GRUB. The available colors for GRUB can be found in the GRUB Manual (https://www.gnu.org/software/grub/manual/html_node/Theme-file-format.html). Here is an example:

Edit `/etc/default/grub`:

```
GRUB_COLOR_NORMAL="light-blue/black"
GRUB_COLOR_HIGHLIGHT="light-cyan/blue"
```

## Hidden menu

One of the unique features of GRUB is hiding/skipping the menu and showing it by holding `Esc` when needed. You can also adjust whether you want to see the timeout counter.

Edit `/etc/default/grub` as you wish. Here are the lines you need to add to enable this feature, the timeout has been set to five seconds and to be shown to the user:

```
GRUB_TIMEOUT=5
GRUB_TIMEOUT_STYLE='countdown'
```

GRUB_TIMEOUT is how many seconds before displaying menu.

### Disable framebuffer

Users who use NVIDIA proprietary driver might wish to disable GRUB's framebuffer as it can cause problems with the binary driver.

To disable framebuffer, edit `/etc/default/grub` and uncomment the following line:

```
GRUB_TERMINAL_OUTPUT=console
```

Another option if you want to keep the framebuffer in GRUB is to revert to text mode just before starting the kernel. To do that modify the variable in `/etc/default/grub`:

```
GRUB_GFXPAYLOAD_LINUX=text
```

## Booting ISO9660 image file directly via GRUB

GRUB supports booting from ISO images directly via loopback devices, see **Multiboot USB drive#Using GRUB and loopback devices** for examples.

## Persistent block device naming

One naming scheme for **Persistent block device naming** is the use of globally unique UUIDs to detect partitions instead of the "old" `/dev/sd*`. Advantages are covered up in the above linked article.

Persistent naming via file system UUIDs are used by default in GRUB.

**Note:** The `/boot/grub.cfg` file needs regeneration with the new UUID in `/etc/default/grub` every time a relevant file system is resized or recreated. Remember this when modifying partitions & file systems with a Live-CD.

Whether to use UUIDs is controlled by an option in `/etc/default/grub`:

```
GRUB_DISABLE_LINUX_UUID=true
```

## Using labels

It is possible to use labels, human-readable strings attached to file systems, by using the `--label` option to `search`. First of all, label your existing partition:

```
# tune2fs -L LABEL PARTITION
```

Then, add an entry using labels. An example of this:

```
menuentry "Arch Linux, session texte" {
  search --label --set=root archroot
  linux /boot/vmlinuz-linux root=/dev/disk/by-label/archroot ro
  initrd /boot/initramfs-linux.img
}
```

## Password protection of GRUB menu

**Warning:** If someone has physical access to your machine and is able to boot a live USB/disk (*i.e.*, BIOS allows booting from an external disk), it's fairly trivial for one to modify GRUB configuration files to bypass this if `/boot` resides on an unencrypted partition. See **GRUB#Encryption** and **Security#Disk encryption**.

If you want to secure GRUB so it is not possible for anyone to change boot parameters or use the command line, you can add a user/password combination to GRUB's configuration files. To do this, run the command `grub-mkpasswd-pbkdf2`. Enter a password and confirm it:

```
grub-mkpasswd-pbkdf2
```

```
[...]
Your PBKDF2 is grub.pbkdf2.sha512.10000.C8ABD3E93C4DFC83138B0C7A3D719BC650E6234310DA069E6FDB0DD4156313DA3D0D9BFFC2846C21D5A2DDA515114CF6378F8A064C94
198D0618E70D23717E82.509BFA8A4217EAD0B33C87432524C0B6B64B34FBAD22D3E6E6874D9B101996C5F98AB1746FE7C7199147ECF4ABD8661C222EEEDB7D14A843261FFF2C07B1269
A
```

Then, add the following to `/etc/grub.d/40_custom`:

```
/etc/grub.d/40_custom

set superusers="username"
password_pbkdf2 username <password>
```

where `<password>` is the string generated by `grub-mkpasswd_pbkdf2`.

Regenerate your configuration file. Your GRUB command line, boot parameters and all boot entries are now protected.

This can be relaxed and further customized with more users as described in the "Security" part of **the GRUB manual (https://www.gnu.org/software/grub/manual/grub.html#Security)**.

## Password protection of GRUB edit and console options only

Adding `--unrestricted` to a menu entry will allow any user to boot the OS while preventing the user from editing the entry and preventing access to the grub command console. Only a superuser or users specified with the `--user` switch will be able to edit the menu entry.

```
/boot/grub/grub.cfg

menuentry 'Arch Linux' --unrestricted --class arch --class gnu-linux --class os ...
```

## Hide GRUB unless the Shift key is held down

In order to achieve the fastest possible boot, instead of having GRUB wait for a timeout, it is possible for GRUB to hide the menu, unless the `Shift` key is held down during GRUB's start-up.

In order to achieve this, you should add the following line to `/etc/default/grub`:

```
GRUB_FORCE_HIDDEN_MENU="true"
```

Then create the file in **[1] (https://gist.githubusercontent.com/anonymous/8eb2019db2e278ba99be/raw/257f15100fd46aeeb8e33a7629b209d0a14b9975/gistfile1.sh)**, make it exectuable, and regenerate the grub configuration:

```
# chmod a+x /etc/grub.d/31_hold_shift
# grub-mkconfig -o /boot/grub/grub.cfg
```

**Note:** This setup uses keystatus to detect keypress event so it may not work on some machines.

## Combining the use of UUIDs and basic scripting

If you like the idea of using UUIDs to avoid unreliable BIOS mappings or are struggling with GRUB's syntax, here is an example boot menu item that uses UUIDs and a small script to direct GRUB to the proper disk partitions for your system. All you need to do is replace the UUIDs in the sample with the correct UUIDs for your system. The example applies to a system with a boot and root partition. You will obviously need to modify the GRUB configuration if you have additional partitions:

```
menuentry "Arch Linux 64" {
    # Set the UUIDs for your boot and root partition respectively
    set the_boot_uuid=ece0448f-bb08-486d-9864-ac3271bd8d07
    set the_root_uuid=c55da16f-e2af-4603-9e0b-03f5f565ec4a

    # (Note: This may be the same as your boot partition)

    # Get the boot/root devices and set them in the root and grub_boot variables
    search --fs-uuid $the_root_uuid --set=root
    search --fs-uuid $the_boot_uuid --set=grub_boot

    # Check to see if boot and root are equal.
    # If they are, then append /boot to $grub_boot (Since $grub_boot is actually the root partition)
    if [ $the_boot_uuid == $the_root_uuid ] ; then
        set grub_boot=($grub_boot)/boot
    else
        set grub_boot=($grub_boot)
    fi

    # $grub_boot now points to the correct location, so the following will properly find the kernel and initrd
    linux $grub_boot/vmlinuz-linux root=/dev/disk/by-uuid/$the_root_uuid ro
    initrd $grub_boot/initramfs-linux.img
}
```

## Manually creating grub.cfg

A basic GRUB config file uses the following options:

- `(hdX,Y)` is the partition *Y* on disk *X*, partition numbers starting at 1, disk numbers starting at 0
- `set default=N` is the default boot entry that is chosen after timeout for user action
- `set timeout=M` is the time *M* to wait in seconds for a user selection before default is booted
- `menuentry "title" {entry options}` is a boot entry titled `title`
- `set root=(hdX,Y)` sets the boot partition, where the kernel and GRUB modules are stored (boot need not be a separate partition, and may simply be a directory under the "root" partition ( `/` )

## Multiple entries

### Disable submenu

If you have multiple kernels installed, say linux and linux-lts, by default `grub-mkconfig` groups them in a submenu. If you do not like this behaviour you can go back to one single menu by adding the following line to `/etc/default/grub`:

```
GRUB_DISABLE_SUBMENU=y
```

### Recall previous entry

GRUB can remember the last entry you booted from and use this as the default entry to boot from next time. This is useful if you have multiple kernels (i.e., the current Arch one and the LTS kernel as a fallback option) or operating systems. To do this, edit `/etc/default/grub` and change the value of `GRUB_DEFAULT`:

```
GRUB_DEFAULT=saved
```

This ensures that GRUB will default to the saved entry. To enable saving the selected entry, add the following line to `/etc/default/grub`:

```
GRUB_SAVEDEFAULT=true
```

This will only work if /boot is not a btrfs, because grub cannot write to btrfs. But it will generate a misleading error message: "sparse file not allowed. Press any key to continue.".

**Note:** Manually added menu items, e.g. Windows in `/etc/grub.d/40_custom` or `/boot/grub/custom.cfg`, will need `savedefault` added.

## Changing the default menu entry

To change the default selected entry, edit `/etc/default/grub` and change the value of `GRUB_DEFAULT`:

Using menu titles :

```
GRUB_DEFAULT='Advanced options for Arch Linux>Arch Linux, with Linux linux'
```

Using numbers :

```
GRUB_DEFAULT="1>2"
```

Grub identifies entries in generated menu counted from zero. That means 0 for the first entry which is the default value, 1 for the second and so on. Main and submenu entries are separated by a `>`.

The example above boots the third entry from the main menu 'Advanced options for Arch Linux'.

## Boot non-default entry only once

The command `grub-reboot` is very helpful to boot another entry than the default only once. GRUB loads the entry passed in the first command line argument, when the system is rebooted the next time. Most importantly GRUB returns to loading the default entry for all future booting. Changing the configuration file or selecting an entry in the GRUB menu is not necessary.

**Note:** This requires `GRUB_DEFAULT=saved` in `/etc/default/grub` (and then regenerating `grub.cfg`) or, in case of hand-made `grub.cfg`, the line `set default="${saved_entry}"`.

## Play a tune

You can play a tune through the PC-speaker while booting by modifying the variable `GRUB_INIT_TUNE`. For example, to play Berlioz's extract from Sabbath Night of Symphonie Fantastique you can add the following: (bassoon part):

```
GRUB_INIT_TUNE="312 262 3 247 3 262 3 220 3 247 3 196 3 220 3 220 3 262 3
262 3 294 3 262 3 247 3 220 3 196 3 247 3 262 3 247 5 220 1 220 5"
```

For information on this, you can look at `info grub -n play`.

## Manual configuration of core image for early boot

If you require a special keymap or other complex steps that GRUB is not able to configure automatically in order to make `/boot` available to the GRUB environment, you can generate a core image yourself. On UEFI systems, the core image is the `grubx64.efi` file that is loaded by the firmware on boot. Building your own core image will allow you to embed any modules required for very early boot, as well as a configuration script to bootstrap GRUB.

Firstly, taking as an example a requirement for the `dvorak` keymap embedded in early-boot in order to enter a password for a crypted `/boot` on a UEFI system:

Determine from the generated `/boot/grub/grub.cfg` file what modules are required in order to mount the crypted `/boot`. For instance, under your `menuentry` you should see lines similar to:

```
insmod diskfilter cryptodisk luks gcry_rijndael gcry_rijndael gcry_sha256
insmod ext2
cryptomount -u 1234abcdef1234abcdef1234abcdef
set root='cryptouuid/1234abcdef1234abcdef1234abcdef'
```

Take note of all of those modules: they'll need to be included in the core image. Now, create a tarball containing your keymap. This will be bundled in the core image as a memdisk:

```
# ckbcomp dvorak | grub-mklayout > dvorak.gkb
# tar cf memdisk.tar dvorak.gkb
```

Now create a configuration file to be used in the GRUB core image. This is in the same format as your regular grub config, but need contain only a few lines to find and load the main config file on the `/boot` partition:

```
early-grub.cfg
```

```
root=(memdisk)
prefix=($root)/

terminal_input at_keyboard
keymap /dvorak.gkb
```

```
cryptomount -u 1234abcdef1234abcdef1234abcdef
set root='cryptouuid/1234abcdef1234abcdef1234abcdef'
set prefix=($root)/grub

configfile grub.cfg
```

Finally, generate the core image, listing all of the modules determined to be required in the generated `grub.cfg`, along with any modules used in the `early-grub.cfg` script. The example above needs `memdisk`, `tar`, `at_keyboard`, `keylayouts` and `configfile`.

```
# grub-mkimage -c early-grub.cfg -o grubx64.efi -O x86_64-efi -m memdisk.tar diskfilter cryptodisk luks gcry_rijndael gcry_sha256 ext2 memdisk tar a
t_keyboard keylayouts configfile
```

The generated EFI core image can now be used in the same way as the image that is generated automatically by `grub-install`: place it in your EFI partition and enable it with `efibootmgr`, or configure as appropriate for your system firmware.

## UEFI further reading

Below is other relevant information regarding installing Arch via UEFI.

### Alternative install method

Usually, GRUB keeps all files, including configuration files, in `/boot`, regardless of where the EFI System Partition is mounted.

If you want to keep these files inside the EFI System Partition itself, add `--boot-directory=esp` to the grub-install command:

```
# grub-install --target=x86_64-efi --efi-directory=esp --bootloader-id=grub --boot-directory=esp --debug
```

This puts all GRUB files in `esp/grub`, instead of in `/boot/grub`. When using this method, make sure you have *grub-mkconfig* put the configuration file in the same place:

```
# grub-mkconfig -o esp/grub/grub.cfg
```

Configuration is otherwise the same.

## UEFI firmware workaround

Some UEFI firmware requires that the bootable `.efi` stub have a specific name and be placed in a specific location: `esp/EFI/boot/bootx64.efi` (where `esp` is the UEFI partition mountpoint). Failure to do so in such instances will result in an unbootable installation. Fortunately, this will not cause any problems with other firmware that does not require this.

To do so, first create the necessary directory, and then copy across the grub `.efi` stub, renaming it in the process:

```
# mkdir esp/EFI/boot
# cp esp/EFI/grub_uefi/grubx64.efi esp/EFI/boot/bootx64.efi
```

## Create a GRUB entry in the firmware boot manager

`grub-install` automatically tries to create a menu entry in the boot manager. If it does not, then see **UEFI#efibootmgr** for instructions to use `efibootmgr` to create a menu entry. However, the problem is likely to be that you have not booted your CD/USB in UEFI mode, as in **UEFI#Create UEFI bootable USB from ISO**.

## GRUB standalone

This section assumes you are creating a standalone GRUB for x86_64 systems (x86_64-efi). For 32-bit (IA32) EFI systems, replace `x86_64-efi` with `i386-efi` where appropriate.

It is possible to create a `grubx64_standalone.efi` application which has all the modules embedded in a tar archive within the UEFI application, thus removing the need to have a separate directory populated with all of the GRUB UEFI modules and other related files. This is done using the `grub-mkstandalone` command (included in **grub (https://www.archlinux.org/packages/?name=grub)**) as follows:

```
# echo 'configfile ${cmdpath}/grub.cfg' > /tmp/grub.cfg
# grub-mkstandalone -d /usr/lib/grub/x86_64-efi/ -O x86_64-efi --modules="part_gpt part_msdos" --locales="en@quot" --themes="" -o "esp/EFI/grub/grub
x64_standalone.efi" "boot/grub/grub.cfg=/tmp/grub.cfg" -v
```

Then copy the GRUB config file to `esp/EFI/grub/grub.cfg` and create a UEFI Boot Manager entry for `esp/EFI/grub/grubx64_standalone.efi` using **efibootmgr**.

**Note:** The option `--modules="part_gpt part_msdos"` (with the quotes) is necessary for the `${cmdpath}` feature to work properly.

**Warning:** You may find that the `grub.cfg` file is not loaded due to `${cmdpath}` missing a slash (i.e. `(hd1,msdos2)EFI/Boot` instead of `(hd1,msdos2)/EFI/Boot`) and so you are dropped into a GRUB shell. If this happens determine what `${cmdpath}` is set to (`echo ${cmdpath}`) and then load the config file manually (e.g. `configfile (hd1,msdos2)/EFI/Boot/grub.cfg`).

## Technical information

The GRUB EFI file always expects its config file to be at `${prefix}/grub.cfg`. However in the standalone GRUB EFI file, the `${prefix}` is located inside a tar archive and embedded inside the standalone GRUB EFI file itself (inside the GRUB environment, it is denoted by `"(memdisk)"`, without quotes). This tar archive contains all the files that would be stored normally at `/boot/grub` in case of a normal GRUB EFI install.

Due to this embedding of `/boot/grub` contents inside the standalone image itself, it does not rely on actual (external) `/boot/grub` for anything. Thus in case of standalone GRUB EFI file `${prefix}==(memdisk)/boot/grub` and the standalone GRUB EFI file reads expects the config file to be at `${prefix}/grub.cfg==(memdisk)/boot/grub/grub.cfg`.

Hence to make sure the standalone GRUB EFI file reads the external `grub.cfg` located in the same directory as the EFI file (inside the GRUB environment, it is denoted by `${cmdpath}`), we create a simple `/tmp/grub.cfg` which instructs GRUB to use `${cmdpath}/grub.cfg` as its config (`configfile ${cmdpath}/grub.cfg` command in `(memdisk)/boot/grub/grub.cfg`). We then instruct grub-mkstandalone to copy this `/tmp/grub.cfg` file to `${prefix}/grub.cfg` (which is actually `(memdisk)/boot/grub/grub.cfg`) using the option `"boot/grub/grub.cfg=/tmp/grub.cfg"`.

This way, the standalone GRUB EFI file and actual `grub.cfg` can be stored in any directory inside the EFI System Partition (as long as they are in the same directory), thus making them portable.

Retrieved from "https://wiki.archlinux.org/index.php?title=GRUB/Tips_and_tricks&oldid=506990"