



DUSTARCH

DustVoice's Arch Linux from scratch

David Holland

September 22, 2022

Contents

1	Inside the archiso	
1.1	Syncing up pacman	
1.1.1	Official repositories	
1.1.2	AUR	
1.1.3	Software categories	
1.1.4	Software installation	
1.1.4.1	Example section	
1.2	Formatting the drive	
1.2.1	The standard way	
1.2.2	Full system encryption	
1.2.2.1	EFI System partition	
1.2.2.2	LUKS	
1.2.2.3	LVM	
1.2.2.4	Format & mount	
1.2.2.5	Unmount & Close	
1.3	Preparing the chroot environment	
2	Entering the chroot.	
2.1	Installing additional packages	
2.2	Master of time	
2.3	Master of locales	
2.4	Naming your machine.	
2.4.1	hostname	
2.4.2	hosts	
2.5	User setup	
2.5.1	Give root a password	
2.5.2	Create a personal user	
2.6	Boot manager	
2.6.1	EFISTUB	
2.6.2	grub	
2.6.2.1	BIOS	
2.6.2.2	UEFI	

2.6.2.3	grub config	
2.7	Switch to a systemd based ramdisk	
2.8	Hibernation	
2.9	Secure Boot	
2.9.1	shim	
2.9.2	The manual way	
2.9.2.1	File formats	
2.9.2.2	Create the keys	
2.9.2.3	Windows stuff	
2.9.2.4	Move the kernel & keys	
2.9.2.5	Signing	
2.9.2.6	Add EFI entries	
2.9.2.7	Enrolling everything	
3	Inside the DustArch	
3.1	Someone there?	
3.2	Update and upgrade	
3.3	Enabling the multilib repository	
3.4	zsh for president	
3.5	git	
3.6	Security is important	
3.6.1	Smartcard shenanigans	
3.7	Additional required tools	
3.8	Setting up a home environment	
3.8.1	Use dotfiles for a base config	
3.8.2	Set up gpg	
3.8.3	Finalize the dotfiles	
3.8.4	gpg-agent forwarding	
3.8.5	Back to your roots	
3.9	Audio	
3.9.1	alsa	
3.9.2	pulseaudio	
3.9.3	jack	
3.9.4	Audio handling	
3.10	Bluetooth	
3.11	Graphical desktop environment	
3.11.1	NVIDIA	
3.11.2	Launching the graphical environment	
3.11.2.1	The NVIDIA way	
3.12	Additional fish software	
3.12.1	tmux	
3.12.2	Communication	
3.12.2.1	weechat	

3.12.3	PDF viewer
3.13	Additional hybrid software.
3.13.1	Password management
3.13.2	python
3.13.3	ruby & asciidoctor
3.13.4	JUCE and FRUT
3.13.4.1	Using JUCE.
3.13.5	Additional development tools
3.13.5.1	Code formatting
3.13.5.2	Documentation
3.13.5.3	Build tools
3.13.6	Android file transfer
3.13.6.1	simple-mtpfs ^{AUR}
3.13.6.2	adb
3.13.7	Partition management
3.13.8	PDF viewer
3.13.9	Process management
3.13.10	Video software
3.13.10.1	Live streaming a terminal session
3.14	Additional GUI software
3.14.1	Session Lock
3.14.2	xfce-polkit ^{AUR}
3.14.3	Desktop background.
3.14.4	Compositing software
3.14.5	networkmanager applet
3.14.6	Show keyboard layout
3.14.7	X clipboard
3.14.8	Taking screen shots
3.14.9	Image viewer
3.14.10	File manager
3.14.11	Archive manager
3.14.12	Web browser
3.14.12.1	Entering the dark side
3.14.13	Office utilities
3.14.13.1	Printing
3.14.14	Communication
3.14.14.1	Email.
3.14.14.2	Telegram
3.14.14.3	TeamSpeak 3
3.14.14.4	Discord

3.14.15	Video software
3.14.15.1	Viewing video.
3.14.15.2	Creating video
3.14.15.3	Editing video
3.14.15.4	Utilizing video
3.14.16	Audio Production.
3.14.16.1	Ardour
3.14.16.2	Reaper
3.14.17	Virtualization
3.14.18	Gaming
3.14.19	Wacom
3.14.20	VNC & RDP
4	Upgrading the system
4.1	Fixing a faulty kernel upgrade.
5	Additional notes

Chapter 1

Inside the `archiso`

This chapter is aimed at assisting with the general setup of a customized Arch Linux installation, using an official Arch Linux image (`archiso`).

NOTE

As Arch Linux is a rolling release GNU/Linux distribution, it is advised, to have a working internet connection, in order to get the latest package upgrades and to install additional software, as the `archiso` doesn't have all packages available from cache, especially the ones that need to be installed from the **AUR**.

Furthermore, one should bear in mind that depending on the version, or rather modification date, of this guide, the exact steps taken may already be outdated. If you encounter any problems along the way, you will either have to resolve the issue yourself, or utilize the great [ArchWiki](https://wiki.archlinux.org/)^a, or the [Arch Linux forums](https://bbs.archlinux.org/)^b.

^a<https://wiki.archlinux.org/>

^b<https://bbs.archlinux.org/>

NOTE

In the following document, I will denote a **root** shell with a preceding `#` and a **user** shell with a preceding `>`.

1.1 Syncing up pacman

First of all we need to sync up `pacman`'s package repository, in order to be able to install the latest, as well as new packages to the `archiso` and our new system.

```
> pacman -Sy
```

WARNING

Using `> pacman -Sy` should be sufficient, in order to be able to search for packages from within the `archiso`, without upgrading the system, but might break your system, if you use this command on an existing installation!

To be on the safe side, it is advised to always use `> pacman -Syu` instead!

`pacstrap` uses the latest packages anyways.

1.1.1 Official repositories

After doing that, we can now install any software from the official repositories by issuing

```
root in ~  
> pacman -S <package_name>
```

where you would replace `<package_name>` with the actual package name.

If you want to remove an installed package, just use

```
root in ~  
> pacman -Rsu <package_name>
```

If you don't know the exact package name, or if you just want to search for a keyword, for example `xfce`, to list all packages having to do something with `xfce`, use

```
root in ~  
> pacman -Ss <keyword>
```

CAUTION

If you really need to force remove a package, which you should use *with extreme caution*, you could use

```
root in ~  
> pacman -Rdd <package_name>
```

1.1.2 AUR

If you want to install a package from the [AUR](https://aur.archlinux.org/)¹, I would advise proceeding in the following manner, in order to install the AUR-helper `paru`.

1. Clone the package with `git`

```
dustvoice in ~
> git clone https://aur.archlinux.org/paru.git
```

NOTE

If you are on a slow PC, or don't want to compile `paru` from scratch, you can also use [paru-bin](https://aur.archlinux.org/paru-bin.git)^a.

^a<https://aur.archlinux.org/paru-bin.git>

2. Switch to the package directory

```
dustvoice in ~
> cd paru
```

3. Execute `> makepkg`

```
dustvoice in ~/paru
> makepkg -si
```

4. Delete all files created, as `paru` will now be handling all the AUR stuff.

```
dustvoice in ~/paru
> cd ..

dustvoice in ~
> rm -rf paru
```

¹<https://aur.archlinux.org/>

NOTE

If you only install AUR packages the manual way, you might have to resolve some AUR dependencies manually, which can't be automatically resolved by `makepkg`'s `-s` option, which uses `pacman`.

WARNING

In order to install a desired AUR package, you *must* switch to your normal, non-root user, because `makepkg` doesn't run as `root`.

1.1.3 Software categories

In this guide, software is categorized in three different categories

- **Fish** software is intended to be used with either the native linux fish, or with a terminal emulator
- **GUI** software is intended to be used within a graphical desktop environment
- **Hybrid** software can either be used within both a fish and a graphical desktop environment (e.g. `networkmanager`), or there are packages available for both fish and a graphical desktop environment (e.g. `pulseaudio` with `pulsemixer` for **Fish** and `pavucontrol` for **GUI**)

1.1.4 Software installation

In this guide, I'll be explicitly listing the packages installed in a specific section at the beginning of the individual sections.

This allows you to

- clearly see what packages get installed / need to be installed in a specific section
- install packages before you start with the section in order to minimize waiting time
- not having to accidentally reinstall already installed packages

NOTE

The packages are always the recommended packages.

For further clarification for specific packages (e.g. UEFI specific packages), continue reading the section, as there is most certainly an explanation there.

Of course, as always, you can and should adapt everything according to your needs, as this guide is, again, *no tutorial, but a guide*.

1.1.4.1 Example section

```
core | libutil-linux
extra | git
community | ardour cadence jsampler
        | linuxsampler qsampler
        | sample-package
AUR | sbupdate
```

You have to configure `sample-package`, by editing `/etc/sample.conf`

```
Sample.text=useful
```

Code-Listing 1.1: `/etc/sample.conf`

1.2 Formatting the drive

First, you probably want to get a list of all available drives, together with their corresponding device name, by issuing

```
root in ~  
> fdisk -l
```

NOTE

The output of `> fdisk -l` is dependent on your system configuration and many other factors, like BIOS initialization order, etc.

CAUTION

Don't assume the same path of a device between reboots!

Always double check!

There is nothing worse than formatting a drive you didn't mean to format!

1.2.1 The standard way

In my case, the partition I want to install the root file system on is `/dev/sdb2`. `/dev/sdb3` will be my `swap` partition.

NOTE

A `swap` size twice the size of your RAM is recommended by a lot of people.

To be exact, every distribution has different recommendations for `swap` sizes. Also `swap` size heavily depends on whether you want to be able to hibernate, etc.

In my opinion You should make the `swap` size at least your RAM size and for RAM sizes over 4GB and the wish to hibernate, at least one and a half your RAM size.

IMPORTANT

If you haven't yet partitioned your disk, please refer to the [general partitioning tutorial](#)^a in the ArchWiki.

^a<https://wiki.archlinux.org/index.php/Partitioning>

Now we need to format the partitions accordingly

```
root in ~
> mkfs.ext4 /dev/sdb2

root in ~
> mkswap /dev/sdb3
```

After doing that, we can turn on the `swap` and mount the root partition.

```
root in ~
> swapon /dev/sdb3

root in ~
> mount /dev/sdb2 /mnt
```

NOTE

If you have an additional EFI System partition, because of a *UEFI - GPT* setup or an existing Windows installation, for example, which we will assume to be located under `/dev/sda2` (`/dev/sda` is the disk of my Windows install), you'll have to mount this partition to the new system's `/boot` folder

```
root in ~
> mkdir /mnt/boot

root in ~
> mount /dev/sda2 /mnt/boot
```

1.2.2 Full system encryption

NOTE

This is only one way to do it and it is the way I have done it. I'm using a LVM on LUKS setup, with `lvm2` and `luks2`. For more information look into the [ArchWiki](https://wiki.archlinux.org/)^a.

^a<https://wiki.archlinux.org/>

NOTE

This setup has different partitions, used for the EFI System partition, the `root` partition, etc., compared to the ones used in the rest of the guide. If you want to use `grub` in conjunction with some full system encryption, you would have to adapt the disk and partition names accordingly. The only part of the guide, which currently uses the drives & partitions used in this section is [The manual way](#).

To start things, we first have to decide, which disk, or partition, is going to hold the `luks2` encrypted `lvm2` stuff.

In my case I'll be using my NVMe SSD, with a GPT partition scheme, for both the EFI System partition, in my case `/dev/nvme0n1p1`, defined as a `EFI System` partition type in `fdisk`, as well as the main LUKS volume, in my case `/dev/nvme0n1p2`, defined as a `Linux filesystem` partition type in `fdisk`.

After partitioning our disk, we now have to set everything up.

1.2.2.1 EFI System partition

core | dosfstools

I won't setup my EFI System partition with `cryptsetup`, as it makes no sense in my case.

Every EFI binary (or STUB) will have to be signed with my custom Secure Boot keys, as described in [The manual way](#), so tempering with the EFI System partition poses no risk to my system.

Instead I will simply format it with a FAT32 filesystem

```
root in ~  
> mkfs.fat -F 32 -L /efi /dev/nvme0n1p1
```

We will bother with mounting it later on.

NOTE

When you *do* want to encrypt your EFI System partition, in conjunction with using `grub`, please either use LUKS 1, or make sure to have the latest version of `grub` installed on your system, to make it work with LUKS 2!

1.2.2.2 LUKS

core | cryptsetup

First off we have to create the LUKS volume

```
root in ~  
> cryptsetup luksFormat --type luks2 /dev/nvme0n1p2
```

After that we have to open the volume

```
root in ~  
> cryptsetup open /dev/nvme0n1p2 cryptroot
```

The volume is now accessible under `/dev/mapper/cryptroot`.

1.2.2.3 LVM

core | lvm2

I'm going to create one PV (Physical Volume) using the just created and opened **cryptroot** LUKS volume, one VG (Volume Group), named **DustArch1**, which will contain two LVs (Logical Volumes) named **root** and **swap** containing the **root** filesystem and the **swap** space respectively.

```
root in ~
> pvcreate /dev/mapper/cryptroot

root in ~
> vgcreate DustArch1 /dev/mapper/cryptroot

root in ~
> lvcreate -l 100%FREE -n root DustArch1

root in ~
> lvreduce -L -32G /dev/DustArch1/root

root in ~
> lvcreate -l 100%FREE -n swap DustArch1
```

1.2.2.4 Format & mount

Now the only thing left to do is formatting our freshly created logical volumes appropriately

```
root in ~
> mkfs.ext4 -L / /dev/DustArch1/root

root in ~
> mkswap /dev/DustArch1/swap
```

as well as mounting them and enabling the `swap`, in order to proceed with the next steps.

```
root in ~
> mount /dev/DustArch1/root /mnt

root in ~
> mkdir /mnt/efi

root in ~
> mount /dev/nvme0n1p1 /mnt/efi

root in ~
> swapon /dev/DustArch1/swap
```

1.2.2.5 Unmount & Close

WARNING

Only do this, after you're finished with your setup within the `archiso` and are about to reboot into your new system, or else the next steps won't work for you.

To close everything back up again,

1. unmount the volumes

```
root in ~  
> umount /mnt/efi /mnt
```

2. deactivate the VG

```
root in ~  
> vgchange -a n DustArch1
```

3. close the LUKS volume

```
root in ~  
> cryptsetup close cryptroot
```

1.3 Preparing the chroot environment

First it might make sense to edit `/etc/pacman.d/mirrorlist` to move the mirrors geographically closest to you to the top.

If you're using an older version of the `archiso`, you might want to replace the mirrorlist present on the `archiso` with the newest one from <https://archlinux.org/mirrorlist>²

```
root in ~
> curl https://archlinux.org/mirrorlist/all >
↪ /etc/pacman.d/mirrorlist
```

NOTE

community | reflector

The best way to do this, is using a package from the official repositories named `reflector`. It comes with all sorts of options, for example sorting mirrors by speed, filtering by country, etc.

```
root in ~
> reflector --verbose --latest 200 --sort rate --save
↪ /etc/pacman.d/mirrorlist
```

After that you would need to reinstall the `pacman-mirror` package and run

```
root in ~
> pacman -Syyuu
```

for the best results.

After that we can `pacstrap` the *minimum packages* needed. We will install all other packages later on.

²<https://archlinux.org/mirrorlist>

core | base linux linux-firmware

NOTE

This is the actual command used in my case

```
root in ~  
> pacstrap /mnt base linux linux-firmware
```

After that generate an `fstab` using `genfstab`

```
root in ~  
> genfstab -U /mnt >> /mnt/etc/fstab
```

and you're ready to enter the `chroot` environment.

Chapter 2

Entering the chroot

NOTE

As we want to set up our new system, we need to have access to the different partitions, the internet, etc. which we wouldn't get by solely using `chroot`.

That's why we are using `arch-chroot`, provided by the `arch-install-scripts` package, which is shipped with the `archiso`. This script takes care of all the aforementioned stuff, so we can set up our system properly.

```
root in ~  
> arch-chroot /mnt
```

Et Voilà! You successfully `chrooted` inside your new system and you'll be greeted by a `bash` prompt, which is the default shell on fresh Arch Linux installations.

2.1 Installing additional packages

core	amd-ucode base-devel diffutils dmraid dnsmasq dosfstools efibootmgr exfat-utils grub iputils lvm2 openssh sudo usbutils
extra	efitools git intel-ucode networkmanager networkmanager-openconnect networkmanager-openvpn parted polkit rsync zsh
community	neovim os-prober

NOTE

There are many command line text editors available, like `nano`, `vi`, `vim`, `emacs`, etc.

I'll be using `neovim`, though it shouldn't matter what editor you choose for the rest of the guide.

Make sure to enable the `NetworkManager.service` service, in order for the Internet connection to work correctly, upon booting into the fresh system later on.

```
root in /  
> systemctl enable NetworkManager.service
```

With `polkit` installed, create a file to enable users of the `network` group to add new networks without the need of `sudo`.

```
polkit.addRule(function(action, subject) {
    if (action.id.indexOf("org.freedesktop.NetworkManager.")
        → == 0 && subject.isInGroup("network")) {
        return polkit.Result.YES;
    }
});
```

Code-Listing 2.1: /etc/polkit-1/rules.d/50-org.fre_ |
edesktop.NetworkManager.rules

If you use UEFI, you'll also need the `efibootmgr`, in order to modify the UEFI entries.

2.2 Master of time

After that, you have to set your timezone and update the system clock.

Generally speaking, you can find all the different timezones under `/usr/share/zoneinfo`.

In my case, my timezone file resides under `/usr/share/zoneinfo/Europe/Berlin`.

To achieve the desired result, I will want to symlink this to `/etc/localtime` and set the hardware clock.

```
root in /
> ln -s /usr/share/zoneinfo/Europe/Berlin /etc/localtime

root in /
> hwclock --systohc --utc
```

Now you can also enable time synchronization over network

```
root in /
> timedatectl set-timezone Europe/Berlin

root in /
> timedatectl set-ntp true
```

and check that everything is alright

```
root in /
> timedatectl status
```

2.3 Master of locales

Now you have to generate your locale information.

For that you have to edit `/etc/locale.gen` and uncomment the locales you want to enable.

NOTE

I recommend to always uncomment `en_US.UTF-8 UTF8`, even if you want to use another language primarily.

In my case I only uncommented the `en_US.UTF-8 UTF8` line

```
en_US.UTF-8 UTF8
```

Code-Listing 2.2: `/etc/locale.gen`

After that you still have to actually generate the locales by issuing

```
root in /  
> locale-gen
```

and set the locale

```
root in /  
> localectl set-locale LANG="en_US.UTF-8"
```

After that we're done with this part.

2.4 Naming your machine

Now we can set the `hostname` for our new install and add `hosts` entries.

Apart from being mentioned in your command prompt, the `hostname` also serves the purpose of identifying, or naming your machine locally, as well as in a networked scenario. This will enable you to see your PC with the correct name in your router, etc.

2.4.1 hostname

To change the `hostname`, simply edit `/etc/hostname`, enter the desired name, then save and quit

```
DustArch
```

Code-Listing 2.3: `/etc/hostname`

2.4.2 hosts

Now we need to specify some `hosts` entries by editing `/etc/hosts`

```
# Static table lookup for hostnames.  
# See hosts(5) for details.  
  
127.0.0.1    localhost          .  
::1         localhost          .  
127.0.1.1   DustArch.localhost DustArch
```

Code-Listing 2.4: `/etc/hosts`

2.5 User setup

Now you should probably change the default `root` password and create a new `non-root` user for yourself, as using your new system purely through the native `root` user is not recommended from a security standpoint.

2.5.1 Give root a password

To change the password for the current user (the `root` user) issue

```
root in /  
> passwd
```

and choose a new password.

2.5.2 Create a personal user

core		sudo
extra		zsh

We are going to create a new user and set the password, groups and shell for this user

```
root in /
> useradd -m -p "" -G "adm,audio,disk,floppy,kvm,log,lp,netw_
↪ ork,rfkill,scanner,storage,users,optical,power,wheel" -s
↪ /usr/bin/zsh dustvoice

root in /
> passwd dustvoice
```

We now have to allow the `wheel` group `sudo` access.

For that we edit `/etc/sudoers` and uncomment the `%wheel [...]` line.

```
%wheel ALL=(ALL) ALL
```

Code-Listing 2.5: `/etc/sudoers`

You could also add a new line below the `root` line

```
root ALL=(ALL) ALL
```

Code-Listing 2.6: `/etc/sudoers`

with your new username

```
dustvoice ALL=(ALL) ALL
```

Code-Listing 2.7: `/etc/sudoers`

to solely grant the `new` user `sudo` privileges.

2.6 Boot manager

In this section different boot managers / boot methods are explained.

2.6.1 EFISTUB

core | efibootmgr

You can directly boot the system, by making use of the EFISTUB contained in the kernel image. To utilize this, we can use `efibootmgr` to create an entry in the UEFI

```
root in /
> efibootmgr --disk /dev/sda --part 2 --create --label "Arch
↳ Linux" --loader /vmlinuz-linux --unicode
↳ 'root=6ff60fab-c046-47f2-848c-791fbc52df09 rw
↳ initrd=\initramfs-linux.img
↳ resume=UUID=097c6f11-f246-40eb-a702-ba83c92654f2'
↳ --verbose
```

NOTE

This only makes sense of course, if you're using UEFI instead of a legacy BIOS. In this case it doesn't matter of course, if your machine *theoretically supports* UEFI, but rather if it is the *enabled mode*!

2.6.2 grub

core		dosfstools efibootmgr grub
extra		mtools
community		os-prober

Of course you can also use a boot manager to boot the system, as the name implies.

If I can't use `EFISTUB`, e.g. either because the system has no `UEFI` support, or because I need another feature of a boot manager, I normally use `grub`.

NOTE

You'll probably only need the `efibootmgr` package, if you plan to utilize `UEFI`.

2.6.2.1 BIOS

If you chose the BIOS - MBR variation, you'll have to *do nothing special*.

If you chose the BIOS - GPT variation, you'll have to *have a +1M boot partition* created with the partition type set to BIOS boot.

In both cases you'll have to *run the following command* now

```
root in /  
> grub-install --target=i386-pc /dev/sdb
```

NOTE

It should be obvious that you would need to replace `/dev/sdb` with the disk you actually want to use. Note however that you have to specify a *disk* and *not a partition*, so *no number*.

2.6.2.2 UEFI

If you chose the UEFI - GPT variation, you'll have to *have the EFI System partition mounted at /boot* (where `/dev/sda2` is the partition holding said EFI System partition in my particular setup)

Now *install grub to the EFI System partition*

```
root in /
> grub-install --target=x86_64-efi --efi-directory=/boot
↳ --bootloader-id=grub --recheck
```

IMPORTANT

If you've planned on dual booting arch with Windows and therefore reused the EFI System partition created by Windows, you might not be able to boot to grub just yet.

In this case, boot into Windows, open a cmd window as Administrator and type in

```
bcdedit /set {bootmgr} path \EFI\grub\grubx64.efi
```

To make sure that the path is correct, you can use

```
root in /
> ls /boot/EFI/grub
```

under Linux to make sure, that the `grubx64.efi` file is really there.

2.6.2.3 grub config

In all cases, you now have to create the main `grub.cfg` configuration file.

But before we actually generate it, we'll make some changes to the default `grub` settings, which the `grub.cfg` will be generated from.

Adjust the timeout First of all, I want my `grub` menu to wait indefinitely for my command to boot an OS.

```
GRUB_TIMEOUT=-1
```

Code-Listing 2.8: `/etc/default/grub`

NOTE

I decided on this, because I'm dual booting with Windows and after Windows updates itself, I don't want to accidentally boot into my Arch Linux, just because I wasn't quick enough to select the Windows Boot Loader from the `grub` menu.

Of course you can set this parameter to whatever you want.

Another way of achieving what I described, would be to make `grub` remember the last selection.

```
GRUB_TIMEOUT=5
GRUB_DEFAULT=saved
GRUB_SAVEDEFAULT="true"
```

Code-Listing 2.9: `/etc/default/grub`

Enable the recovery After that I also want the recovery option showing up, which means that besides the standard and fallback images, also the recovery one would show up.

```
GRUB_DISABLE_RECOVERY=false
```

Code-Listing 2.10: `/etc/default/grub`

NVIDIA fix Now, as I'm using the binary NVIDIA driver for my graphics card, I also want to make sure, to revert `grub` back to text mode, after I select

a boot entry, in order for the NVIDIA driver to work properly. You might not need this

```
GRUB_GFXPAYLOAD_LINUX=text
```

Code-Listing 2.11: /etc/default/grub

Add power options I also want to add two new menu entries, to enable me to shut down the PC, or reboot it, right from the grub menu.

```
menuentry '=> Shutdown' {
    halt
}

menuentry '=> Reboot' {
    reboot
}
```

Code-Listing 2.12: /etc/grub.d/40_custom

Installing memtest As I want all possible options to possibly troubleshoot my PC right there in my **grub** menu, without the need to boot into a live OS, I also want to have a memory tester there.

BIOS

extra | memtest86+

For a BIOS setup, you'll simply need to install the **memtest86+** package, with no further configuration.

UEFI

AUR | memtest86-efi

For a UEFI setup, you'll first need to install the package and then tell **memtest86-efi**^{AUR} how to install itself

```
root in /
> mentest86-efi -i
```

Now select option 3, to install it as a `grub2` menu item.

Enabling hibernation We need to add the `resume` kernel parameter to `/etc/default/grub`, containing my `swap` partition UUID, in my case

```
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet
↪ resume=UUID=097c6f11-f246-40eb-a702-ba83c92654f2"
```

Code-Listing 2.13: `/etc/default/grub`

NOTE

If you have to change anything, like the `swap` partition UUID, inside the `grub` configuration files, you'll always have to rerun `> grub-mkconfig` as explained in the paragraph [Generating the grub config](#) of the section [grub config](#).

Disabling os-prober Sometimes it makes sense to disable the `os-prober` functionality of `grub`, even though `os-prober` is installed on the system (which auto enables it), for example when installing `arch` for portability purposes. We can disable the `os-prober` functionality in the `grub` default config file.

```
GRUB_DISABLE_OS_PROBER=true
```

Code-Listing 2.14: `/etc/default/grub`

Generating the grub config Now we can finally generate our `grub.cfg`

```
root in /
> grub-mkconfig -o /boot/grub/grub.cfg
```

Now you're good to boot into your new system.

2.7 Switch to a `systemd` based `ramdisk`

NOTE

There is nothing particularly better about using a `systemd` based `ramdisk` instead of a `busybox` one, it's just that I prefer it.

Some advantages, at least in my opinion, that the `systemd` based `ramdisk` has, are the included `resume` hook, as well as password caching, when decrypting encrypted volumes, which means that because I use the same LUKS password for both my data storage HDD, as well as my `cryptroot`, I only have to input the password once for my `cryptroot` and my data storage HDD will get decrypted too, without the need to create `/etc/crypttab` entries, etc.

To switch to a `systemd` based `ramdisk`, you will normally need to substitute the `busybox` specific hooks for `systemd` ones. You will also need to use `systemd` hooks from now on, for example `sd-encrypt` instead of `encrypt`.

- `base`

In my case, I left the `base` hook untouched, to get a `busybox` recovery shell, if something goes wrong, although you wouldn't technically need it, when using `systemd`.

WARNING

Don't remove this, when using `busybox`, unless you're absolutely knowing what you're doing.

- `udev`

Replace this with `systemd` to switch from `busybox` to `systemd`.

- `keymap` and/or `fishfont`

These two, or one, if you didn't use one of them, need to be replaced with `sd-vfish`. Everything else stays the same with these.

- `encrypt`

Isn't used in the default `/etc/mkinitcpio.conf`, but could be important later on, for example when using [Full system encryption](#). You need to substitute this with `sd-encrypt`.

- `lvm2`

Same thing as with `encrypt` and needs to be substituted with `sd-lvm2`.

NOTE

You can find all purposes of the individual hooks, as well as the `busybox` / `systemd` equivalent of each one in the [ArchWiki^a](https://wiki.archlinux.org/index.php/Mkinitcpio#Common_hooks).

^ahttps://wiki.archlinux.org/index.php/Mkinitcpio#Common_hooks

2.8 Hibernation

In order to use the hibernation feature, you should make sure that your `swap` partition/file is at least the size of your RAM.

NOTE

If you use a `busybox` based `ramdisk`, you need to

1. add the `resume` hook to `/etc/mkinitcpio.conf`, before `fsck` and definitely after `block`

```
HOOKS=(base udev autodetect modconf block filesystems
↪ keyboard resume fsck)
```

Code-Listing 2.15: `/etc/mkinitcpio.conf`

2. run

```
root in /
> mkinitcpio -p linux
```

NOTE

When using `EFISTUB` without `sbupdate`, your motherboard has to support kernel parameters for boot entries. If your motherboard doesn't support this, you would need to use `systemd-boot`^a.

^a<https://wiki.archlinux.org/index.php/Systemd-boot>

2.9 Secure Boot

2.9.1 shim

AUR | [shim-signed](#)

WARNING

This is a way of handling secure boot that aims at just making everything work!

It is not the way Secure Boot was intended to be used and you might as well disable it.

If you need Secure Boot to be enabled, e.g. for Windows, but you couldn't care less for the security it could bring to your device, use this method.

If you want to actually make use of the Secure Boot feature, read [The manual way](#).

I know I told you that you're now good to boot into your new system. That is only correct, if you're *not* using Secure Boot.

You can either proceed by disabling Secure Boot in your firmware settings, or by using `shim` as kind of a pre-bootloader, as well as signing your bootloader (`grub`) and your kernel.

If you decided on using Secure Boot, you will first have to install the package.

Now we just need to copy `shimx64.efi`, as well as `mmx64.efi` to our EFI System partition

```
root in /
> cp /usr/share/shim-signed/shimx64.efi /boot/EFI/grub/

root in /
> cp /usr/share/shim-signed/mmx64.efi /boot/EFI/grub/
```

NOTE

If you have to use `bcdedit` from within Windows, as explained in section [UEFI](#), you need to adapt the command accordingly

```
bcdedit /set {bootmgr} path \EFI\grub\shimx64.efi
```

Now you will be greeted by `MokManager` everytime you update your boot-loader or kernel.

Just choose “Enroll hash from disk” and enroll your bootloader binary (`grubx64.efi`) and kernel (`vmlinuz-linux`).

Reboot and your system should fire up just fine.

2.9.2 The manual way

WARNING

As this is a very tedious and time consuming process, it only makes sense when also utilizing some sort of disk encryption, which is, why I would advise you to read [Full system encryption](#) first.

2.9.2.1 File formats

In the following subsections, we will be dealing with some different file formats.

.key

PEM format private keys for EFI binary and EFI signature list signing.

.crt

PEM format certificates for `sbsign`.

.cer

DER format certificates for firmware.

.esl

Certificates in EFI Signature List for `KeyTool` and/or firmware.

.auth

Certificates in EFI Signature List with authentication header (i.e. a signed certificate update file) for `KeyTool` and/or firmware.

2.9.2.2 Create the keys

First off, we have to generate our Secure Boot keys.

These will be used to sign any binary which will be executed by the firmware.

GUID Let's create a GUID first to use with the next commands.

```
root in ~/sb
> uuidgen --random > GUID.txt
```

PK We can now generate our PK (Platform Key)

```
root in ~/sb
> openssl req -newkey rsa:4096 -nodes -keyout PK.key -new
↳ -x509 -sha256 -subj "/CN=Platform Key for DustArch/"
↳ -out PK.crt

root in ~/sb
> openssl x509 -outform DER -in PK.crt -out PK.cer

root in ~/sb
> cert-to-efi-sig-list -g "$(< GUID.txt)" PK.crt PK.esl

root in ~/sb
> sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt
↳ PK PK.esl PK.auth
```

In order to allow deletion of the PK, for firmwares which do not provide this functionality out of the box, we have to sign an empty file.

```
root in ~/sb
> sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt
↳ PK /dev/null rm_PK.auth
```

KEK We proceed in a similar fashion with the KEK (Key Exchange Key)

```
root in ~/sb
> openssl req -newkey rsa:4096 -nodes -keyout KEK.key -new
↳ -x509 -sha256 -subj "/CN=Key Exchange Key for DustArch/"
↳ -out KEK.crt
```

```
root in ~/sb
> openssl x509 -outform DER -in KEK.crt -out KEK.cer

root in ~/sb
> cert-to-efi-sig-list -g "$(< GUID.txt)" KEK.crt KEK.esl

root in ~/sb
> sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt
↪ KEK KEK.esl KEK.auth
```

DB And finally the DB (Signature Database) key.

```
root in ~/sb
> openssl req -newkey rsa:4096 -nodes -keyout db.key -new
↪ -x509 -sha256 -subj "/CN=Signature Database key for
↪ DustArch" -out db.crt

root in ~/sb
> openssl x509 -outform DER -in db.crt -out db.cer

root in ~/sb
> cert-to-efi-sig-list -g "$(< GUID.txt)" db.crt db.esl

root in ~/sb
> sign-efi-sig-list -g "$(< GUID.txt)" -k KEK.key -c KEK.crt
↪ db db.esl db.auth
```

2.9.2.3 Windows stuff

As your plan is to be able to control, which things do boot on your system and which don't, you're going through all this hassle to create and enroll custom keys, so only EFI binaries signed with said keys can be executed.

But what if you have a Windows dual boot setup?

Well the procedure is actually pretty straight forward. You just grab Microsoft's certificates, convert them into a usable format, sign them and enroll them. No need to sign the Windows boot loader.

```
root in ~/sb
> curl -fLo WinCert.crt https://www.microsoft.com/pkiops/cer_j
↳ ts/MicWinProPCA2011_2011-10-19.crt

root in ~/sb
> openssl x509 -inform DER -outform PEM -in MicWinCert.crt
↳ -out MicWinCert.pem

root in ~/sb
> cert-to-efi-sig-list -g
↳ 77fa9abd-0359-4d32-bd60-28f4e78f784b MicWinCert.pem
↳ MS_db.esl

root in ~/sb
> sign-efi-sig-list -a -g
↳ 77fa9abd-0359-4d32-bd60-28f4e78f784b -k KEK.key -c
↳ KEK.crt db MS_db.esl add_MS_db.auth
```

2.9.2.4 Move the kernel & keys

In order to ensure a smooth operation, with actual security, we need to move some stuff around.

Kernel, initramfs, microcode `pacman` will put its unsigned and unencrypted kernel, `initramfs` and microcode images into `/boot`, which is, why it will be no longer a good idea, to leave your EFI System partition mounted there. Instead we will create a new mountpoint under `/efi` and modify our `fstab` accordingly.

Keys As you probably want to automate signing sooner or later and only use the ultimately necessary keys for this process, as well as store the other more important keys somewhere more safe and secure than your `root` home directory, we will move the necessary keys.

I personally like to create a `/etc/efi-keys` directory, `chmodded` to `700` and place my `db.crt` and `db.key` there. All the keys will get packed into a `tar` archive and encrypted with a strong symmetric pass phrase and stored somewhere secure and safe.

2.9.2.5 Signing

Signing is the process of, well, signing your EFI binaries, in order for them to be allowed to be executed, by the motherboard firmware. At the end of the day, that's why you're doing all this, to prevent an attack by launching unknown code.

Manual signing Of course, you can sign images yourself manually. In my case, I used this, to sign the boot loader, kernel and `initramfs` of my USB installation of Arch Linux.

NOTE

As always, manual signing also comes with its caveats!

If I update my kernel, boot loader, or create an updated `initramfs` on my Arch Linux USB installation, I have to sign those files again, in order to be able to boot it on my PC.

Of course you can always script and automate stuff, but if you want something more easy for day to day use, I really recommend that you try out `sbupdate`, which I will explain in the next paragraph [sbupdate](#).

For example, if I want to sign the kernel image of my USB installation, where I mounted the boot partition to `/mnt/dustarchusb/boot`, I would have to do the following

```
root in ~/sb
> sbsign --key /etc/efi-keys/db.key --cert
  ↪ /etc/efi-keys/db.crt --output /boot/vmlinuz-linux
  ↪ /boot/vmlinuz-linux
```

`sbupdate`

AUR | `sbupdate-git`

Of course, if you're using Secure Boot productively, you would want something more practical than manual signing, especially since you need to sign

- the boot loader
- the kernel image
- the `initramfs`

Fortunately there is an easy and uncomplicated tool out there, that does all that for you, called **sbupdate**.

It not only signs everything and also foreign **EFI** binaries, if specified, but also combines your kernel and **initramfs** into a single executable **EFI** binary, so you don't even need a boot loader, if your motherboard implementation supports booting those.

After installing **sbupdate**, we can edit the `/etc/sbupdate.conf` file, to set everything up.

Everything in this config should be self-explanatory.

You will probably need to

- set `ESP_DIR` to `/efi`
- add any other **EFI** binary you want to have signed to `EXTRA_SIGN`
- add your kernel parameters, for example

- `rd.luks.name`
- `root`
- `rw`
- `resume`
- `etc.`

to `CMDLINE_DEFAULT`

After you've successfully configured **sbupdate**, you can run it as root, to create all the signed files.

NOTE

sbupdate will be executed upon kernel updates by **pacman**, but not if you change your **initramfs** with something like **mkinitcpio**. In that case you will have to run **sbupdate** manually.

2.9.2.6 Add EFI entries

core | efibootmgr

Now the only thing left to do, if you want to stay boot loader free anyways, is to add the signed images to the boot list of your NVRAM. You can do this with `efibootmgr`.

```
root in ~/sb
> efibootmgr -c -d /dev/nvme0n1 -p 1 -L "Arch Linux
↳ fallback" -l "EFI\Arch\linux-fallback-signed.efi

root in ~/sb
> efibootmgr -c -d /dev/nvme0n1 -p 1 -L "Arch Linux" -l
↳ "EFI\Arch\linux-signed.efi
```

Of course you can extend this list, with whichever entries you need.

2.9.2.7 Enrolling everything

First off, copy all `.cer`, `.esl` and `.auth` files to a FAT formatted filesystem. I'm using my EFI System partition for this.

After that reboot into the firmware setup of your motherboard, clear the existing Platform Key, to set the firmware into "Setup Mode" and enroll the db, KEK and PK certificates in sequence.

NOTE

Enroll the Platform Key last, as it sets most firmware's Secure Boot sections back into "User mode", exiting "Setup Mode".

Chapter 3

Inside the DustArch

This section helps at setting up the customized system from within an installed system.

This section mainly provides aid with the basic set up tasks, like networking, dotfiles, etc.

NOTE

Not everything in this section is mandatory.

This section is rather a guideline, because it is easy to forget some steps needed, for example `jack` for audio production, that only become apparent, when they're needed.

It is furthermore the responsibility of the reader to decide which steps to skip and which need further research. As I mentioned, this is only a guide and not the answer to everything.

3.1 Someone there?

First we have to check if the network interfaces are set up properly.

To view the network interfaces with all their properties, we can issue

```
dustvoice in ~  
> ip link
```

To make sure that you have a working *Internet* connection, issue

```
dustvoice in ~  
> ping archlinux.org
```

Everything should run smoothly if you have a wired connection.

If there is no connection and you're indeed using a wired connection, try restarting the `NetworkManager` service

```
dustvoice in ~  
> sudo systemctl restart NetworkManager.service
```

and then try `> ping` again.

If you're trying to utilize a Wi-Fi connection, use `nmcli`, the NetworkManager's command line tool, or `nmtui`, the NetworkManager terminal user interface, to connect to a Wi-Fi network.

NOTE

I never got `nmtui` to behave like I wanted it to, in my particular case at least, which is the reason why I use `nmcli` or the GUI tools.

First make sure, the scanning of nearby Wi-Fi networks is enabled for your Wi-Fi device

```
dustvoice in ~  
> nmcli radio
```

and if not, enable it

```
dustvoice in ~  
> nmcli radio wifi on
```

Now make sure your Wi-Fi interface appears under

```
dustvoice in ~  
> nmcli device
```

Rescan for available networks

```
dustvoice in ~  
> nmcli device wifi rescan
```

and list all found networks

```
dustvoice in ~  
> nmcli device wifi list
```

After that connect to the network

```
dustvoice in ~  
> nmcli device wifi connect --ask
```

Now try > pinging again.

3.2 Update and upgrade

After making sure that you have a working Internet connection, you can then proceed to update and upgrade all installed packages by issuing

```
dustvoice in ~  
> sudo pacman -Syu
```

3.3 Enabling the multilib repository

In order to make 32-bit packages available to `pacman`, we'll need to enable the multilib repository in `/etc/pacman.conf` first. Simply uncomment

```
[multilib]
Include = /etc/pacman.d/mirrorlist
```

Code-Listing 3.1: `/etc/pacman.conf`

and update `pacman`'s package repositories afterwards

```
dustvoice in ~
> sudo pacman -Syu
```

3.4 zsh for president

Of course you can use any shell you want. In my case I'll be using the `zsh` shell.

NOTE

I am using `zsh` because of its auto completion functionality and extensibility, as well as a brilliant `vim` like navigation implementation through a plugin, though that might not be what you're looking for.

We already set the correct shell for the `dustvoice` user in the [Create a personal user](#) step, but I want to use `zsh` for the `root` user too, so I'll have to change `root`'s default shell to it.

```
dustvoice in ~  
> sudo chsh -s /usr/bin/zsh root
```

Don't worry about the looks by the way, we're gonna change all that in just a second.

3.5 git

extra | git

Install the package and you're good to go for now, as we'll care about the `.gitconfig` in just a second.

3.6 Security is important

core | gnupg

If you've followed the tutorial using a recent version of the archiso, you'll probably already have the most recent version of `gnupg` installed by default.

3.6.1 Smartcard shenanigans

extra		libusb-compat
community		ccid opensc pcsclite

After that you'll still have to setup `gnupg` correctly. In my case I have my private keys stored on a smartcard.

To use it, I'll have to install the listed packages and then enable and start the `pcscd.service` service

```
dustvoice in ~
> sudo systemctl enable pcscd.service

dustvoice in ~
> sudo systemctl start pcscd.service
```

After that, you should be able to see your smartcard being detected

```
dustvoice in ~
> gpg --card-status
```

NOTE

If your smartcard still isn't detected, try logging off completely or even restarting, as that sometimes is the solution to the problem.

3.7 Additional required tools

core		make openssh
extra		clang cmake jdk-openjdk python
community		pass python-pynvim

To minimize the effort required by the following steps, we'll install most of the required packages beforehand

This will ensure, we proceed through the following section without the need for interruption, because a package needs to be installed, so the following content can be condensed to the relevant informations.

3.8 Setting up a home environment

In this step we're going to setup a home environment for both the `root` and my personal `dustvoice` user.

NOTE

In my case these 2 home environments are mostly equivalent, which is why I'll execute the following commands as the `dustvoice` user first and then switch to the `root` user and repeat the same commands.

I decided on this, as I want to edit files with elevated permissions and still have the same editor style and functions/plugins.

Note that this comes with some drawbacks. For example, if I change a configuration for my `dustvoice` user, I would have to regularly update it for the `root` user too. This bears the problem, that I have to register my smartcard for the `root` user. This in turn is problematic, cause the `gpg-agent` used for `ssh` authentication, doesn't behave well when used within a `> su` or `> sudo -i` session. So in order to update `root`'s config files I would either need to symlink everything, which I won't do, or I'll need to login as the `root` user now and then, to update everything.

NOTE

In my case, I want to access all my `git` repositories with my `gpg` key on my smartcard. For that I have to configure the `gpg-agent` with some configuration files that reside in a `git` repository. This means I will have to reside to using the `https` URL of the repository first and later changing the URL either in the corresponding `.git/config` file, or by issuing the appropriate command.

3.8.1 Use dotfiles for a base config

To provide myself with a base configuration, which I can then extend, I have created a `dotfiles` repository, which contains all kinds of configurations.

The special thing about this `dotfiles` repository is that it *is* my home folder. By using a curated `.gitignore` file, I'm able to only include the configuration files I want to keep between installs into the repository and ignore everything else.

To achieve this very specific setup, I have to turn my home directory into said `dotfiles` repository first

```
dustvoice in ~
> git init

dustvoice in ~
> git remote add origin
→ https://git.dustvoice.de/DustVoice/dotfiles.git

dustvoice in ~
> git fetch

dustvoice in ~
> git reset origin/master --hard

dustvoice in ~
> git branch --set-upstream-to=origin/master master
```

Now I can issue any `git` command in my `~` directory, because it now is a `git` repository.

3.8.2 Set up gpg

As I wanted to keep my `dotfiles` repository as modular as possible, I utilize `git`'s `submodule` feature. Furthermore I want to use my `nvim` repository, which contains all my configurations and plugins for `neovim`, on Windows, but without all the Linux specific configuration files. I am also using the `Pass` repository on my Android phone and Windows PC, where I only need this repository without the other Linux configuration files.

Before we'll be able to update the `submodules` (`nvim` config files and `password-store`) though, we will have to setup our `gpg` key as an `ssh` key, as I use it to authenticate

```
dustvoice in ~
> chmod 700 .gnupg

dustvoice in ~
> gpg --card-status

dustvoice in ~
> gpg --card-edit
```

```
(insert) gpg/card> fetch
(insert) gpg/card> q
```

```
dustvoice in ~
> gpg-connect-agent updatestartuptty /bye
```

NOTE

You would have to adapt the `keygrip` present in the `~/.gnupg/_sshcontrol` file to your specific `keygrip`, retrieved with `> gpg -K --with-keygrip`.

Now, as mentioned before, I'll switch to using `ssh` for authentication, rather than `https`

```
dustvoice in ~
> git remote set-url origin
↪ git@git.dustvoice.de:DustVoice/dotfiles.git
```

As the best method to both make `zsh` recognize all the configuration changes, as well as the `gpg-agent` behave properly, is to re-login, we'll do just that

```
dustvoice in ~  
> exit
```

WARNING

It is very important to note, that I mean *a real re-login*.

That means that if you've used `ssh` to log into your machine, it probably won't be sufficient to login into a new `ssh` session. You'll probably need to restart the machine completely.

3.8.3 Finalize the dotfiles

Now log back in and continue

```
dustvoice in ~
> git submodule update --recursive --init

dustvoice in ~
> source .zshrc

dustvoice in ~
> cd .config/nvim

dustvoice in ~/.config/nvim
> echo 'let g:platform = "linux"' >> platform.vim

dustvoice in ~/.config/nvim
> echo 'let g:use_autocomplete = 3' >> custom.vim

dustvoice in ~/.config/nvim
> echo 'let g:use_clang_format = 1' >> custom.vim

dustvoice in ~/.config/nvim
> echo 'let g:use_font = 0' >> custom.vim

dustvoice in ~/.config/nvim
> nvim --headless +PlugInstall +qa

dustvoice in ~/.config/nvim
> cd plugged/YouCompleteMe

dustvoice in ~/.config/nvim/plugged/YouCompleteMe
> python3 install.py --clang-completer --java-completer

dustvoice in ~/.config/nvim/plugged/YouCompleteMe
> cd ~
```

3.8.4 gpg-agent forwarding

Now there is only one thing left to do, in order to make the `gpg` setup complete: `gpg-agent` forwarding over `ssh`. This is very important for me, as I want to use my smartcard on my development server too, which requires me, to forward/tunnel my `gpg-agent` to my remote machine.

First of all, I want to setup a config file for `ssh`, as I don't want to pass all parameters manually to `ssh` every time.

```
Host <connection name>
  HostName <remote address>
  ForwardAgent yes
  ForwardX11 yes
  RemoteForward <remote agent-socket> <local
    ↪ agent-extra-socket>
  RemoteForward <remote agent-ssh-socket> <local
    ↪ agent-ssh-socket>
```

Code-Listing 3.2: `~/.ssh/config`

NOTE

You would of course, need to adapt the content in between the `<` and `>` brackets.

To get the paths needed as parameters for `RemoteForward`, issue

```
dustvoice in ~
> gpgconf --list-dirs
```

Example 1. *An example for a valid `~/.ssh/config` would be*

```
Host archserver
  HostName pc.dustvoice.de
  ForwardAgent yes
  ForwardX11 yes
  RemoteForward /run/user/1000/gnupg/S.gpg-agent
  → /run/user/1000/gnupg/S.gpg-agent.extra
  RemoteForward /run/user/1000/gnupg/S.gpg-agent.ssh
  → /run/user/1000/gnupg/S.gpg-agent.ssh
```

Code-Listing 3.3: ~/.ssh/config

Now you'll still need to enable some settings on the remote machines.

```
StreamLocalBindUnlink yes
AllowAgentForwarding yes
X11Forwarding yes
```

Code-Listing 3.4: /etc/ssh/sshd_config

Now just restart your remote machines and you're ready to go.

NOTE

If you use `alacritty`, to connect to your remote machine over `ssh`, you will need to install the `alacritty` on the remote machine too, as `alacritty` uses its own `$TERM`.

Another option would be changing that variable for the `ssh` command

```
dustvoice in ~
> TERM=xterm-256colors ssh remote-machine
```

3.8.5 Back to your roots

As mentioned before, you would now switch to the `root` user, either by logging in as `root`, or by using

```
dustvoice in ~  
> sudo -iu root
```

Now go back to [Setting up a home environment](#) to repeat all commands for the `root` user.

WARNING

A native login would be better compared to `> sudo -iu root`, as there could be some complications, like already running `gpg-agent` instances, etc., which you would need to manually resolve, when using `> sudo -iu root`.

3.9 Audio

Well, why wouldn't you want audio ...

3.9.1 alsa

extra | alsa-utils

NOTE

You're probably better off using `pulseaudio` and/or `jack`.

Now choose the sound card you want to use

```
dustvoice in ~  
> cat /proc/asound/cards
```

and then create `/etc/asound.conf`

```
defaults.pcm.card 2  
defaults.ctl.card 2
```

Code-Listing 3.5: `/etc/asound.conf`

NOTE

It should be apparent, that you would have to switch out 2 with the number corresponding to the sound card you want to use.

3.9.2 pulseaudio

extra		pavucontrol pulseaudio
community		pulsemixer

Some applications require `pulseaudio`, or work better with it, for example `discord`, so it might make sense to use `pulseaudio`

For enabling real-time priority for `pulseaudio` on Arch Linux, please make sure your user is part of the `audio` group and edit the file `/etc/pulse/daemon.conf`, so that you uncomment the lines

```
high-priority = yes
nice-level = -11

realtime-scheduling = yes
realtime-priority = 5
```

Code-Listing 3.6: `/etc/pulse/daemon.conf`

If your system can handle the load, you can also increase the remixing quality, by changing the `resample-method`

```
resample-method = speex-float-10
```

Code-Listing 3.7: `/etc/pulse/daemon.conf`

Of course a restart of the `pulseaudio` daemon is necessary to reflect the changes you just made

```
dustvoice in ~
> pulseaudio --kill

dustvoice in ~
> pulseaudio --start
```

3.9.3 jack

extra		pulseaudio-jack
community		cadence jack2

If you either want to manually control audio routing, or if you use some kind of audio application like **ardour**, you'll probably want to use **jack** and **cadence** as a GUI to control it, as it has native support for bridging **pulseaudio** to **jack**.

3.9.4 Audio handling

extra		libao libid3tag libmad
		libpulse opus wavpack
community		sox twolame

To also play audio, we need to install the mentioned packages and then simply do

```
dustvoice in ~  
> play audio.wav  
  
dustvoice in ~  
> play audio.mp3
```

to play audio.

3.10 Bluetooth

extra	bluez bluez-utils pulseaudio-bluetooth
community	blueman

To set up Bluetooth, we need to install the `bluez` and `bluez-utils` packages in order to have at least a command line utility `bluetoothctl` to configure connections

Now we need to check if the `btusb` kernel module was already loaded

```
dustvoice in ~
> sudo lsmod | grep btusb
```

After that we can enable and start the `bluetooth.service` service

```
dustvoice in ~
> sudo systemctl enable bluetooth.service

dustvoice in ~
> sudo systemctl start bluetooth.service
```

NOTE

To use `bluetoothctl` and get access to the Bluetooth device of your PC, your user needs to be a member of the `lp` group.

Now simply enter `bluetoothctl`

```
dustvoice in ~
> bluetoothctl
```

In most cases your Bluetooth interface will be preselected and defaulted, but in some cases, you might need to first select the Bluetooth controller

```
(insert) [DustVoice]# list
(insert) [DustVoice]# select <MAC_address>
```

After that, power on the controller

```
(insert) [DustVoice]# power on
```

Now enter device discovery mode

```
(insert) [DustVoice]# scan on
```

and list found devices

```
(insert) [DustVoice]# devices
```

NOTE

You can turn device discovery mode off again, after your desired device has been found

```
(insert) [DustVoice]# scan off
```

Now turn on the agent

```
(insert) [DustVoice]# agent on
```

and pair with your device

```
(insert) [DustVoice]# pair <MAC_address>
```

NOTE

If your device doesn't support PIN verification you might need to manually trust the device

```
(insert) [DustVoice]# trust <MAC_address>
```

Finally connect to your device

```
(insert) [DustVoice]# connect <MAC_address>
```

NOTE

If your device is an audio device, of some kind you might have to install `pulseaudio-bluetooth`.

You will then also need to append 2 lines to `/etc/pulse/system.pa`

```
load-module module-bluetooth-policy
load-module module-bluetooth-discover
```

Code-Listing 3.8: `/etc/pulse/system.pa`

and restart `pulseaudio`

```
dustvoice in ~
> pulseaudio --kill

dustvoice in ~
> pulseaudio --start
```

If you want a GUI to do all of this, just install `blueman` and launch `blueman-manager`

3.11 Graphical desktop environment

extra		ttf-hack xclip xorg xorg-drivers xorg-xinit
community		arandr alacritty bspwm dmenu sxhkd
AUR		polybar

If you decide, that you want to use a graphical desktop environment, you have to install additional packages in order for that to work.

NOTE

`xclip` is useful, when you want to send something to the X clipboard. It is also required, in order for `neovim`'s clipboard to work correctly. It is not required though.

3.11.1 NVIDIA

```
extra | nvidia nvidia-utils  
      | nvidia-settings opencl-nvidia
```

If you also want to utilize special NVIDIA functionality, for example for `davinci-resolve`, you'll most likely need to install their proprietary driver.

To configure the X server correctly, one can use `nvidia-xconfig`

```
dustvoice in ~  
> sudo nvidia-xconfig
```

If you want to further tweak all settings available, you can use `nvidia-settings`.

```
dustvoice in ~  
> sudo nvidia-settings
```

will enable you to “*Save to X Configuration File*”, which merges your changes with `/etc/X11/xorg.conf`.

With

```
dustvoice in ~  
> nvidia-settings
```

you'll only be able to save the current configuration to `~/.nvidia-setting_1 s-rc`, which you have to source after X startup with

```
dustvoice in ~  
> nvidia-settings --load-config-only
```

NOTE

You will have to reboot sooner or later after installing the NVIDIA drivers, so you might as well do it now, before any complications come up.

3.11.2 Launching the graphical environment

After that you can now do `startx` in order to launch the graphical environment.

If anything goes wrong in the process, remember that you can press `Ctrl` `]` `+Alt` `<Number>` to switch `ttys`.

3.11.2.1 The NVIDIA way

community		bbswitch
AUR		nvidia-xrun

If you're using an NVIDIA graphics card, you might want to use `nvidia-xrunAUR` instead of `startx`. This has the advantage, of the `nvidia` kernel modules, as well as the `nouveau` ones not loaded at boot time, thus saving power. `nvidia-xrunAUR` will then load the correct kernel modules and run the `.nvidia-xinitrc` script in your home directory (for more file locations look into the documentation for `nvidia-xrunAUR`).

IMPORTANT

At the time of writing, `nvidia-xrunAUR` needs `sudo` permissions before executing its task.

NOTE

AUR		nvidia-xrun-pm
-----	--	----------------

If your hardware doesn't support `bbswitch`, you would need to use `nvidia-xrun-pmAUR` instead.

Now we need to blacklist *both* `nouveau` and `nvidia` kernel modules.

To do that, we first have to find out, where our active `modprobe.d` directory is located. There are 2 possible locations, generally speaking: `/etc/modprobe.d` and `/usr/lib/modprobe.d`. In my case it was the latter, which I could tell, because this directory already had files in it.

Now I'll create a new file named `nvidia-xrun.conf` and write the following into it

```
blacklist nvidia
blacklist nvidia-drm
blacklist nvidia-modeset
blacklist nvidia-uvmm
blacklist nouveau
```

Code-Listing 3.9:
`/usr/lib/modprobe.d/nvidia-xrun.conf`

With this config in place,

```
dustvoice in ~
> lsmod | grep nvidia
```

and

```
dustvoice in ~
> lsmod | grep nouveau
```

should return no output. Else you might have to place some additional entries into the file.

NOTE

Of course, you'll need to reboot, after blacklisting the modules and before issuing the 2 commands mentioned.

NOTE

If you installed `nvidia-xrun-pm` instead of `nvidia-xrun` and `bbswitch`, you might want to also enable the `nvidia-xrun-pm` service

```
dustvoice@dustArch ~
$ sudo systemctl enable nvidia-xrun-pm.service
```

NOTE

The required `.nvidia-xinitrc` file, mentioned previously, should already be provided in the `dotfiles` repository.

Now instead of `startx`, just run `nvidia-xrun`, enter your `sudo` password and you're good to go.

3.12 Additional fish software

Software that is useful in combination with a fish.

3.12.1 `tmux`

community | `tmux`

I would recommend to install `tmux` which enables you to have multiple terminal instances (called `windows` in `tmux`) open at the same time. This makes working with the linux terminal much easier.

NOTE

To view a list of keybinds, you just need to press `Ctrl+b` followed by `?`.

3.12.2 Communication

Life is all about communicating. Here are some pieces of software to do exactly that.

3.12.2.1 weechat

community | weechat

`weechat` is an IRC client for the terminal, with the best features and even a vim mode, by using a plugin

To configure everything, open `weechat`

```
dustvoice in ~  
> weechat
```

and install `vimode`, as well as configure it

```
/script install vimode.py  
/vimode bind_keys  
/set  
→ plugins.var.python.vimode.mode_indicator_normal_color_bg  
→ "blue"
```

Now add `mode_indicator+` in front of and `,[vi_buffer]` to the end of `weechat.bar.input.items`, in my case

```
/set weechat.bar.input.items  
→ "mode_indicator+[input_prompt]+(away),[input_search],[input_paste],input_text,[vi_buffer]"
```

Now add `,cmd_completion` to the end of `weechat.bar.status.items`, in my case

```
/set weechat.bar.status.items "[time],[buffer_last_number],[buffer_plugin],buffer_number+:+buffer_name+(buffer_modes  
→ )+{buffer_nicklist_count}+buffer_zoom+buffer_filter,scroll,[lag],[hotlist],completion,cmd_completion"
```

Now enable `vimode` searching

```
/set plugins.var.python.vimode.search_vim on
```

Now you just need to add a new connection, for example `irc.freenode.net`

```
/server add freenode irc.freenode.net
```

and connect to it

```
/connect freenode
```

NOTE

You might need to authenticate with NickServ, before being able to write in a channel

```
/msg NickServ identify <password>
```

NOTE

Instead of directly `/setting` the values specified above, you can also do

```
/fset weechat.var.name
```

after that, using the cursor, select the entry you want to modify (for example `plugins.var.python.vimode`) and then press `s` (make sure you're in `insert` mode) and `Return`, in order to modify the existing value.

3.12.3 PDF viewer

```
extra | ghostscript  
community | fbida
```

To use `asciidoctor-pdf`, you might be wondering how you are supposed to open the generated PDFs from the native linux fish.

This `fbida` package provides the `fbgs` software, which renders a PDF document using the native framebuffer.

To view this PDF document (`Documentation.pdf`) for example, you would run

```
dustvoice in ~  
> fbgs Documentation.pdf
```

NOTE

You can view all the controls by pressing `h`.

3.13 Additional hybrid software

Some additional software providing some kind of GUI to work with, but that can be useful in a `fish` only environment nevertheless.

3.13.1 Password management

I'm using `pass` as my password manager. As we already installed it in the [Additional required tools](#) step and updated the `submodule` that holds our `.password-store`, there is nothing left to do in this step

3.13.2 python

extra | python

Python has become really important for a magnitude of use cases.

3.13.3 ruby & asciidoctor

extra | ruby rubygems

In order to use `asciidoctor`, we have to install `ruby` and `rubygems`. After that we can install `asciidoctor` and all its required gems.

NOTE

If you want to have pretty and highlighted source code, you'll need to install a code formatter too.

For me there are mainly two options

- `pygments.rb`, which requires `python` to be installed

```
dustvoice in ~  
> gem install pygments.rb
```

- `rouge` which is a native ruby gem

```
dustvoice in ~  
> gem install rouge
```

Now the only thing left, in my case at least, is adding `~/.gem/ruby/2.7.0/bin` to your path.

NOTE

Please note that if you run a ruby version different from `2.7.0`, or if you upgrade your ruby version, you have to use the `bin` path for that version.

For `zsh` you'll want to add a new entry inside the `.zshpath` file

```
path+=("$HOME/.gem/ruby/2.7.0/bin")
```

Code-Listing 3.10: `~/.zshpath`

which then gets sourced by the provided `.zshenv` file. An example is provided with the `.zshpath.example` file

NOTE

You might have to re-> `source` the `.zshenv` file to make the changes take effect immediately

```
dustvoice in ~  
> source .zshenv
```

NOTE

If you want to add a new entry to the `path` variable, you have to append it to the array

```
path+=("$HOME/.gem/ruby/2.7.0/bin"  
↪ "$HOME/.gem/ruby/2.6.0/bin")
```

NOTE

If you use another shell than `zsh`, you might have to do something different, to add a directory to your `PATH`.

3.13.4 JUCE and FRUT

JUCE is a library for C++ that enables you to develop cross-platform applications with a single codebase.

FRUT makes it possible to manage JUCE projects purely from `cmake`.

NOTE

Note that apparently in the new JUCE version, `cmake` support is integrated. It remains to be seen how well this will work and if FRUT will become obsolete.

The information in this guide should be updated ASAP, if it is apparent that FRUT has now become obsolete.

```
dustvoice in ~
> git clone https://github.com/WeAreROLI/JUCE.git

dustvoice in ~
> cd JUCE

dustvoice in ~/JUCE
> git checkout develop

dustvoice in ~/JUCE
> cd ..

dustvoice in ~
> git clone https://github.com/McMartin/FRUT.git
```

3.13.4.1 Using JUCE

core	gcc gnutls
extra	alsa-lib clang freeglut freetype2 ladspa libx11 libxcomposite libxinerama libxrandr mesa webkit2gtk
community	jack2 libcurl-gnutls
multilib	lib32-freetype

In order to use JUCE, you'll need to have some dependency packages installed, where `ladspa` and `lib32-freetype` are not necessarily needed.

3.13.5 Additional development tools

Here are just some examples of development tools one could install in addition to what we already have.

3.13.5.1 Code formatting

community | `astyle`

We already have `clang-format` as a code formatter, but this only works for C-family languages. For java stuff, we can use `astyle`

3.13.5.2 Documentation

extra | `doxygen`

To generate a documentation from source code, I mostly use `doxygen`

3.13.5.3 Build tools

community | ninja

In addition to `make`, I'll often times use `ninja` for my builds

3.13.6 Android file transfer

```
extra | gvfs-mtp libmtp
```

Now you should be able to see your phone inside either your preferred filemanager, in my case `thunar`, or `gigolo`^{AUR}.

If you want to access the android's file system from the command line, you will need to either install and use `simple-mtpfs`^{AUR}, or `adb`

3.13.6.1 simple-mtpfs^{AUR}

AUR | simple-mtpfs

Edit `/etc/fuse.conf` to uncomment

```
user_allow_other
```

Code-Listing 3.11: `/etc/fuse.conf`

and mount the android device

```
dustvoice in ~
> simple-mtpfs -l

dustvoice in ~
> mkdir ~/mnt

dustvoice in ~
> simple-mtpfs --device <number> ~/mnt -allow_other
```

and respectively unmount it

```
dustvoice in ~
> fusermount -u mnt

dustvoice in ~
> rmdir mnt
```

3.13.6.2 adb

community | android-tools

Kill the adb server, if it is running

```
dustvoice in ~  
> adb kill-server
```

NOTE

If the server is currently not running, `> adb` will output an error with a `Connection refused` message.

Now connect your phone, unlock it and start the adb server

```
dustvoice in ~  
> adb start-server
```

If the PC is unknown to the android device, it will display a confirmation dialog. Accept it and ensure that the device was recognized

```
dustvoice in ~  
> adb devices
```

Now you can push/pull files.

```
dustvoice in ~  
> adb pull /storage/emulated/0/DCIM/Camera/IMG.jpg .  
  
dustvoice in ~  
> adb push IMG.jpg /storage/emulated/0/DCIM/Camera/IMG2.jpg  
  
dustvoice in ~  
> adb kill-server
```

NOTE

Of course you would need to have the *developer options* unlocked, as well as the *USB debugging* option enabled within them, for `adb` to even work.

3.13.7 Partition management

extra | gparted parted

You may also choose to use a graphical partitioning software instead of `fdisk` or `cfdisk`. For that you can use `gparted`. Of course there is also the fish equivalent `parted`.

3.13.8 PDF viewer

extra		evince
community		zathura zathura-pdf-mupdf

To use `asciidocctor-pdf`, you might be wondering how you are supposed to open the generated PDFs using the GUI.

The software `zathura` has a minimalistic design and UI with a focus on vim keybinding, whereas `evince` is a more desktop like experience, with things like a print dialogue, etc.

3.13.9 Process management

```
extra | htop xfce4-taskmanager
```

The native tool is `top`.

The next evolutionary step would be `htop`, which is an improved version of `top` (like `vi` and `vim` for example)

If you prefer a GUI for that kind of task, use `xfce4-taskmanager`.

3.13.10 Video software

Just some additional software related to videos.

3.13.10.1 Live streaming a terminal session

community | tmate

For this task, you'll need a program called `tmate`.

3.14 Additional GUI software

As you now have a working graphical desktop environment, you might want to install some software to utilize your newly gained power.

3.14.1 Session Lock

community | xsecurelock xss-lock

Probably the first thing you'll want to set up is a session locker, which locks your X-session after resuming from sleep, hibernation, etc. It then requires you to input your password again, so no unauthorized user can access you machine.

I'll use `xss-lock` to hook into the necessary `systemd` events and then use `xsecurelock` as my locker.

IMPORTANT

You need to make sure this command gets executed upon start of the X-session, so hook it into your window manager startup script, or in a file called by your desktop environment

```
dustvoice in ~  
> xss-lock -l -- xsecurelock &
```

3.14.2 `xfce-polkit`^{AUR}

AUR | `xfce-polkit`

In order for GUI applications to acquire `sudo` permissions, we need to install a `PolicyKit` authentication agent.

We could use `gnome-polkit` for that purpose, which resides inside the official repositories, but I decided on using `xfce-polkit`^{AUR}.

Now you just need to startup `xfce-polkit`^{AUR} before trying to execute something like `gparted` and you'll be prompted for your password.

As I already launch it as a part of my `bspwm` configuration, I won't have to worry about that.

3.14.3 Desktop background

```
extra | nitrogen
```

You might want to consider installing **nitrogen**, in order to be able to set a background image

3.14.4 Compositing software

community | picom

To get buttery smooth animation as well as e.g. smooth video playback in **brave** without screen tearing, you might want to consider using a compositor, in my case one named **picom**

WARNING

In order for **obs**' screen capture to work correctly, you need to kill **picom** completely before using **obs**.

```
dustvoice in ~  
> killall picom
```

or

```
dustvoice in ~  
> ps aux | grep picom  
  
dustvoice in ~  
> kill -9 <pid>
```

3.14.5 networkmanager applet

extra | network-manager-applet

To install the `NetworkManager` applet, which lives in your tray and provides you with a quick method to connect to different networks, you have to install the `network-manager-applet` package

Now you can start the applet with

```
dustvoice in ~  
> nm-applet &
```

If you want to edit the network connections with a more full screen approach, you can also launch `> nm-connection-editor`.

NOTE

The `nm-connection-editor` doesn't search for available Wi-Fis. You would have to set up a Wi-Fi connection completely by hand, which could be desirable depending on how difficult it is to set up your Wi-Fi.

3.14.6 Show keyboard layout

AUR | `xkblayout-state`

To show, which keyboard layout and variant is currently in use, you can use `xkblayout-state`^{AUR}

Now simply issue the `layout` alias, provided by my custom `zsh` configuration.

3.14.7 X clipboard

extra | xclip

To copy something from the terminal to the `xorg` clipboard, use `xclip`

3.14.8 Taking screen shots

community | scrot

For this functionality, especially in combination with `rofi`, use `scrot`.

> `scrot $HOME/Pictures/filename.png` then saves the screen shot under `$HOME/Pictures/filename.png`.

3.14.9 Image viewer

extra | ristretto

Now that we can create screen shots, we might also want to view those

```
dustvoice in ~  
> ristretto filename.png
```

3.14.10 File manager

```
extra | gvfs thunar
      |
AUR   | gigolo
```

You probably also want to use a file manager. In my case, `thunar`, the `xfce` file manager, worked best.

To also be able to mount removable drives, without being `root` or using `sudo`, and in order to have a GUI for mounting stuff, you would need to use `gigolo`^{AUR} and `gvfs`.

3.14.11 Archive manager

extra		cpio unrar unzip zip
community		xarchiver

As we now have a file manager, it might be annoying, to open up a terminal every time you simply want to extract an archive of some sort. That's why we'll use `xarchiver`.

3.14.12 Web browser

extra	firefox firefox-i18n-en-us
community	browserpass

As you're already using a GUI, you also might be interested in a web browser. In my case, I'm using **firefox**, as well as **browserpass** from the official repositories, together with the [uBlock Origin](#)¹, [Dark Reader](#)², [DuckDuckGo Privacy Essentials](#)³, [Vimium](#)⁴ and finally [Browserpass](#)⁵ add-ons, in order to use my passwords in **firefox** and have best protection in regard to privacy, while browsing the web.

We still have to setup **browserpass**, after installing all of this

```
dustvoice in ~
> cd /usr/lib/browserpass

dustvoice in /usr/lib/browserpass
> make hosts-firefox-user

dustvoice in /usr/lib/browserpass
> cd ~
```

¹<https://addons.mozilla.org/en-US/firefox/addon/ublock-origin/>

²<https://addons.mozilla.org/en-US/firefox/addon/darkreader/>

³<https://addons.mozilla.org/en-US/firefox/addon/duckduckgo-for-firefox/>

⁴<https://addons.mozilla.org/en-US/firefox/addon/vimium-ff/>

⁵<https://addons.mozilla.org/en-US/firefox/addon/browserpass-ce/>

3.14.12.1 Entering the dark side

AUR | `tor-browser`

You might want to be completely anonymous whilst browsing the web at some point. Although this shouldn't be your only precaution, using `tor-browser`^{AUR} would be the first thing to do

NOTE

You might have to check out how to import the `gpg` keys on the AUR page of `tor-browser`.

3.14.13 Office utilities

```
extra | libreoffice-fresh
```

I'll use `libreoffice-fresh` for anything that I'm not able to do with `neovim`.

3.14.13.1 Printing

```
extra | avahi cups cups-pdf nss-mdns  
      | print-manager  
      | system-config-printer
```

In order to be able to print from the gtk print dialog, we'll also need `system-config-printer` and `print-manager`.

```
dustvoice in ~  
> sudo systemctl enable avahi-daemon.service  
  
dustvoice in ~  
> sudo systemctl start avahi-daemon.service
```

Now you have to edit `/etc/nsswitch.conf` and add `mdns4_minimal [NOTFOUND=return]`

```
hosts: files mymachines myhostname mdns4_minimal  
↪ [NOTFOUND=return] resolve [!UNAVAIL=return] dns
```

Code-Listing 3.12: `/etc/nsswitch.conf`

Now continue with this

```
dustvoice in ~  
> avahi-browse --all --ignore-local --resolve --terminate  
  
dustvoice in ~  
> sudo systemctl enable org.cups.cupsd.service  
  
dustvoice in ~  
> sudo systemctl start org.cups.cupsd.service
```

Just open up `system-config-printer` now and configure your printer.

To test if everything is working, you could open up `brave`, then go to *Print* and then try printing.

3.14.14 Communication

Life is all about communicating. Here are some pieces of software to do exactly that.

3.14.14.1 Email

extra | thunderbird

There is nothing better than some classical email.

3.14.14.2 Telegram

community | telegram-desktop

You want to have your telegram messages on your desktop PC?

3.14.14.3 TeamSpeak 3

community | teamspeak3

Wanna chat with your gaming friends and they have a `teamspeak3` server?

3.14.14.4 Discord

community | discord

You'd rather use discord?

3.14.15 Video software

Just some additional software related to videos.

3.14.15.1 Viewing video

extra | vlc

You might consider using `vlc`

3.14.15.2 Creating video

```
AUR | obs-linuxbrowser-bin  
    | obs-glcapture-git  
    | obs-studio-git
```

`obs-studio-git`^{AUR} should be the right choice.

You can also make use of the plugins provided in the package list above.

Showing keystrokes

```
AUR | screenkey
```

In order to show the viewers what keystrokes you're pressing, you can use something like `screenkey`^{AUR}

NOTE

For ideal use with `obs`, my `dotfiles` repository already provides you with the `> screenkey-obs` alias for you to run with `zsh`.

3.14.15.3 Editing video

AUR | `davinci-resolve`

In my case, I'm using `davinci-resolveAUR`.

3.14.15.4 Utilizing video

AUR | `teamviewer`

Wanna remote control your own or another PC?
`teamviewer`^{AUR} might just be the right choice for you

3.14.16 Audio Production

You might have to edit `/etc/security/limits.conf`, to increase the allowed locked memory amount.

In my case I have 32GB of RAM and I want the `audio` group to be able to allocate most of the RAM, which is why I added the following line to the file

```
@audio - memlock 29360128
```

Code-Listing 3.13: `/etc/security/limits.conf`

3.14.16.1 Ardour

community | ardour

To e.g. edit and produce audio, you could use **ardour**, because it's easy to use, stable and cross platform.

NOTE

extra | ffmpeg

Ardour won't natively save in the **mp3** format, due to licensing stuff. In order to create **mp3** files, for sharing with other devices, because they have problems with **wav** files, for example, you can just use **ffmpeg**.
and after that we're going to convert **in.wav** to **out.mp3**

```
dustvoice in ~  
> ffmpeg -i in.wav -acodec mp3 out.mp3
```

3.14.16.2 Reaper

AUR | [reaper-bin](#)

Instead of **ardour**, I'm using **reaper**, which is available for linux as a beta version, in my case more stable than **ardour** and more easy to use for me.

3.14.17 Virtualization

community		virtualbox
		virtualbox-host-modules-arch

You might need to run another OS, for example Mac OS, from within Linux, e.g. for development/testing purposes. For that you can use `virtualbox`.

Now when you want to use `virtualbox` just load the kernel module

```
dustvoice in ~  
> sudo modprobe vboxdrv
```

and add the user which is supposed to run `> virtualbox` to the `vboxusers` group

```
dustvoice in ~  
> sudo usermod -a G vboxusers $USER
```

and if you want to use `rawdisk` functionality, also to the `disk` group

```
dustvoice in ~  
> sudo usermod -a G disk $USER
```

Now just re-login and you're good to go.

3.14.18 Gaming

extra	pulseaudio pulseaudio-alsa
community	lutris
multilib	lib32-libpulse
	lib32-nvidia-utils steam

The first option for native/emulated gaming on Linux is obviously **steam**.

The second option would be **lutris**, a program, that configures a wine instance correctly, etc.

3.14.19 Wacom

```
extra | libwacom xf86-input-wacom
```

In order to use a Wacom graphics tablet, you'll have to install some packages
You can now configure your tablet using the `xsetwacom` command.

3.14.20 VNC & RDP

extra		libvncserver
community		remmina
AUR		freerdp

In order to connect to a machine over VNC or to connect to a machine using the Remote Desktop Protocol, for example to connect to a Windows machine, I'll need to install `freerdp`^{AUR}, as well as `libvncserver`, for RDP and VNC functionality respectively, as well as `remmina`, to have a GUI client for those two protocols.

Now you can set up all your connections inside `remmina`.

Chapter 4

Upgrading the system

You're probably wondering why this gets a dedicated section.

You'll probably think that it would be just a matter of issuing

```
dustvoice in ~  
> sudo pacman -Syu
```

That's both true and false.

You have to make sure, *that your boot partition is mounted at /boot* in order for everything to upgrade correctly. That's because the moment you upgrade the `linux` package without having the correct partition mounted at `/boot`, your system won't boot. You also might have to do `> grub-mkconfig -o /boot/grub/grub.cfg` after you install a different kernel image.

If your system *indeed doesn't boot and boots to a recovery fish*, then double check that the issue really is the not perfectly executed kernel update by issuing

```
root in ~  
> uname -a
```

and

```
root in ~  
> pacman -Q linux
```

The version of these two packages should be exactly the same!

If it isn't there is an easy fix for it.

4.1 Fixing a faulty kernel upgrade

First off we need to restore the old `linux` package.

For that note the version number of

```
root in ~
> uname -a
```

Now we'll make sure first that nothing is mounted at `/boot`, because the process will likely create some unwanted files. The process will also create a new `/boot` folder, which we're going to delete afterwards.

```
root in ~
> umount /boot
```

Now `cd` into `pacman`'s package cache

```
root in ~
> cd /var/cache/pacman/pkg
```

There should be a file located named something like `linux-<version>.pkg.tar.xz`, where `<version>` would be somewhat equivalent to the previously noted version number

Now downgrade the `linux` package

```
root in ~
> pacman -U linux-<version>.pkg.tar.xz
```

After that remove the possibly created `/boot` directory

```
root in ~
> rm -rf /boot

root in ~
> mkdir /boot
```

Now reboot and mount the `boot` partition, in my case an EFI System partition.

Now simply rerun

```
dustvoice in ~  
> sudo pacman -Syu
```

and you should be fine now.

Chapter 5

Additional notes

If you've printed this guide, you might want to add some additional blank pages for notes.