



DustArch

DustVoice's Arch Linux from scratch

David Holland

Version 8.3, 2020-02-03

Table Of Contents

| | |
|--|----|
| 1. Inside the archiso | 1 |
| 1.1. Syncing up pacman | 2 |
| 1.1.1. Official repositories | 3 |
| 1.1.2. AUR | 4 |
| 1.1.3. Software categories | 6 |
| 1.1.4. Software installation | 7 |
| 1.1.4.1. Example section | 8 |
| 1.2. Formatting the drive | 9 |
| 1.3. Preparing the chroot environment | 11 |
| 2. Entering the chroot | 12 |
| 2.1. Installing additional packages | 13 |
| 2.2. Master of time | 15 |
| 2.3. Master of locales | 16 |
| 2.4. Naming your machine | 17 |
| 2.4.1. hostname | 18 |
| 2.4.2. hosts | 19 |
| 2.5. User setup | 20 |
| 2.5.1. Give root a password | 21 |
| 2.5.2. Create a personal user | 22 |
| 2.6. grub | 24 |
| 2.6.1. BIOS | 25 |
| 2.6.2. UEFI | 26 |
| 2.6.3. grub config | 27 |
| 2.6.3.1. Adjust the timeout | 28 |
| 2.6.3.2. Enable the recovery | 29 |

| | |
|--|----|
| 2.6.3.3. NVIDIA fix | 30 |
| 2.6.3.4. Add power options | 31 |
| 2.6.3.5. Installing mementest | 32 |
| 2.6.3.5.1. BIOS | 33 |
| 2.6.3.5.2. UEFI | 34 |
| 2.6.3.6. Enabling hibernation | 35 |
| 2.6.3.7. Generating the grub config | 36 |
| 2.7. Secure Boot | 37 |
| 2.7.1. shim | 38 |
| 2.7.2. The manual way | 40 |
| 3. Inside the DustArch | 41 |
| 3.1. Someone there? | 42 |
| 3.2. Update and upgrade | 45 |
| 3.3. Enabling the multilib repository | 46 |
| 3.4. zsh for president | 47 |
| 3.5. git | 48 |
| 3.6. Security is important | 49 |
| 3.6.1. Smartcard shenanigans | 50 |
| 3.7. Additional required tools | 51 |
| 3.8. Setting up a home environment | 52 |
| 3.8.1. Use dotfiles for a base config | 54 |
| 3.8.2. Set up gpg | 55 |
| 3.8.3. Finalize the dotfiles | 57 |
| 3.8.4. gpg-agent forwarding | 58 |
| 3.8.5. Back to your roots | 60 |
| 3.9. fstab | 61 |
| 3.10. Audio | 63 |

| | |
|--|-----|
| 3.10.1. alsa | 64 |
| 3.10.2. pulseaudio | 65 |
| 3.10.3. jack | 67 |
| 3.10.4. Audio handling | 68 |
| 3.11. Bluetooth | 69 |
| 3.12. Graphical desktop environment | 73 |
| 3.12.1. NVIDIA | 74 |
| 3.12.2. Launching the graphical environment | 75 |
| 3.12.2.1. The NVIDIA way | 76 |
| 3.13. Additional console software | 79 |
| 3.13.1. tmux | 80 |
| 3.13.2. Communication | 81 |
| 3.13.2.1. weechat | 82 |
| 3.13.3. PDF viewer | 85 |
| 3.14. Additional hybrid software | 86 |
| 3.14.1. Password management | 87 |
| 3.14.2. python | 88 |
| 3.14.3. ruby & asciidoctor | 89 |
| 3.14.4. JUCE and FRUT | 92 |
| 3.14.4.1. Using JUCE | 93 |
| 3.14.5. Additional development tools | 94 |
| 3.14.5.1. Code formatting | 95 |
| 3.14.5.2. Documentation | 96 |
| 3.14.5.3. Build tools | 97 |
| 3.14.6. Android file transfer | 98 |
| 3.14.6.1. simple-mtpfs ^{AUR} | 99 |
| 3.14.6.2. adb | 100 |

| | |
|---|-----|
| 3.14.7. Partition management | 102 |
| 3.14.8. PDF viewer | 103 |
| 3.14.9. Process management | 104 |
| 3.14.10. Video software | 105 |
| 3.14.10.1. Live streaming a terminal session | 106 |
| 3.15. Additional GUI software | 107 |
| 3.15.1. Session Lock | 108 |
| 3.15.2. <code>xfce-polkit</code> ^{AUR} | 109 |
| 3.15.3. Desktop background | 110 |
| 3.15.4. Compositing software | 111 |
| 3.15.5. <code>networkmanager</code> applet | 112 |
| 3.15.6. Show keyboard layout | 113 |
| 3.15.7. X clipboard | 114 |
| 3.15.8. Taking screen shots | 115 |
| 3.15.9. Image viewer | 116 |
| 3.15.10. File manager | 117 |
| 3.15.11. Archive manager | 118 |
| 3.15.12. Web browser | 119 |
| 3.15.12.1. Entering the dark side | 120 |
| 3.15.13. Office utilities | 121 |
| 3.15.13.1. Printing | 122 |
| 3.15.14. Communication | 124 |
| 3.15.14.1. Email | 125 |
| 3.15.14.2. Telegram | 126 |
| 3.15.14.3. TeamSpeak 3 | 127 |
| 3.15.14.4. Discord | 128 |
| 3.15.15. Video software | 129 |

| | |
|---|-----|
| 3.15.15.1. Viewing video | 130 |
| 3.15.15.2. Creating video | 131 |
| 3.15.15.2.1. Showing keystrokes | 132 |
| 3.15.15.3. Editing video | 133 |
| 3.15.15.4. Utilizing video | 134 |
| 3.15.16. Audio Production | 135 |
| 3.15.16.1. Ardour | 135 |
| 3.15.16.2. Reaper | 136 |
| 3.15.17. Virtualization | 137 |
| 3.15.18. Gaming | 138 |
| 3.15.19. Wacom | 139 |
| 3.15.20. VNC & RDP | 140 |
| 4. Upgrading the system | 141 |
| 4.1. Fixing a faulty kernel upgrade | 143 |
| 5. Additional notes | 145 |

1. Inside the `archiso`

This section is aimed at providing help with the general installation of a customized Arch Linux from within an official Arch Linux image (`archiso`).



As Arch Linux is a rolling release 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` only has very few packages available from cache.

Furthermore, one should bear in mind that depending on the version, or rather modification date, the guide 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](#), or the [Arch Linux forums](#).

1.1. Syncing up pacman

First of all we need to sync up `pacman`'s package repository, in order to be able to install packages

```
root@archiso ~ # pacman -Sy
```



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@archiso ~ # pacman -S <package_name>
```

where you would replace `<package_name>` with the actual 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@archiso ~ # pacman -Ss <keyword>
```

If you want to remove an installed package, just use

```
root@archiso ~ # pacman -Rsu <package_name>
```



If you have to force remove, which you should use **with extreme caution**, you can use

```
root@archiso ~ # pacman -Rdd  
<package_name>
```

1.1.2. AUR

If you want to install a package from the **AUR**, I would proceed as follows

1. **cd** into the dedicated directory, if you're using the **dotfiles** repo, which provides an **update.sh** script within that folder, to check every subfolder for updates

```
dustvoice@archiso ~ $ cd AUR
```

2. Clone the package with **git**

```
dustvoice@archiso ~/AUR $ git clone  
https://aur.archlinux.org/pacman-git.git
```

3. Switch to the package directory

```
dustvoice@archiso ~/AUR $ cd pacman-git
```

4. Execute **makepkg**

```
dustvoice@archiso ~/AUR/pacman-git $ makepkg -si
```

5. Delete all files created by **makepkg**, in order to easily see, if a package needs an update by using **git fetch --all** and **git status**
-

```
dustvoice@archiso ~/AUR/pacman-git $ git reset HEAD  
--hard  
dustvoice@archiso ~/AUR/pacman-git $ git clean -fdx
```



You might have to resolve any **AUR** dependencies, which can't be resolved with **pacman**.



In order to install that **AUR** package, you **must** switch to your normal user, because **makepkg** doesn't run as root.

1.1.3. Software categories

In this guide, I'll be marking some headings according to which kind of software it uses.

There are three categories of software:

- **Console** software is intended to be used with either the native linux console, or with a terminal emulator
 - **GUI** software is intended to be used in a graphical desktop environment
 - **Hybrid** software can either be used within both a console and a graphical desktop environment (**networkmanager**), or there are packages available for both console and a graphical desktop environment (**pulseaudio** with **pulsemixer** for ^{console} and **pavucontrol** for ^{GUI})
-

1.1.4. Software installation

In this guide, I'll be explicitly mark the packages installed in a specific section.

This enables 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 have to read through bloating lines like

```
dustvoice@DustArch ~  
$ sudo pacman -S some-package
```

- not have to accidentally reinstall already installed packages



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 a explanation there.

Of course, you can adapt everything to your needs, especially in the [\[additional-setup-packages\]](#) step.

1.1.4.1. Example section

| Software Packages | |
|-------------------|---|
| 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`

`/etc/sample.conf`

```
Sample.text=useful
```

1.2. Formatting the drive

First you have to list all the available drives by issuing

```
root@archiso ~ # fdisk -l
```



The output of `fdisk -l` is dependent on your system configuration.

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



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

With bigger RAM sizes available today, this isn't necessary anymore. 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.

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.



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

Now we need to format the partitions accordingly

```
root@archiso ~ # mkfs.ext4 /dev/sdb2  
root@archiso ~ # mkswap /dev/sdb3
```

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

```
root@archiso ~ # swapon /dev/sdb3  
root@archiso ~ # mount /dev/sdb2 /mnt
```



If you have an additional **EFI system partition**, because of a *UEFI - GPT* setup or e.g. an existing Windows installation, 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 systems **/boot** folder

```
root@archiso ~ # mkdir /mnt/boot  
root@archiso ~ # mount /dev/sda2  
/mnt/boot
```


1.3. Preparing the **chroot** environment

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

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

| Software Packages | |
|-------------------|--|
| <code>core</code> | <code>base linux linux-firmware</code> |



This is the actual command used in my case

```
root@archiso ~ # pacstrap /mnt base  
linux linux-firmware
```

After that generate an **fstab** using **genfstab**

```
root@archiso ~ # genfstab -U /mnt >> /mnt/etc/fstab
```

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

2. Entering the `chroot`



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 already shipped with the `archiso`. This script takes care of all that stuff, so we can set up our system properly.

```
root@archiso ~ # arch-chroot /mnt
```

Et Voila! You successfully `chrooted` inside your new system and you'll be greeted by a `bash` prompt.

2.1. Installing additional packages

| Software 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 |



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.

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

```
[root@archiso /]# systemctl enable
NetworkManager.service
```

With `polkit` installed, create a file `/etc/polkit-1/rules.de/50-org.freedesktop.NetworkManager.rules` to enable users of the `network` group to add new networks without the need of `sudo`.

/etc/polkit-1/rules.de/50-org.freedesktop.NetworkManager.rules

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

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 resides under `/usr/share/zoneinfo/Europe/Berlin`.

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

```
[root@archiso /]# ln -s  
/usr/share/zoneinfo/Europe/Berlin /etc/localtime  
[root@archiso /]# hwclock --systohc --utc
```

Now you can also enable time synchronization over network

```
[root@archiso /]# timedatectl set-timezone  
Europe/Berlin  
[root@archiso /]# timedatectl set-ntp true  
[root@archiso /]# timedatectl status
```

and check that everything is alright

```
[root@archiso /]# 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.



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

/etc/locale.gen

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

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

```
[root@archiso /]# locale-gen
```

and set the locale

```
[root@archiso /]# 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` and add `hosts` entries.

Apart from being mentioned in your command prompt, the `hostname` also serves the purpose of identifying, or naming your machine. This enables you to see your PC in your router, etc.

2.4.1. `hostname`

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

/etc/hostname

```
DustArch
```

2.4.2. `hosts`

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

/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
```

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@archiso /]# passwd
```

and choose a new password.

2.5.2. Create a personal user

| Software Packages | |
|-------------------|------|
| core | sudo |
| extra | zsh |

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

```
[root@archiso /]# useradd -m -p "" -G  
"adm,audio,disk,floppy,kvm,log,lp,network,rfskill,scanner,  
storage,users,optical,power,wheel" -s /usr/bin/zsh  
dustvoice  
[root@archiso /]# passwd dustvoice
```

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

For that we edit **/etc/sudoers** and uncomment the **%wheel** line

/etc/sudoers

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

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

/etc/sudoers

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

with your new username

/etc/sudoers

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

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

2.6. grub

| Software Packages | |
|-------------------|-----------------|
| core | efibootmgr grub |

Now onto installing the boot manager. We will use **grub** in this guide.

First make sure, all the required packages are installed

```
[root@archiso /]# pacman -S grub dosfstools os-prober  
mtools
```

and if you want to use **UEFI**, also

```
[root@archiso /]# pacman -S efibootmgr
```

2.6.1. 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@archiso /]# grub-install --target=i386-pc  
/dev/sdb
```



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. 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@archiso /]# grub-install --target=x86_64-efi  
--efi-directory=/boot --bootloader-id=grub --recheck
```

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@archiso /]# ls /boot/EFI/grub
```

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

2.6.3. 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.

2.6.3.1. Adjust the timeout

First of all, I want my **grub** menu to wait indefinitely for my command to boot an OS.

/etc/default/grub

```
GRUB_TIMEOUT=-1
```



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.

/etc/default/grub

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

2.6.3.2. 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.

/etc/default/grub

```
GRUB_DISABLE_RECOVERY=false
```

2.6.3.3. 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

/etc/default/grub

```
GRUB_GFXPAYLOAD_LINUX=text
```

2.6.3.4. 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.

/etc/grub.d/40-custom

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

2.6.3.5. 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.

2.6.3.5.1. BIOS

| Software Packages | |
|-------------------|------------|
| extra | memtest86+ |

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

2.6.3.5.2. UEFI

| Software Packages | |
|-------------------|---------------|
| 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@archiso /]# memtest86-efi -i
```

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

2.6.3.6. Enabling hibernation

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

After that we need to perform two tasks

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

/etc/mkinitcpio.conf

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

2. Add the **resume** kernel parameter to **/etc/default/grub**, containing my **swap** partition **UUID**, in my case

/etc/default/grub

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

After that we have to run

```
[root@archiso /]# mkinitcpio -p linux
```



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 [Generating the grub config](#).

2.6.3.7. Generating the grub config

Now we can finally generate our grub.cfg

```
[root@archiso /]# grub-mkconfig -o /boot/grub/grub.cfg
```

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

2.7. Secure Boot

2.7.1. shim

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](#).

| Software Packages | |
|-------------------|-------------|
| AUR | shim-signed |

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@archiso /]# cp /usr/share/shim-signed/shimx64.efi /boot/EFI/grub/  
[root@archiso /]# cp /usr/share/shim-signed/mmx64.efi /boot/EFI/grub/
```



If you have to use `bcdedit` from within Windows, as explained previously, 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 bootloader or kernel.

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

Reboot and your system should fire up just fine.

2.7.2. The manual way

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.



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

```
DustArch% ip link
```

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

```
DustArch% 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

```
DustArch% sudo systemctl restart  
NetworkManager.service
```

and then try **ping**ing again.

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



I never got `nmtool` 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

```
DustArch% nmcli radio
```

and if not, enable it

```
DustArch% nmcli radio wifi on
```

Now make sure your Wi-Fi interface appears under

```
DustArch% nmcli device
```

Rescan for available networks

```
DustArch% nmcli device wifi rescan
```

and list all found networks

```
DustArch% nmcli device wifi list
```

After that connect to the network

```
DustArch% 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

```
DustArch% 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

/etc/pacman.conf

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

and update **pacman**'s package repositories afterwards

```
DustArch% 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.



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.

```
DustArch% 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

| Software Packages | |
|-------------------|-----|
| 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

| Software Packages | |
|-------------------|-------|
| 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

| Software Packages | |
|-------------------|-----------------------|
| extra | libusb-compat |
| community | ccid opencsc pcsclite |

```
DustArch% sudo pacman -S pcsclite libusb-compat ccid opencsc
```

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

```
DustArch% sudo systemctl enable pcscd.service
DustArch% sudo systemctl start pcscd.service
```

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

```
DustArch% gpg --card-status
```



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

| Software Packages | |
|-------------------|-----------------------------------|
| core | make openssh |
| extra | clang cmake jdk-openjdk python |
| community | pass |

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.

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.



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

```
DustArch% git init
DustArch% git remote add origin
https://github.com/DustVoice/dotfiles.git
DustArch% git fetch
DustArch% git reset origin/master --hard
DustArch% 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@DustArch ~  
$ chmod 700 .gnupg  
dustvoice@DustArch ~  
$ gpg --card-status  
dustvoice@DustArch ~  
$ gpg --card-edit
```

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

```
dustvoice@DustArch ~  
$ gpg-connect-agent updatestartuptty /bye
```



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@DustArch ~  
$ git remote set-url origin  
git@github.com: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@DustArch ~  
$ exit
```

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@DustArch ~
$ git submodule update --init --recursive
dustvoice@DustArch ~
$ source .zshrc
dustvoice@DustArch ~
$ cd .config/nvim
dustvoice@DustArch ~/.config/nvim
$ echo 'let g:platform = "linux"' >> platform.vim
dustvoice@DustArch ~/.config/nvim
$ echo 'let g:use_autocomplete = 3' >> custom.vim
dustvoice@DustArch ~/.config/nvim
$ echo 'let g:use_clang_format = 1' >> custom.vim
dustvoice@DustArch ~/.config/nvim
$ echo 'let g:use_font = 0' >> custom.vim
dustvoice@DustArch ~/.config/nvim
$ sudo pip3 install neovim
dustvoice@DustArch ~/.config/nvim
$ nvim --headless +PlugInstall +qa
dustvoice@DustArch ~/.config/nvim
$ cd plugged/YouCompleteMe
dustvoice@DustArch
~/.config/nvim/plugged/YouCompleteMe
$ python3 install.py --clang-completer --java
-completer
dustvoice@DustArch
~/.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.

~/.ssh/config

```
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>
```



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@DustArch ~
$ !gpgconf --list-dirs
```

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

machine(s).

/etc/ssh/sshd_config

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

Now just restart your remote machine(s) and you're ready to go.

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@DustArch ~  
$ sudo -iu root
```

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



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. fstab

In my case, I'm sharing an **exFat** partition between my **DustArch** and my Windows. This was a result of some major inconvenience because of some weird **NTFS** permission stuff, which apparently Windows didn't like. Since I've avoided directly writing to Windows partitions since then, I'll quickly demonstrate what **fstab** entries I have and why

/etc/fstab

```
1 UUID=e26de048-6147-42e5-a34b-59f1a50621bb      /
  ext4              rw,relatime                    0 1
2
3 UUID="C8E3-A0FD"
  /boot             vfat                            defaults
  0 1
4
5 UUID="DC88-5A4E"
  /mnt/projects     exfat                            rw,relatime
  0 0
6
7 UUID=7A16569B51903310
  /mnt/data         ntfs
  ro,nosuid,nodev,noauto 0 0
```

The

1. entry should be pretty straight forward. It's my root partition of my **DustArch** install.
2. entry is quite important too. It's my **EFI System Partition**, which gets mounted at boot time, in order to prevent kernel

orphaning, which means, that the kernel version installed on the system doesn't match the one on the **boot** partition.

3. entry is my shared **exFat** partition, which we are allowed to write to.
4. entry is important, because of the options. These options prevent me from modifying files on that **NTFS** partition.

3.10. Audio

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

3.10.1. **alsa**

| Software Packages | |
|-------------------|-------------------|
| extra | alsa-utils |



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

Now choose the sound card you want to use

```
dustvoice@DustArch ~  
$ cat /proc/asound/cards
```

and then create **/etc/asound.conf**

/etc/asound.conf

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



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

3.10.2. pulseaudio

| Software Packages | |
|-------------------|------------------------|
| 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

/etc/pulse/daemon.conf

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

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

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

/etc/pulse/daemon.conf

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

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

```
dustvoice@DustArch ~  
$ pulseaudio --kill  
dustvoice@DustArch ~  
$ pulseaudio --start
```

3.10.3. jack

| Software Packages | |
|-------------------|-----------------|
| 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**.

3.10.4. Audio handling

| Software Packages | |
|-------------------|---|
| 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@DustArch ~  
$ play audio.wav  
dustvoice@DustArch ~  
$ play audio.mp3
```

to play audio.

3.11. Bluetooth

| Software Packages | |
|-------------------|---------------------------------------|
| extra | bluez bluez-util 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@DustArch ~  
$ sudo lsmod | grep btusb
```

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

```
dustvoice@DustArch ~  
$ sudo systemctl enable bluetooth.service  
dustvoice@DustArch ~  
$ sudo systemctl start bluetooth.service
```



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@DustArch ~  
$ 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
```



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>
```



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>
```

If your device is an audio device, of some kind you might have to install `pulseaudio-bluetooth` and append 2 lines to `/etc/pulse/system.pa` as well.

append the following 2 lines

/etc/pulse/system.pa



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

and restart `pulseaudio`

```
dustvoice@DustArch ~  
$ pulseaudio --kill  
dustvoice@DustArch ~  
$ pulseaudio --start
```

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

3.12. Graphical desktop environment

| Software Packages | |
|-------------------|--|
| extra | ttf-hack xorg xorg-drivers xorg-xinit |
| community | arandr alacritty i3 i3status rofi |

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

```
dustvoice@DustArch ~  
$ sudo pacman -S xorg xorg-xinit xorg-drivers i3  
i3status rofi ttf-hack xfce4-terminal arandr
```

3.12.1. NVIDIA

| Software Packages | |
|-------------------|---|
| 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



You would have to reboot sooner or later after installing the NVIDIA drivers.

Also to get the best performance, at least for something like screen capturing in `obs`, go to **X Server Display Configuration** inside `nvidia-settings`, switch to **Advanced** and enable **Force Composition Pipeline**, as well as **Force Full Composition Pipeline**.

3.12.2. 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`.

3.12.2.1. The NVIDIA way

| Software Packages | |
|-------------------|-------------|
| 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`).



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



| Software Packages | |
|-------------------|----------------|
| 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

```
/usr/lib/modprobe.d/nvidia-xrun.conf
```

```
1 blacklist nvidia
2 blacklist nvidia-drm
3 blacklist nvidia-modeset
4 blacklist nvidia-vm
5 blacklist nouveau
```

With this config in place,

```
dustvoice@DustArch ~
$ lsmod | grep nvidia
```

and

```
dustvoice@DustArch ~
$ lsmod | grep nouveau
```

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



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



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
```



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.

3.13. Additional console software

Software that is useful in combination with a console.

3.13.1. `tmux`

| Software Packages | |
|------------------------|-------------------|
| <code>community</code> | <code>tmux</code> |

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.



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

3.13.2. Communication

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

3.13.2.1. weechat

| Software Packages | |
|-------------------|---------|
| 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@DustArch ~  
$ 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
```



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

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

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



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

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

3.13.3. PDF viewer

| Software Packages | |
|-------------------|-------------|
| 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 console.

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@DustArch ~
$ fbgs Documentation.pdf
```

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

3.14. Additional hybrid software

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

3.14.1. 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

3.14.2. python

| Software Packages | |
|-------------------|--------|
| extra | python |

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

3.14.3. ruby & asciidoctor

| Software Packages | |
|-------------------|---------------|
| 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.



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@DustArch ~  
$ gem install pygments.rb
```

- **rouge** which is a native **ruby** gem

```
dustvoice@DustArch ~  
$ gem install rouge
```

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



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

~/.zshpath

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

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



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

```
dustvoice@DustArch ~  
$ source .zshenv
```

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



~/.zshpath

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




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

3.14.4. JUCE and FRUT

JUCE is a header only 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**.

```
dustvoice@DustArch ~  
$ git clone https://github.com/WeAreROLI/JUCE.git  
dustvoice@DustArch ~  
$ cd JUCE  
dustvoice@DustArch ~/JUCE  
$ git checkout develop  
dustvoice@DustArch ~/JUCE  
$ cd ..  
dustvoice@DustArch ~  
$ git clone https://github.com/McMartin/FRUT.git
```

3.14.4.1. Using JUCE

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

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

3.14.5. Additional development tools

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

3.14.5.1. Code formatting

| Software Packages | |
|-------------------|--------|
| 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`

3.14.5.2. Documentation

| Software Packages | |
|-------------------|---------|
| extra | doxygen |

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

3.14.5.3. Build tools

| Software Packages | |
|-------------------|-------|
| community | ninja |

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

3.14.6. Android file transfer

| Software Packages | |
|--------------------|------------------------------|
| <code>extra</code> | <code>gvfs-mtp libmtp</code> |

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`

3.14.6.1. `simple-mtpfs`^{AUR}

| Software Packages | |
|-------------------|---------------------------|
| AUR | <code>simple-mtpfs</code> |

Edit `/etc/fuse.conf` to uncomment

`/etc/fuse.conf`

```
user_allow_other
```

and mount the android device

```
dustvoice@DustArch ~  
$ simple-mtpfs -l  
dustvoice@DustArch ~  
$ mkdir ~/mnt  
dustvoice@DustArch ~  
$ simple-mtpfs --device <number> ~/mnt -allow_other
```

and respectively unmount it

```
dustvoice@DustArch ~  
$ fusermount -u mnt  
dustvoice@DustArch ~  
$ rmdir mnt
```

3.14.6.2. adb

| Software Packages | |
|-------------------|---------------|
| community | android-tools |

Kill the **adb** server, if it is running

```
dustvoice@DustArch ~  
$ adb kill-server
```



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@DustArch ~  
$ 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@DustArch ~  
$ adb devices
```

Now you can **push/pull** files.

```
dustvoice@DustArch ~  
$ adb pull /storage/emulated/0/DCIM/Camera/IMG.jpg .  
dustvoice@DustArch ~  
$ adb push IMG.jpg  
/storage/emulated/0/DCIM/Camera/IMG2.jpg  
dustvoice@DustArch ~  
$ adb kill-server
```



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.

3.14.7. Partition management

| Software Packages | |
|-------------------|----------------|
| 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 `console` equivalent ``parted`.

3.14.8. PDF viewer

| Software Packages | |
|-------------------|--------|
| extra | evince |
| community | mupdf |

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

If you want to have changes made to the PDF reflected immediately in the viewer, you would need `evince` instead of `mupdf`, but `mupdf` has a more minimalistic interface, which comes in handy when using a tiling window manager.

3.14.9. Process management

| Software Packages | |
|-------------------|------------------------|
| 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`.

3.14.10. Video software

Just some additional software related to videos.

3.14.10.1. Live streaming a terminal session

| Software Packages | |
|-------------------|-------|
| community | tmate |

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

3.15. 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.

3.15.1. Session Lock

| Software Packages | |
|-------------------|-----------------|
| community | i3lock 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 your machine.

I'll use **xss-lock** to hook into the necessary **systemd** events and **i3lock** as my locker.



I have placed the required command to start **xss-lock** with the right parameters inside my **i3** configuration file.

If you use something other than **i3**, you need to make sure this command gets executed upon start of the **X**-session

```
xss-lock -- i3lock -n -e -c 333333
```

3.15.2. `xfce-polkit`^{AUR}

| Software Packages | |
|-------------------|--------------------------|
| AUR | <code>xfce-polkit</code> |

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 `i3` configuration, I won't have to worry about that.

3.15.3. Desktop background

| Software Packages | |
|-------------------|----------|
| extra | nitrogen |

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

3.15.4. Compositing software

| Software Packages | |
|-------------------|-------|
| 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**



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

```
dustvoice@DustArch ~  
$ killall picom
```

or

```
dustvoice@DustArch ~  
$ ps aux | grep picom  
dustvoice@DustArch ~  
$ kill -9 <pid>
```

3.15.5. networkmanager applet

| Software Packages | |
|-------------------|------------------------|
| 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@DustArch ~  
$ nm-applet &
```

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



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 to set up your Wi-Fi is.

3.15.6. Show keyboard layout

| Software Packages | |
|-------------------|-----------------|
| 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.

3.15.7. X clipboard

Software Packages

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

3.15.8. Taking screen shots

| Software Packages | |
|-------------------|-------|
| community | scrot |

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

`scrot ~/Pictures/filename.png` then saves the screen shot under `~/Pictures/filename.png`.

3.15.9. Image viewer

| Software Packages | |
|-------------------|-----------|
| extra | ristretto |

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

```
dustvoice@DustArch ~  
$ ristretto filename.png
```

3.15.10. File manager

| Software Packages | |
|-------------------|-------------|
| 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`.

3.15.11. Archive manager

| Software Packages | |
|-------------------|----------------------|
| 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**.

3.15.12. Web browser

| Software Packages | |
|-------------------|-------------|
| AUR | brave-bin |
| community | browserpass |

As you're already using a GUI, you also might be interested in a web browser. In my case, I'll install **brave-bin**^{AUR}, as well as **browserpass** from the official repositories, in order to use my passwords in **brave**.

We still have to setup **browserpass**

```
dustvoice@DustArch ~  
$ cd /usr/lib/browserpass  
dustvoice@DustArch /usr/lib/browserpass  
$ make hosts-brave-user  
dustvoice@DustArch /usr/lib/browserpass  
$ make policies-brave-user  
dustvoice@DustArch /usr/lib/browserpass  
$ cd ~
```

Now the only thing left is, to fire up **brave** and install the **browserpass** extension from the chrome store.

3.15.12.1. Entering the dark side

| Software Packages | |
|-------------------|-------------|
| 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-browserAUR` would be the first thing to do



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

3.15.13. Office utilities

| Software Packages | |
|-------------------|-------------------|
| extra | libreoffice-fresh |

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

3.15.13.1. Printing

| Software Packages | |
|-------------------|---|
| 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@DustArch ~  
$ sudo systemctl enable avahi-daemon.service  
dustvoice@DustArch ~  
$ sudo systemctl start avahi-daemon.service
```

Now you have to edit **/etc/nsswitch.conf**

so this line

/etc/nsswitch.conf

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

becomes this line

/etc/nsswitch.conf

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

Now continue with this

```
dustvoice@DustArch ~  
$ avahi-browse --all --ignore-local --resolve  
--terminate  
dustvoice@DustArch ~  
$ sudo systemctl enable org.cups.cupsd.service  
dustvoice@DustArch ~  
$ 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.

3.15.14. Communication

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

3.15.14.1. Email

| Software Packages | |
|-------------------|-------------|
| extra | thunderbird |

There is nothing better than some classical email.

3.15.14.2. Telegram

| Software Packages | |
|-------------------|------------------|
| community | telegram-desktop |

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

3.15.14.3. TeamSpeak 3

| Software Packages | |
|-------------------|------------|
| community | teamspeak3 |

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

3.15.14.4. Discord

| Software Packages | |
|-------------------|---------|
| community | discord |

You'd rather use discord?

3.15.15. Video software

Just some additional software related to videos.

3.15.15.1. Viewing video

| Software Packages | |
|-------------------|-----|
| extra | vlc |

You might consider using `vlc`

3.15.15.2. Creating video

| Software Packages | |
|-------------------|----------------|
| AUR | obs-studio-git |

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

3.15.15.2.1. Showing keystrokes

| Software Packages | |
|-------------------|-----------|
| AUR | screenkey |

In order to show the viewers what keystrokes you're pressing, you can use something like `screenkeyAUR`



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

3.15.15.3. Editing video

| Software Packages | |
|-------------------|-----------------|
| AUR | davinci-resolve |

In my case, I'm using `davinci-resolve`^{AUR}.

3.15.15.4. Utilizing video

| Software Packages | |
|-------------------|------------|
| AUR | teamviewer |

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

3.15.16. 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 allocate most of the RAM, which is why I added the following line to the file

`/etc/security/limits.conf`

```
@audio - memlock 29360128
```

3.15.16.1. Ardour

| Software Packages | |
|------------------------|---------------------|
| <code>community</code> | <code>ardour</code> |

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

| Software Packages | |
|-------------------|--------|
| 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@DustArch ~  
$ ffmpeg -i in.wav -acodec mp3 out.mp3
```

3.15.16.2. Reaper

| Software Packages | |
|-------------------|------------|
| 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.

3.15.17. Virtualization

| Software Packages | |
|-------------------|---|
| 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@DustArch ~  
$ sudo modprobe vboxdrv
```

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

```
dustvoice@DustArch ~  
$ sudo usermod -a G vboxusers $USER
```

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

```
dustvoice@DustArch ~  
$ sudo usermod -a G disk $USER
```

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

3.15.18. Gaming

| Software Packages | |
|-------------------|---|
| 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.

3.15.19. Wacom

| Software Packages | |
|--------------------|--|
| <code>extra</code> | <code>libwacom xf86-input-wacom</code> |

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.

3.15.20. VNC & RDP

| Software Packages | |
|-------------------|--------------|
| 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.

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@DustArch ~  
$ 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 console**, then double check that the issue really is the not perfectly executed kernel update by issuing

```
root@DustArch ~  
$ uname -a
```

and

```
root@DustArch ~  
$ 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@DustArch ~  
$ 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@DustArch ~  
$ umount /boot
```

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

```
root@DustArch ~  
$ 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@DustArch ~  
$ pacman -U linux-<version>.pkg.tar.xz
```

After that remove the possibly created `/boot` directory

```
root@DustArch ~  
$ rm -rf /boot  
root@DustArch ~  
$ mkdir /boot
```

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

Now simply rerun

```
dustvoice@DustArch ~  
$ sudo pacman -Syu
```

and you should be fine now.



Consider setting up an `fstab` entry for the `boot` partition, in order to avoid such dilemma in the future.

See `fstab` for more.

5. Additional notes

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