Linux Days – Linux in Practice

Date: Fall 2015 - rev. 1

Contents

About this script 3

Basics about Linux 4
- What is Linux? A brief introduction. 4
- Linux Distributions (distros) 4
- Desktop Environments (DEs) 5

Installing a Linux distro 7
- Create an install media 7
- Booting from the installation media 7
- Installing Linux 8

A few cool programs 9
- Video and Audio Software 9
- Optical media 10
- Gaming 10
- Utilities 10
- Internet 11
- Office applications 11

The Unix Console 13
- Overview 13
- Navigating through your folders 14
- File and folder operations in the console 15
- Regular expressions 15
- Quick glance at the sudo command and file permissions 16
- Starting programs 17
- Dealing with processes 17
- Concatenating commands 18
- More useful commands... 18
- Output redirection and piping 18
- Environment (global) variables 19
- Scripting: sh and python 20
- .bashrc: Configuring bash 20
- ssh: Accessing a Linux computer remotely 20
- VPN 21

Users and groups 22

Dealing with files and folders 24
- Theory 24
- The openSUSE file hierarchy 25
- Mounting and Unmounting 26
- Hardlinks, symbolic links 27
- File ownership and privileges 27

The package manager 29
- Theory 29
- Practice: openSUSE 31

Installing software manually 33
- Compiling from source 33
- Self-extracting executable 33

Maintenance 34
- Updating your system 34
- Backup 34
  - using rsync 34
  - using something else 34

Repairing a broken system using chroot 36

Where to go now? 37

About this script

This script is part the two "Linux in Practice" courses of TheAlternative.ch in connection with our LinuxDays. The goal of these courses is to make beginners of Linux understand basic and more advanced features of Linux distributions, in particular openSUSE. Some elements we teach, e.g. the exact syntax of some console commands, can be hard to memorize at first sight. We thought it would be smart to write a script that summarizes the courses in a simple manner.
This script is likely to be updated in the future and will be available in the newest version on our website "`http://linux.thealternative.ch`".

Authors: Dimitri Stanojevic, Sandro Kalbermatter, Maximilian Falkenstein

Basics about Linux

What is Linux? A brief introduction.

Who uses Linux? There are many reasons for people to use it. Linux can be installed on ordinary computers, but it is also used for highly specific tasks. You will find it in TVs, smartphones (Android™ is based on Linux), credit card readers, bus & timetable screens, ticket machines, cars, huge server farms and much more. The Linux community is extraordinarily colorful and diverse, because the majority of it is Free Open Software, allowing anyone to take an existing program and adjust it to their needs. Programs are built on top of each other and based on other programs. You can get the same thing in various variants (often called flavors or forks) and everything can be personalized, exchanged or combined with something else. Linux is what happens when everybody is free to do his/her own thing, all is combined and then you get to choose which way to go... not an easy choice. But do not worry, we will tell you where a good place to start is if you are overwhelmed with the freedom gained by using Linux.

There was a time where Linux was just for geeks and pros. These days have been over for quite a while now. Today, there is a large choice of Linux systems that focus on ease of use and simplicity. Many modern Linux systems are easier and more user-friendly than more mainstream operating systems (OSes) like Windows® and Mac OSX. However, those easy-to-use Linux systems have all the power of Linux under their hood. They will present you a simple user interface (UI) that you can understand without knowing much about computers. The cool thing is that even those systems allow you to see what is behind the scenes, taking advantage of more advanced tasks than what you could imagine right now... if you want to learn about those things. You can use Linux like the OS you have used until now, clicking your way through simple and nice looking dialog boxes. Or you can go on, learn more and move on through this incredible, infinite world for multiple decades (you will never have seen it all). The choice is yours! Do it your way and be happy.

This course will give you a solid base from which you can go on and experiment. It probably contains more info than what you want to know, so we try to explain at the beginning of every chapter what its content are and the respective uses. You can then choose if you want to read on or jump to the next chapter or section. However, we recommend that you have at least a short glance at everything, so that if something comes up to you one day, you will remember that you have seen it somewhere in here and you can come back to read through the corresponding chapter.

First, let us try to give you an idea of what Linux is. Linux is not exactly an OS, but it is a kernel, the central part of an OS. It coordinates running programs and links them to the hardware which actually runs them.

Imagine the Linux kernel like the engine of a car. It is what makes it move, but you will need a frame, wheels and many other components in order to make it useful. Those components can have various types, colors, shapes, looking and working in diverse ways, but still being run by the same engine.

With Linux alone, there is not much you can do. You will need software that goes along with it which enables you to perform your daily tasks on your computer, like a text editing application, a web browser, e-mail client, media player, messaging client, image editor,... all of these are not part of Linux and have to be installed on top of it.

It would be very painful to install and configure the dozens and hundreds of components that have to work together in order to form a useful computer. There are people (often lovingly called geeks) that do this kind of stuff, but you are not forced to. Usually you install an entire bundle of software together with the Linux kernel, all being already pre-configured to work with each other. Such bundles are called "distributions" (in short "distros").

Linux Distributions (distros)

When you get a car, you typically do not just buy an engine, but you acquire all the required components in a pre-assembled package. Distros have the same concept, as they often provide a ready-to-go software bundle, letting you drive right off after the installation. However, distros are more than that: they even pre-configure the available software that you have not installed yet, so that when you install it, it will work with what you already have. This is possible thanks to the concept of Free Open Software: Such software is released under licenses like the GNU General Public License (GPL), allowing anyone to run, view, modify and share them. The Linux kernel itself is Free Open Software, making it possible to acquire it for no cost and to modify & redistribute it. This allows the developers of distros to package existing software together into an ISO which you can simply burn onto a CD or flash onto an USB stick from which you can install it very easily.

A distro always comes with a community. Communities are groups of people from all over the world who communicate mostly via internet and share a common idea of what good software should be like. Each community has its own mentality and usually provides forums and wikis. In the forums, you can ask questions and they are often answered within less than a day by the community. Forums are where developers and users talk and discuss. Sometimes they also use chat application like IRC clients. Wikis provide a database of knowledge documenting everything there

is to know about the distro. This is like the Wikipedia principle, but specific to the distro.
Note: There is no real distinction between developer and user as anyone can modify anything. The developer of program A is a user of the program B whose developer might be user of program A again. Many programs are developed by dozens of people. When someone makes a modified version of an existing program (called a fork), the original developers are free to integrate that work back into their program if they think it is good and the license allows it.
Often, you can directly e-mail the developer of a program. Please do not abuse this for spamming the poor developer who might still be a student developing the program in his spare time. If you encounter problems, use search engines like Google™Search, read wikis or ask your question in a forum.
Of course there are many different ways to feel about computers. As a consequence, there are hundreds or even thousands of distros out there, each with its own mentality, purpose and feel. Some distros focus on beautiful graphical effects. Others are minimalistic and specifically built for running fast on old and weak computers, sometimes not even providing a graphical UI (GUI). There are distros designed for hacking (useful in "security testing"), some were made by governments for governments, some for schools, servers, routers, TVs, Hi-Fi stereos or literally anything you could possibly think of. There is no way you could ever try them all. It is usually a good idea to start with the most popular distros, as they provide large and helpful communities.
In the end the choice is yours! Of course, there are many other distros with those goals, feel free to try any of them.
This script will mainly focus on openSUSE, but most information is valid on all or most distros.

Desktop Environments (DEs)

The Desktop Environment (DE) is what you interact with as soon as you log in into your system. It usually provides a start menu which allows you to access your installed software, shut down your computer, a few toolbars which you use to switch between windows or to see the time, window frames with buttons for maximizing or closing the window and it mostly has a control panel for changing behavior like the keyboard layout, power settings, shortcuts, time zones, design & layout, autostart programs, filetype associations, mouse settings, removable media behavior and much more. The DE is what gives you the feel of your system. It defines your workflow, making it an incredibly important element in your daily computer use.
This is where the Linux world becomes really different from other OSes. In contrast to what you are used to, the DE is not part of the OS, but it consists of components that you can install or exchange by whatever you want. You can even have multiple DEs installed at once, choosing between them every time you log in. But beware: every DE is its own little (or huge) world. When mixing them, you might encounter some compatibility issues.
No need to tell you that if such a crucial part of the system is replaceable, there are tons of flavors of it. Indeed, there are a lot of DEs out there. You could code your own. There are huge DEs with everything, or very tiny ones which give you a taskbar and just display the window.
It would be senseless to try to explain the differences of all these DEs. Best is, you search for screenshots and see which one you like most. Here is a quick overview of the most popular ones in the user-friendly field: For fast, strong machines, there is GNOME, Unity (only on Ubuntu), KDE, MATE and Cinnamon. More lightweight DEs are XFCE and LXDE.
Of course, all these DEs are customizable, some more than others. The most custormizable is probably KDE where you can even set if the checkboxes should be round or squared. You can install different start menus, search engines and more themes. Many of those DEs are compatible with each other, for example you can run tray apps (apps which display information in the panel) written for GNOME also in XFCE, Cinnamon, MATE and Unity. Everything interconnects somehow together and if you are hooked, you can spend years on trying out combinations of applications, configurations and Desktop Environments. If you do not really care about this, just go with the default DE of your distro and you will be fine.
An important part of a DE is the so-called File Manager (FM). This corresponds to Windows Explorer (explorer.exe) in Windows or Finder in Mac OS. Most DEs (especially all of the above) come with their own file manager, called Thunar (XFCE), Nemo (Cinnamon), Dolphin (KDE), PCManFM (LXDE), Nautilus (GNOME) etc. If you like your DE but not its FM, you can just install another one. You can have as many file managers as you want and use them all in parallel. Often, you can even drag files from one FM and drop them in another one.
Note: You can theoretically combine everything with anything. Exotic mixes might result in compatibility issues, but if you are determined to make it, you could fork a project and make your own. Things can get very complicated though as everything is built on top of something else. For example, it can be a bad idea to mix programs written for and by the GNOME-community (including other GTK-based DEs like MATE, Cinnamon, Unity and XFCE) with Qt-based programs like the ones by the KDE community. As a concrete example, consider PDF viewers: if you install KDE's PDF viewer "Okular" on your XFCE system, you will find yourself with a very large download including half a KDE system that is required to be installed before you can run Okular. Also, the first start of Okular after every boot will be slow, because

many KDE components need to be loaded. As XFCE is GTK-based, you should rather install the PDF viewer "Evince" from the GNOME community, as it will run on stuff you already have installed. Of course, all this mixing is only for people who like to experiment.
If you just want a working system, you do not have to worry about these compatibility issues. Pick the distro which best fits your mood and upon installation, if prompted, choose the DE which you like the most as many distros give you the choice of which DE to install before you install the OS
In case you can't decide - on openSUSE you can have multiple desktop environments installed at the same time and switch between them by logout an login.
Note: Ubuntu goes a different way. There is a sub-distribution for each DE: Kubuntu comes with KDE, Xubuntu with XFCE, Lubuntu with LXDE, Ubuntu Gnome with GNOME etc.

Installing a Linux distro

This chapter is about installing Linux onto a computer. If you already have a working installation, feel free to jump over it, unless you want to get an impression of how it would be done.
Installing Linux on your computer can be anything from extremely straightforward to impossible. Some manufacturers make it as hard as possible to run anything else but their preferred OS on their machines, others nicely follow international standards so that you will not encounter any problems at all. It becomes clear, the nature of the installation process strongly depends on the type and model of your computer.
During our Linux Days, we have two Install Events where we sit down with you and help you install your favorite distro. If you feel afraid, come on in and let us take care of you and your machine. Check out our website for the dates and rooms: `http://linux.thealternative.ch`
In the following, we try to give you an idea of how the installation could work if you want to try it by yourself. Once you have your Linux installed, things will get much easier, since we are all taking about the same thing then. At this step though, you will maybe need some imagination to figure out what the described step looks like on your computer.
First, **CREATE A BACKUP**. You will be walking on a really thin bridge, holding your hard drive in your hands. If you have no idea what you are doing, you can easily lose your data. Also, a hardware or software problem causing loss of data could occur at any moment. We require you to make a backup even before coming to our Install Event and we are not responsible if you lose any data — that is your risk and responsibility. Note that your files are never safe if you just keep them in one place. A hard disk drive (HDD) or solid state disk (SSD) can always fail at any moment, no matter how expensive it was, how it was treated and/or how old it is. This is why you should always have everything that is important to you on two places simultaneously. Typically, backups are simply a copy of your data on an external storage medium. They should be done every two weeks and every time before you do something critical (like installing an OS).
The process of installing usually takes three steps:

Create an install media

Go to the distribution's website (e.g. `download.opensuse.org`) and download the .iso file. Unless you have a very old computer (e.g. Intel Core Duo® processor), go ahead and download the 64-bit version.
You can then either burn the ISO image to a CD/DVD or extract it onto a USB drive. For the latter, check out the information provided on the distro's website.
If you are using Windows, check out Lili USB Creator which will do this all automatically for you: `www.linuxliveusb.com`.

Booting from the installation media

Now you need to tell your computer to start (boot) from your freshly created installation media instead of the internal drive. Depending on your machine, this can get. . . interesting. Be patient and do not forget that you can always come to our Install Events which take place every semester.
On a Mac, things are weird and depend on what kind of Mac you have. In most cases, you will have to shrink you OSX partition from your running OSX installation, reboot into the openSUSE installer and take special care with the bootloader.
On a PC, there are many, many possibilities, depending on your manufacturer and model. Ideally, you plug in your installation media and reboot and it boots directly into your distro. Unfortunately, this is getting less and less likely.
Note: If the following paragraph does not work for you and you are using Windows 8 or later, keep reading.
When booting, your computer traditionally briefly shows its manufacturer's logo. At that moment, you have to press the magic key which depends on your machine. On HP laptops, this is typically the Escape key. On Lenovo notebooks, it is likely to be the Return key. On Sony, good luck, as it seems to be different for every model and there is a lot of disinformation about it on the web. If Windows is being started, reboot and try again. Be patient. Drink some tea or coffee. Once you have succeeded to get the boot menu, look for something like "Choose a temporary startup device". There, you select your USB or CD drive using the keyboard, then hit Enter. Hopefully your distro's install screen is now being displayed.
You are using Windows 8 or 8.1? Well, the above might not have worked for you. Boot up your Windows, then go to the Metro UI and select Reboot while pressing and holding Shift on your keyboard. You should land in a blue menu, select Troubleshoot. Select Advanced options and go to the UEFI Firmware Configuration. There, look for a feature called "Secure Boot" and make sure it is turned off. Then, look for an option called "Boot Order" and put the internal drive below CD and USB in order to make the computer look for external bootable devices before it starts your internal OS. All this strongly depends on your machine. Then save and reboot. Hopefully, your distro will be booted.
Desperate? Search online if someone has documented Linux booting on your machine. And always remember: feel free to come to our Install Events.

Installing Linux

Congratulations, you got your machine to boot! For most distros, you now find yourself in a welcome screen(OpenSUSE instead boots straight into the installer). After setting language and keyboard layout and asking whether to use online packet sources during the installation(only do so if you are sure that your connection won't fail during the installation), you will be asked whether you want to delete your disk and install Linux only, or if Linux should be installed next to your existing OS. Make sure the installer has recognized your existing installation, otherwise it might overwrite it, deleting all your data! This is the reason why backups are so important.

If the installer does not provide you the option to install your distro next to the existing OS and you want to keep the latter, you have to chose the advanced option. You will now have to deal with partitions and file systems. These are described in the chapter "Dealing with files and folders", under "Theory". We recommend that you read the chapter before commencing.

Advanced Option: On earlier machines, you can only have 4 primary partitions. If there are already four of them, you need to delete one before moving on. Usually, it is best to delete a small service partition (HP calls theirs "HP_TOOLS"). Then shrink your Windows partition (always shrink it from the back since if you move the front, Windows will fail to boot and you need to fix it with the recovery CD which will then destroy the Linux boot which will need to be repaired itself too). Behind your shrunk Windows partition, create a new extended (logical) partition that will take all the space except for the size of your RAM. It is important to hav at least that amount of space free to allow for hiberation on laptops. Format it as ext4 and select '/' as mount point. Then, create another logical partition taking up the remaining space and format it as swap.

Advanced Option: On more recent machines, you can usually have as many primary partitions as you want and it is not possible to create logical partitions. Just shrink Windows and create the two partitions as described above, but the partitions are primary in this case.

If you did enable the online repositories before it will now ask you which ones specifically you'd like to have. For most users, OSS, Non-OSS, Updates and Non-Open-Source Updates are probably the best choice.

Afterwards you will have to set up an user account. Linux also generally has a special account called root which will also need a password.

The next screen is about local time: Select the correct time zone(you can click on the map, too). If you're *not* dual-boooting Windows you can check the UTC checkbox. If you check it when you're dual booting, your clock will be off by your timezone offset everytime you switch between Windows and linux.

Finally, the installer will display a short summary and also show you where it thinks to install the bootloader, typically GRUB or rEFIt. A bootloader is a program that is booted instead of an OS and which displays a menu that lets you choose which OS to boot if you have several OSes installed. The bootloader is managed by Linux and should to be configured there.

Upon clicking on install, the process will start. Resizing partitions takes between one second and several hours. It is the critical part for the data, make sure your machine is plugged in and securely placed. When it displays a slideshow, relax and wait for the installation to complete. You can also watch the packages being installed.

When done, you might be asked to reboot. Go ahead and enjoy your new system. Try out stuff and experiment. We will show you some cool software in the next chapter. If either system fails to boot, do not worry – your data is probably still there but the little link needed for starting it is broken. If Windows fails to boot, use the Windows recovery DVD to restore it. This will override the bootloader and therefore your Linux will no longer boot. If Linux fails to boot, check the chapter "Repairing a broken system" below or look at your distro's wiki to know how to fix it.

A few cool programs

In this chapter, we briefly highlight a few cool applications that you might like. Some of them are installed by default on any openSUSE system, others have to be installed by you. The chapter on the package manager will help you with that.

Video and Audio Software

Although you may have a hard time trying to run iTunes on Linux, there are many different music players for Linux that are just as good. Most Distros have preinstalled applications for playing music. Of course you are free to try out any of the others. Some well known alternatives are Rhythmbox, Clementine, Gmusicbrower, Amarok or Banshee.

Amarok
Amarok comes as default in openSUSE if you choose KDE. It can manage your music collection on your computer and on any attached music player/i-somthing, display the lyrics of the song currently played and crossfade between songs, to name a few things. There are lots of plugins to provide advanced functionality, so check them out if you are missing something.

Rhythmbox
Rhythmbox comes as default in openSUSE if you choose GNOME. With Rhythmbox you can manage and play back your music collection, listen to podcasts, to list a few things it can do. It also supports iDevices.
Rhythmbox also has a plugin system but there are not many plugins.

Clementine
Clementine is a music player(forked from Amarok) that comes with many more features compared to Rhythmbox. For example Clementine will automatically search the internet for information about the artist you play and will show you the biography, similar artists, etc. Clementine is highly configurable and allows fading music in and out between the songs or when you pause/resume them, which is nice for smooth playback.

Picard
Picard is a software that scans your music library and automatically tags songs, meaning that it will find the name of your song, the artist, the album and much more. Picard can write this information into your mp3 file and even move/rename your music file to create a clean and tidy library automatically. It finds out what the songs are by calculating audio fingerprints of your music files and comparing them to a huge database at `musicbrainz.org`.
Picard is very useful for maintaining larger digital music collections.

EasyMP3Gain
If the varying loudness level of your songs is constantly forcing you to adjust the volume, you may try EasyMP3Gain. It will readjust the gain of your songs by making the more quiet songs louder. This is a fully reversible process, as the gain adjustment is only saved as meta-data and does not affect the audio data itself.

Audacity
Audacity is an audio wave editor in the style of Steinberg's Wave Lab or Adobe Audition. It is not as advanced and has less features than these professional tools, but it is still quite useful, providing a considerable amount of filters and effects. Audacity can be used for audio recording, mixing and editing. It can do anything from recording your voice to mixing music you recorded from different instruments.

VLC
You may have already used VLC on Windows or Mac. It is THE player for pretty much any kind of digital video or audio. If there is any video or audio codec that VLC does not play, probably no other player will do.

Openshot
Unfortunately, there are no Free video editors as powerful as the Adobe Suite or Final Cut Pro yet. There are powerful tools which are hard to use or simplistic ones that offer a limited amount of functions, but a great suite for everything is yet missing in the Open Source world.
We choose to present Openshot because it is quite stable and simplistic. Openshot is like a better version of the Windows Movie Maker. It has a very clean user interface and gets its job done efficiently.
There is also other video editing software like Cinelerra, Kdenlive or Avidemux with more advanced features but they usually tend to be more complicated to use.

Blender
Blender is considered THE open source software for creating 3D objects and animations. It is constantly being enhanced by adding additional features and improving performance and stability.
The Blender community published many stunning movies showcasing the abilities of the software. It can also be used for designing 3D objects for printing.

Optical media

K3b

K3b is a CD/DVD/BluRay burning software that gets installed on openSUSE when choosing KDE. It can burn and transcode(although you would better use Handbrake for transcoding) audio and video CDs/DVDs. It's also possible to burn data CDs/DVDs/BluRays.

Brasero

Brasero is a CD and DVD burning software that is comes preinstalled in openSUSE when using GNOME. It can burn audio, video, or data CD/DVDs. Also, it is able to create video DVDs/audio CDs that will play on your DVD/CD player.
Alternatives to Brasero are Xfburn and k3b, while Xfburn is more lightweight simplistic and k3b has more configuration options.

Asunder

This very simple and straightforward program transforms your audio CDs into a set of mp3 files which you can play on your computer, phone or mp3-player. The process is called audio ripping. This is legal in Switzerland, as long as you do not share these files with anyone.

Handbrake

Handbrake is one of the most powerful and reliable pieces of Free Software to rip DVDs and convert video files. With the right libraries installed, it breaks the encryption of DVDs and transforms them into simple mp4 files. Instead of a DVD, you can also feed it a video file in a different format, which it will convert to mp4. Again, ripping is legal in Switzerland, as long as the files are not shared.
Note that you probably need to install Handbrake from PPA (refer to the chapter about the package manager).

Gaming

Steam

If you have used Steam before, you may be happy to hear that the Steam client is available for Linux. It can even be installed through community repositories. However you will only be able to install games that are actually written for Linux. Unfortunately you will also probably require proprietary drivers for your graphics cards for games to run decently.
If you want to run Steam games written for Windows, you can install the Steam client for Windows using Wine (see section about wine below). There are many people who tried this already and you can always find information online about which games work and which do not work.
At this point we must add, that neither Steam nor many of the games you can buy on Steam are open source.

Utilities

Virtualbox

Virtualbox is a software for creating and running virtual machines on top of your Linux distribution. This means that parts of your actual hardware will be used to run another operating on top of the one you are currently running. You can test or use other Linux distributions, Windows, or even Mac OSX(theoretically only on Mac hosts) on top of your Linux Desktop. To start a virtual machine, boot up your computer, then start Virtualbox and tell it to start the machine. A window will open, containing the screen of your virtual computer. OSes in virtual machines will always run slower than on your real computer. However, virtual machines have several great advantages: They work in a completely protected environment which allows you to try out dangerous things without putting your real OS or hardware at risk. For example, what your virtual OS thinks is a hard disk is in reality just a file. Virtualbox allows snapshots, meaning that you can store the current state of the disk and come back to it at any time (just like a gamesave). Also, you can play around with the hardware you are emulating, like pretending that your processor only has one core or adding virtual network connections or as many virtual CD drives as you want. Virtualbox relies on so-called kernel modules, which are pieces of software that extend the operating systems functionality. On openSUSE, all this is set for you when you install Virtualbox. If you receive any errors when trying to run your virtual machine, try loading the following modules with:

```
sudo modprobe vboxdrv vboxnetadp vboxnetflt vboxpci
```

Wine: Running Windows programs on Linux

Wine is software that lets you run programs written and compiled for Windows under a Linux system. Typically, just install the wine package and run the .exe files. Note that you must have the execute privilege on the .exe file. Right-click it and under Permissions, check "run file as a program".
Many programs and even games can be brought to life on Linux that way, especially if they are simple or have a certain age. Wine is huge and very extensible. To mess around with settings, squeeze out every possible functionality, is to install the "`winetricks`" package. It lets you install tons of Windows libraries (like the .NET framework). If you are unable to run a program at first, there are many options you can play with to try to make it run anyway. Some programs like Microsoft Office run partly, but not very smooth. Other software like DirectoryCompare runs perfectly

fine right from the beginning.
The Wine community is very large and seems to consist of people who spend their days playing around with settings trying to get their favorite program to work. Check out their nice website which offers a considerable knowledge database about this topic: `www.winehq.org`.
There is an interesting project based on wine that focuses on gaming on Linux, called PlayOnLinux. They have a package that you can easily install on most distributions. PlayOnLinux comes with lots of tweaks and tricks for each game so that you do not have to figure these out on your own. Their website is `www.playonlinux.com`.

copyq
A clipboard manager saves a history of all your clipboards allowing you to work with several clipboards. I once happened to lose a password because I copied it to a clipboard and then lost it by copying something else to the clipboard. This would not have happened if I were using copyq at the time. KDE users might prefer Klipper, which comes with KDE.

guake
Though every Linux distro has its own preferred terminal emulator preinstalled, it is very simple to install and try out different ones. Guake is a terminal emulator that works similar to consoles you may encounter in certain games. By default guake appears on the top of your screen by pressing the F12 button and disappears again if you click on anything else on your screen. This way guake can be used very efficiently for smaller console tasks without disturbing your workflow. There are many other terminal emulators you should try. The maybe most feature packed terminal is called terminator. On first glance it resembles the default gnome-terminal but it has several additional features like splitting the window to open several terminals besides each other. The equivalent from the KDE Project is called "yakuake".

Internet

Skype
Skype is available on Linux and can be installed from the Skype web site, however it is closed source software. It is being uploaded by Microsoft, not by SUSE.
There is currently an open source alternative being developed called Tox, but it is in early stages of development and not ready for productive use yet.

Dropbox
Dropbox is a cloud storage service that you may already be familiar with. It is good for backing up and synchronizing files over multiple devices or sharing data with friends or workmates. Dropbox is available for Linux and can usually be installed from the main repositories. By default it installs a folder called 'Dropbox' and will synchronize all data inside that folder. You can however synchronize any folder on your computer by simply creating a symbolic link (see chapter 5) and pasting it in that 'Dropbox' folder.

Chromium
Chromium is an open source browser developed by the Chromium Project. Google Chrome on the other hand is based on Chromium but contains additional code, some of which is not open source. Some of this additional code contains the Adobe Flash Player, the option to send usage statistics to Google, and an auto updater. Since version 11 Adobe has stopped developing Adobe Flash Player for Linux. However many websites have already started using HTML5 and we hope the trend will continue.

Office applications

TeXstudio
TeXstudio is a fork of the LaTeX editor Texmaker, expanding it by some additional features. In case you do not know, LaTeX is a markup language and system for writing documents. In contrast to WYSIWYG ("what you see is what you get") editors like Libre Office Writer or Microsoft Word, the approach in LaTeX is to describe the text using properties similar to a html-page and then compile it to a pdf. LaTeX is especially useful for writing mathematical formulas.
As a matter of fact this document has been written in LaTeX with TeXstudio to a large extend.

Libre Office
Libre Office is an independent derivative of the Open Office project. It contains many of the software that can be found in the Microsoft Office suite. Libre Office consists of:
- Writer: word processor
- Calc: spreadsheet editor like Excel
- Impress: presentation program similar to PowerPoint
- Draw: vector graphics application
- Math.: for creating mathematical formulae
- Base: database management

To put it briefly Libre Office is a pretty good replacement for Microsoft Office when doing day-to-day office work. However it lacks many of the newer features that Microsoft office offers.
If you do not need a full office suite and just want to have a simple and fast document writer and spreadsheet application, you can instead try Abiword and Gnumerik.

PDF viewers
Evince is the default PDF viewer on GNOME based Desktops. It is a very simple but fast PDF viewer without any special features.
Okular on the other hand is a much heavier kde based feature rich viewer that allows you to draw onto documents, change the background color for better readability or even tweak some performance options.
You can however also install the official PDF reader from Adobe called acroread. Acroread is not open source but should definitely be most standard compliant pdf viewer as it is created by the designers of the format.

Simple Scan
Simple Scan is the default scanning program on Ubuntu and it works quite simple. It allow you to scan Documents using two different modes Images and Documents. In the preferences you can also adjust the level of detail of the scan.

pdfshuffler
Pdfshuffler is a simple program for manipulating already generated PDF files. It can merge or split PDF documents, rotate certain pages or change the order of pages. However, Pdfshuffler cannot change the contents of PDFs.

GIMP
GIMP is a program for image editing based on raster graphics. This means that it treats images as a table of pixels, small points that each have a designated color. GIMP can read and write to most popular image formats like JPEG, PNG, GIF or bitmap. It comes with an own format for saving layered images and also supports (to a certain degree) the Photoshop format.
While the popular image editor Photoshop is usually being developed by a team about a hundred paid full time employees, there are only a handful of volunteers behind GIMP. For that reason GIMP lacks some of the advanced features that Photoshop has. It is a very useful and feature rich application for all kinds graphics editing tasks though.

Darktable
Darktable works similar to Adobe Lightroom. You can import a set of images, modify them in an environment where every step is reversible and then export them. While Gimp offers much more functionality for detailed image editing, Darktable is much better for optimizing larger collections of photographs and giving your pictures the "last touch".

Inkscape
While Gimp deals with pixel graphics, Inkscape is designed for dealing with vector graphics. If you are familiar with Adobe products, you can think of Gimp and Inkscape as equivalents to Photoshop and Illustrator.

Scribus
Scribus is a desktop publishing software similar to Adobe Indesign. It can be used to design journals, magazines and books.

Ocrgui
Imagine you have a larger pile of text documents that you want to scan and then turn into a text based PDF (not images). Ocrgui is the user interface to gocr and tesseract which are both so called optical character recognition (OCR) software. You can specify the image of some text document and ocrgui will try to recognize the text segments and generate a text output. Ocrgui can also use hunspell, a spell checking software, for better results. In case you have problems with Ocrgui, there is a similar program called Gimagereader that you can try instead.

The Unix Console

So far, you have mainly used the mouse to interact with your computer. You lost a lot of time searching for buttons and toolbars, then aiming and finally clicking. More advanced users still swear on the console. Especially on Windows, the console has become forgotten. This is very sad, since even today it is the fastest possible way to interact with a computer. At first sight, consoles look ugly and frightening, but you will get to love its minimalist, simplistic beauty in the next few chapters.
Note: The console cannot replace a graphical environment. Consoles cannot display images or videos, so you are bound to switch between console and GUI all the time if you want to get maximum productivity. However, this is not a problem. Linux-based systems have adapted to this for years and offer various tricks that make the coexistance of the graphical and the text-only interface natural. You will be surprised how fast you will get used to the console!

Overview

tty vs terminal

There are mainly two ways to access a console in Linux. The first one is to completely leave the DE and use a so-called tty. There are usually 6 ttys on your system which you can access with [Ctrl]+[Alt]+[Fi], $i \in \{1, ..., 6\}$. Use [Ctrl]+[Alt]+[F7] to get back to the GUI. Note that you have to log into each of the ttys you want to use. You can be logged in several times at once on the same machine, that is no problem.
The second way to get to a console is to open up a terminal within the DE. A terminal is a software application that allows you run a console within a window. Also, it provides functions like text selection, searching, tabbing, scrolling in history and much more. Terminals are regular programs; you can install as many of them as you want and use them in parallel.
Note: In a console, [Ctrl]+[C] stands for 'abort'. If you want to copy-paste, you typically use [Ctrl]+[Shift]+[C] and [Ctrl]+[Shift]+[V]. Note that when you select a word in a terminal, it automatically gets copied into another clipboard. It can then be pasted by middle-click (pressing the scroll wheel on you mouse).

The shell

When you are in a console, you are running a program that executes your commands, a so-called shell. You can run any shell in any terminal, so combine whatever fits your personality. The default shell on openSUSE is called Bash. Do NOT remove bash as many programs assume it exists on your system.

An alternative shell would be the "friendly interactive shell", fish. fish is more intuitive than bash; if you want to try it, simply install it using the Software Center or the package manager (see chapter about package manager). If you are in bash and you want to switch to fish, simply type "fish" (without the quotes). You can type "bash" when you are in fish to spawn a bash-shell. Also, you can go to your terminal's settings and change the default shell.
Some uses also prefer the z-Shell (zsh for short) which can be highly custormized.
To exit a shell, type "exit" or press [Ctrl]+[D]. For example, if you started a terminal with bash and you started fish from there, [Ctrl]+[D] will get you back to bash. When you exit out of the last shell, your session will be closed. On a tty, that means you get back to the login screen; on a terminal, it will close the window or tab.
When running a terminal, it is good practice to never exit it using [Alt]+[F4] or the close-button, but instead use [Ctrl]+[D] to close it. This way, you avoid killing a process you have forgotten that is still running in the terminal.

bash basics

If you want to learn how to use fish, simply type "help" and it will open up a browser with a very good documentation. In this script, we will not discuss fish any further and instead stick to the more traditional bash.
When you turn on your terminal you will usually see a line looking like this:

```
peter@linux-machine: ~ $
```

These are the name of the user, followed by the name of the computer and by the path of the current directory. `~` stands for your home directory which is usually `/home/user/` where "`user`" is the username of your account on that computer.
There are two types of privileges: If you are using the console as a regular user, the line will end with a '$' whereas if you are logged in as root (the 'administrator'), it will show a '#'.
In the following section we discuss the most used console commands. In openSUSE most of these commands have a manual page (called **manpage**). The manpage is a summary of the program and explains how it is used. It can easily be called by typing **man 'program'**, where 'program' is the name of the program you are interested in. You can even type "man firefox" to see about the various parameters you could start firefox with. You can leave the manpage with 'q'.
Alternatively you can call some programs with the **--help**/**-h** parameter to print the most important info directly into the console. So whenever you are not sure about what some command does or want to know about the parameters you can look it up in the manpage.
Caution: Remember that everything you type into the console is case-sensitive, which means lowercase and uppercase letters are treated differently, i.e. "Hello" is

not the same as "hello". The same rule applies to files and folders.

Structure of commands
When you type a command in the console, it is usually of the form **command -abc --option1 --option2='This is an option' argument1 argument2**. **command** is what there is to execute. Then come the options. They are either single-letter options like **-a**. Such options start with a single dash ('-') symbol. Writing **-abc** is equivalent to **-a -b -c**. For passing options that are longer than a single letter, you need two dashes ('--'). They can specify values which are typically written in quotes and behind '='. Finally, you pass the arguments.
The command tells the system what to do (what program to run for doing it). The options specify how to do it and the arguments define where to do it.
Example: **nano -mB --backupdir=mybackups/ file.txt**: The command is **nano**, a console text editor. There are three options: **-m**, which makes nano listen to the mouse (meaning the cursor will jump to the position you clicked at, this works even in a console!), **-B**, which tells nano to backup the old file before overwriting it, and **--backupdir=mybackups/** which means that the backup file shall be placed in the directory `mybackups/`. Finally, one argument is given: `file.txt` which is the file you want to edit.
Note that this structure is only a convention. There are commands that use a different structure. Everything you type behind the command is simply handed over to the command. To know what kind of options/arguments the command expects, check out its manpage: **man *yourCommand***

Navigating through your folders

A path to a file or folder always starts with / (also called the root). It consists of the name of all parent folders separated with / signs and ends with the name of the specified file or folder. Note that Windows uses the \ (backslash) instead of the / (slash). An example of a correct UNIX path would be `/home/peter/Pictures/wallpaper/beautiful-ocean-view.jpg`.
Because on linux the path `/home/user` (the so called home-directory) is abbreviated as `~` the above example is equivalent to `~/Pictures/wallpaper/beautiful-ocean-view.jpg`. There is more about this topic in the chapter "Dealing with files and folders", section "The openSUSE file hierarchy".
The navigation inside the console is very simple and only requires the following two commands:

ls
ls shows you all files and folders in the current directory. You can type a location after ls to see what is in that location. So if you type "ls Pictures/" in your home folder (written as ~) and press Enter, ls will display the contents of the Pictures folder in your terminal.
ls -a will additionally show all hidden files. Hidden files are files that start with a `.`. Here **-a** is a parameter for ls.
ls -l displays additional info like file size and privileges (more info about file privileges is to come in the chapter "Dealing with files and folders").
You can combine such parameters by simply concatenating them: **ls -lah** is equivalent to **ls -l -a -h'** and displays all files (including hidden ones) with their properties. The **-h** option transforms unreadable file sizes like 467915 into human-readable (this is why it is -h) numbers like 457K where K stands for Kilobytes.
Note: Options like **-h** or often have similar behavior on other commands as well. For example, the command **rsync** (which we will see in the chapter about maintenance) will also display human-readable numbers when called with **-h**. However, this is not a convention and **-h** can stand for anything, so be sure to check the man pages before blindly typing a command.
ls has many other useful parameters and you can always look them up with **ls --help** or **man ls**.
Hint: There is no option to hide a file in Linux. Instead, all files and folders starting with a `.` are hidden, for example `.mozilla/` or `.my-awesome-document.odt`. To unhide them, simply remove the dot from the file name.

cd
cd is used to change the directory. If you type **cd Pictures/** the current directory will change to `~ /Pictures` and the terminal will therefore display `peter@linux-machine: ~ /Pictures $`. Of course if you now type **ls**, it will again show the contents of `~ /Pictures`.
cd .. is used to change to the parent directory. **cd ../..** will go up two directories.
Note: `..` can always be used to address the parent directory when typing an address. For example `Pictures/../Pictures/../Pictures/` is a valid address and equivalent to `Pictures/`. Similarly, `.` refers to the current directory. `./././..././` is therefore equivalent to `../`. This can be useful when you want to run a program that is in the current folder.
The Linux Terminal offers auto completion. If you type **cd Pi** and then press tab, the terminal will complete **Pi** to **Pictures**. If there is more than one folder starting with **Pi**, the terminal will not auto complete. If you then press tab again, the terminal will show all valid possibilities. So with tab the console will auto complete as many

characters as possible as long as they are unique. Note that case-sensitivity is very important here: 'blah' is not the same as 'Blah' or 'bLah'.
If your file path contains spaces , you can either put it into quotes (`'/home/peter/my folder'`) or put a backslash before the space (`/home/peter/my\ folder`).
Finally if you just type **cd**, the current directory will change back to your home folder, independently of the current path.

File and folder operations in the console

cp source destination copies the file **source** and saves it under **destination**. If you need to copy whole directories, you need to add the **-r** parameter which stands for recursive. So **cp -r folder1 folder2** will copy **folder1** with all its contents and name it **folder2**.
mv source destination moves the file **source** to **destination** without copying it. If **source** and **destination** are located in the same directory, then the effect of mv is the same as renaming the file from **source** to **destination**. Therefore mv is often used for renaming files or folders. In contrary to cp, mv is capable of moving around folders without the need of the **-r** parameter. If you however move a folder to an already existing folder, i.e **mv folder1 folder2** then the first source folder will be moved into the destination folder. If you want to specify the current path, for example you want to move a file to the current path of the terminal, you can do so with `./`. For example **mv Pictures/wallpaper ./** will move the wallpaper folder next to the Pictures folder.
It is also possible to address several files using the ***** sign. The ***** is called a wildcard character and substitutes any sequence of characters. **mv folder1/* folder2** will move all files and folders located in **folder1** to **folder2**. Similarly **mv folder1/a* folder2** will move all files starting with the letter **a**. It is also possible to call **mv *.jpg target/** to move all JPEG-pictures in the current folder into subfolder target. The ***** wildcard can be used in any terminal commands like **ls**, **cp**, **rm**, etc. for selecting files. It can even be used with the package manager when selecting software that is to be installed (more on that later).
Caution: cp will only copy directories (folders) if you call it with **-r**. **mv** however moves folders even without the **-r** option!
mkdir 'name' is used to create a new empty directory.
rm 'name' removes (deletes) the file from the hard disk.
rmdir 'name' is used to remove an empty directory. If the directory contains files, you first have to remove these. If you want to remove a directory with its entire contents, use **rm -r**.
Caution: If you remove a file with the **rm** command, it will **not** be moved to the trash bin. Instead it will be erased from the drive. It is gone forever! Be careful with **rm**!
Again: Be advised that most of these commands are **not reversible**. That means that there is no undo function as you may know it from various file browsers. If for example you do **mv hi.txt hello.txt** while there already exists a file called `hello.txt`, the original `hello.txt` will be overwritten without any warnings!
If you should ever need to edit a file using only your console, you can do so by typing **nano 'filename'**. Nano is a very simple text editor that exists in most Linux distros. Look at the bottom of nano's interface: you will see a footer text which contains fields like '^X Quit'. The ^ stands for the [Ctrl] key, so exit Nano using [Ctrl]+[X]. There are other very cool console editors with more complex features. Two well known ones are **vim** and **emacs**. If you are interested in doing more things in the console, for example programming, it is definitely a good idea to take a look at one of them. The main benefit of these editors is that they do not require a mouse or GUI. Everything you would usually do with a mouse, like selecting text or scrolling, ca be done with the keyboard.

Regular expressions

Regular expressions, also called regex, allow you to create more complex search patterns. They are often being used in scripts when you need to search and replace certain sequences of characters in a text file. Also applications like Libre Office Writer or Eclipse support regular expressions in their search module. In contrast to wildcard matches, which also share a similar syntax, regular expressions have more so called metacharacters and thus give you advanced control when writing search patterns. Metacharacters are special characters, that are used to define a search pattern, but are not part of the pattern that is being searched for. The most important metacharacters are { } [ ] ( ) ^$. | * + ? and \. We will now go through the basics of regex and provide you with some simple examples:
The dot '.' metacharacter is used to match any character. The search pattern ".ubu.tu" will match the words Xubuntu, Kubuntu, Lubuntu, but also Gubultu or Fubugtu.
To restrict the search further we can replace the dot with possible characters written in square brackets: [XKL]ubuntu will now only match Xubuntu, Kubuntu and Lubuntu. Instead of listing the possible letters you could also write [a-z] for a,b,...,z, [a-dx-z] for a,b,c,d,x,y,z or any such combinations, while '-' specifies the range between the two surrounding characters. The '^' character is used to negate the above expressions. [^abc] will match all characters that are neither a,b nor c.
The '^' character is also used to match the start of a line. "^For" will only match words containing 'For' are at the beginning of a new line. '$' on the other hand will match the end of a line.

Differently than in wildcards the star '*' metacharacter is used for specifying that a certain pattern may appear multiple times. The expression "Yipee* " will match Yippe, Yipee, Yipeee or Yipeeeeeeee, while, "Yip(ee)* " will only match Yipee's with an even amount of e's. So the star only affects the preceding character or group of characters written in brackets. (Note that there is a space at the end of the expression as we only want to search for complete words.)
Curly brackets can be seen as an extension of the star character. {m,n} will match sequences where the preceding character appears at least m and at most n times in a row. "Yipe{2,4} " will match only Yipee's with two, three or four e's.
Regex also defines the question mark '?' and the plus sign '+' as specializations of the star character. '?' will match if the character or sequence appears once or not all and '+' will match if it appears at least once or multiple times in a row. So as an example "bark?" will match the words bar and bark, whereas "bark+" will match bark, barkk, barkkk and so on. You can see, that the star meta character is like the union of the the plus and the question mark character.
There is also a way to express a logical-or between different sequences with the pipe character '|'. "hi|hello|good morning" will match either hi, hello or good morning.
Regex includes a few more meta characters but these basic ones should suffice in the beginning. You can combine all of them to create more complex expressions. Let us end with a few examples with the pattern matching command grep:
Let us assume we have a file called passwords that as the name suggests contains some passwords. Let us also assume that these passwords are scrambled across the file but that each password starts with a new line. Our goal is to print out all passwords that consist of between eight and ten alphabetic letters, followed by any sequence of the numbers 4,7 or 9 and end with a 0.
First we think of the regex expression we will use: The first part can be expressed as [a-z]{8,10}, the second we will write as [479]* and for the ending zero we will simply write 0. Now because we only want to match sequences that fill out a whole line, we will surround the expression with ^and $. So our expression will be the following:

```
^[a-z]{8,10}[479]*0$
```

To wrap this into the grep command, we write:

```
grep '^[a-z]{8,10}[479]*0$' passwords -E
```

Here the quotation marks are needed to indicate the start end end of the regular expression, passwords is the name of the textfile to be searched and -E is a flag used to indicate that we want to use the extended regex notation.

Quick glance at the sudo command and file permissions

When you log into your openSUSE account, you have all rights to read and change the files in your home folder. However if you try to change anything located at a different location, for example you try to copy a new program into the `/usr/local/bin` folder, the terminal will harshly respond with the error-message 'Permission denied'. This is an important security feature: as a normal user, you do not have the right to write to that destination, only root can do that. Root is the super-user on a Linux machine. Root is God, root can do anything. So, in order to write to `/usr/local/bin`, you have to login as root.
The simplest way to solve this problem is the use of the **sudo** command. The **sudo** command can be used before any other command and signals that you want to run the specified command with root privileges.
Be aware that **sudo** really gives you power over the whole system. It is even possible to mess up the most critical system functions. Running applications with the **sudo** command without any real reason is very dangerous because it will grant these applications the permission to mess with all your system files. Suppose you run Firefox with the **sudo** command (`sudo firefox`) and for some reason you install some dangerous plugin which succeeds to override Firefox's defenses and tries to harm your system. That plugin will then have all the rights to deal with your system. On the other hand if you do not run Firefox with the **sudo** command (which usually happens if you run it) in the above scenario the malicious plugin will have a very hard time reaching your system files. This is also one of the reasons why Linux is very well protected against viruses and worms. It will not protect your own files from any harm though... At this point it is important to note that Linux systems are not immune to viruses and other harmful software. However there are much less viruses designed for Linux than Windows and Mac because of the much smaller user base. It is also significantly harder to get one due to the way Linux is designed (tested repositories, the need of the **sudo** command, etc.). You will find out in the chapter about the package manager that you hardly ever have to run setups; instead, you stay in a protected environment when you install normal Linux software.
When you execute **sudo**, you will be asked for root's password only the first time. You can then keep using it without password in the same console, for your convenience. However, there is a timeout: if you do not run the **sudo** command for a while, it will ask you for password again.
You can permanently become root by typing **sudo -i** (which has a timeout) or **su** (no timeout). The '$' sign at the end of the command prompt then becomes a '#', indicating that you are now root. You can leave the superuser-mode with [Ctrl]+[D]

(the line must be empty) or by typing `exit`. su also performs a full login process for root. The most essential difference is that su allows you to correctly run graphical applications (like YaST).
Caution: When you are root, your home directory is `/root/` and not `/home/peter/`. **cd** will therefore bring you there.
This is just a very brief introduction into the **sudo** command, which can do many more things and which can even be configured. There will be more about it in the chapter about users and groups.
Note: In UNIX, every file and directory has an owner and permission types. You will find out more about it in section "File ownership and privileges" of the chapter "Dealing with files and folders".

Starting programs

Most terminal commands are actually programs located in the directories `/bin/`, `/usr/bin/` or `/usr/local/bin/`. To start such programs, you can simply type their name and the system will look for the corresponding binary in the folders above. For example, you can simply type firefox and the terminal will launch the browser for you. To run a program or script that is not in /bin or `/usr/bin/`, for a example something you just downloaded or coded yourself, you have to type its path, e.g. downloads/program. If the program is in the current directory, you have to type **./program**. Note that you must have the right to execute that file. Read more about it in section "File ownership and privileges" of the chapter "Dealing with files and folders" (coming up).
The advantage of starting programs in the console is that you will be able to see error notifications. If for example you have a program that always crashes at startup you can run this program over the console and see why the program crashes. Even if you do not understand the error description you can still use it to ask a question online (e.g. in a forum).
When you have started a program in the console, that program will occupy your terminal and you will not be able to type any other command as long as the program is running. This is where process management becomes useful.

Dealing with processes

Pausing a process and putting it into the background
[Ctrl]+[Z] will put the program to sleep and let the terminal return the command line. You will notice that the program you have been running is now grayed out by the desktop environment and does not respond anymore. There are two ways to continue running this program:
You can either type **bg** and then the name of the program in parentheses (use tab to find it) to continue running the program outside of your console. Note that you lose control over it by backgrounding it, but its output will still appear in that console.
Instead of backgrounding, you can also continue to execute the process in the console, called foregrounding. For this, type **fg** and the name of the program and it will continue to run in your terminal like nothing happened.
Note: If you simply type **bg** or **fg** (with no name specified), the last paused process will be taken.
If you want to start a program from the console and having it run in the background immediately, place the '&' sign at the end of the command. For example **gnome-terminal &** will open a second terminal window without occupying the first one. Still, the output of the program will appear in the console you started it from.

How to end Processes
If you have a program running in your current terminal you can press [Ctrl]+[C] to terminate it immediately.
To kill a process that is not running in that terminal, you have to go to the system and tell it to shut the process down. There are two ways to do it: either you use an identifier which is unique for every process, or you terminate the process by its name, killing all processes with that name.
Every running program in Linux has a process ID (abbreviated as PID). This is a number that makes each process uniquely distinguishable from the others. The command **ps** shows you the processes that are being run from the current terminal. To see all processes currently running on your computer, use **ps -A**. You can now look for the desired process and address it with the associated process ID. When you know it is PID, it is very easy to end that process with the command **kill pid**.
To kill a process by its name (for example firefox), you can type **killall firefox** and every process with that name will be terminated. This is useful when you just have one process of that name. If you have two processes with the same name and you just want to kill one of them, you have to use the PID method.
There are also console task managers that allow you to scroll through your processes and deal with them. Almost every Linux system has the task manager **top** installed. One can scroll through the list of processes and simply type 'k', to kill the desired process. Leave top with 'q'.
If top is too ugly or unintuitive for you, try **htop** (you might have to install it first, see chapter about the package manager below). **htop** provides a more natural user interface and displays hints about how to use it.
Hint: Similarly to **top** and **htop** for processes, there is **iotop** for monitoring the hard disk's activity, **jnettop** for the network and **powertop** for information about power

consumption. These are very powerful tools, it might be useful to have a look at them.
Note: By default, **kill**, **killall** etc. send a signal to the process requesting kindly "would you please come to an end" and letting the process nicely finish itself. If a program has frozen, it might not react to it. You can then use the option **'-9'** (this is the digit nine, e.g. **kill -9 515** or **killall -9 firefox**) to pull the plug. This will shoot the process right in the heart. It is is a cruel method – only use it if the gentle method has no effect. Sometimes, even the **-9** option does not help and task managers will display the process as a "zombie". This is very rare (for example, it sometimes happened with earlier versions of Skype). When you search online for "How to kill a zombie under Linux", a likely answer will be: "You cannot kill a zombie as it is already dead. To get rid of it, you need to kill its parent". Sounds very funny, but this makes perfectly sense: To get rid of the zombie process, you must terminate the process that started it. If you do not know which one that is, a reboot will get rid of the zombie.

Concatenating commands

To run two commands unconditionnally, one after the other, use **command1; command2**. **command2** will be started as soon as command1 terminates. For example **firefox; thunderbird** will start both firefox and then thunderbird when you have closed firefox. This is very useful when typing commands that take a longer time to finish and you do not want to wait the time the first command to finish before typing the second one.
Every process returns a signal to indicate whether it was successful or not. If you want command2 only to be run if command1 returned the value "everything ok!", use **command1 && command2**. An example to this would be **sudo apt-get update && sudo apt-get upgrade -y** which will install updates only if searching for updates was successful. You will understand these commands in the chapter about the package manager.

More useful commands. . .

which is a command that prints location of the binary of some program. For example **which firefox** will print: `/usr/bin/firefox`. This is very useful when you have some program installed but do not know where its binaries or other associated files are. The output of which indicates which file is actually run when you simply type **firefox**. Most programs are installed in several locations. To get them all, type **whereis firefox** and the output will be: `firefox: /usr/bin/firefox /usr/lib/firefox`, indicating that Firefox has also installed some libraries in `/usr/lib/`. Note that the whereis command might not print all the locations created by the program; for example the Firefox directory `/home/peter/.mozilla/firefox/`, which holds your personal Firefox configuration, is not listed.
cnf tells you which package a command belongs to. This is useful if you want to use a command which isn't available but do not know the exact package to install.
apropos searches the man pages for certain keywords and prints the titles of all the pages that contain it. This is useful if you want to find information about something but do not exactly know where in which man page to look for. You may for example type **apropos vpn** to list all man pages that are somehow related to VPN.
echo simply prints a string. For example, **echo I am so proud** will print `I am so proud`. This becomes useful in scripts or when redirecting the output (see below).
less displays the contexts of a text-only file. For example, to read the contents of hello.txt, type **less hello.txt**. Leave less by pressing the key 'q'.
wc counts the lines, words and characters of a file. Usage: **wc hello.txt**
grep is an extremely powerful tool to search within a file. For example, **grep TODO work.lst** searches for all occurrences of the word TODO in the file `work.lst` and highlights them.
Finally, here is a rather useless but fun command: **yes my-text** will prompt **my-text** until you hit [Ctrl]+[C]. When you redirect its output, you can easily create large files that way.

Output redirection and piping

Redirecting a program's output

By now, you have probably found out that most console commands print their output to the screen. You can actually redirect that output to a file or device.
There are two types of output: Regular output and errors/warnings. Warnings inform you that something is not ideal but that the program can continue despite the problem. An error is a fatal problem that causes the failure of the command, for example if you do **cp source target** and source does not exist. Every thing else (like the output of **ls**) is considered regular output.
To redirect regular output only, add **> target** to the command, where target is either a file or a device. For example **echo Hi there! > greetings.txt** will create a file greetings.txt and write `Hi there!` into it. Note that the content of `greetings.txt` will be deleted and replaced by `Hi there!`. If you want to keep the existing content of greetings.txt, use the '»' operator: **echo Hi there! » greetings.txt** will append `Hi there` to whatever is already in the file. If the file does not exist, it will be created.
To redirect error messages only, add **2> target** to the command. For example, **firefox 2> errors.log** will display regular blablah in the console and write only errors and warnings to errors.log.

As said above, you can as well redirect output to a device and not just to a file. This is mostly used for discarding the output of a program. To achieve this, redirect to `/dev/null`. For example, **firefox > /dev/null 2» errors.log** will get rid of regular output and write only errors and warnings to errors.log. Note that '2»' means keeping the current content of the file and append to the end.
Hint: If you want to redirect all outputs to a location, use the form **mycommand > target 2>&1** where **2>&1** stands for "and put the errors where the rest is".
Caution: If you overwrite a file by accident by using > or 2>, its contents are gone! Make sure you check if such a file exists before redirecting output with a single >.

Piping
Just like the text output, you can redirect the result of a command. For instance, you can take the list generated by **ls -1** and pass it to wc in order to know how many files and folders (excluding hidden ones) there are in the current directory. Note that **ls -1** is not an L but the number one. When called with paramenter **-1**, ls will display every file / folder on a separate line.
The pipe operator is '|' so our example would be **ls -1 | wc**. If you run it, three numbers will be displayed. You know that **ls -1** prints exactly one line per file / folder and that the first number given by wc is the amount of lines in the file specified. However, there was no file specified to wc this time. Instead, we directly handed it the output of ls. Therefore, the first number displayed is the amount of files and folders we were looking for.
By curiosity, what happens if you type **echo These are four words. | wc**? The result is 1, 4, 22: The output of the echo command, which is "These are four words." contains one line, 4 words and 22 characters. Note that the end-of-file character is included in this count, so the visible sentence actually only has 21 characters.
The pipe takes the output of a program and makes it be the input of another program. Sometimes though you want the output of the first program be the arguments and not the input of the following command. For example, if you get a list of filenames, you cannot pipe them to the rm command as rm can only delete files, not text. Therefore you would like to have this list passed to rm as arguments, because rm deletes every file whose name it gets as an argument. To illustrate this: Imagine a pipe like **...| rm** where the '...' command produces the list {file1, file2} as output. rm will now complain that it got invalid operands: It cannot delete the string file1. You want rm to delete the files with the names file1 and file2, so you actually want **rm file1 file2**. This is where the command **xargs** becomes handy: it takes a piped input and passes it forward as argument to the following command. So in this example, the correct command chain would be '...| xargs rm'. Note that there is no pipe between **xargs** and **rm**. Instead, **rm** is passed to **xargs** as an argument.

You can combine all this knowledge: If you want to delete all files in the current folder whose filename contains the word "stupid", run **ls | grep stupid | xargs rm**. Explanation: **ls** generates a string of the names of all files in the current directory which will be piped to **grep**. **grep** takes the incoming string and filters out everything that does not contain the word stupid. The resulting list of filenames, all containing the word stupid, is then piped again to **xargs** which takes its argument (**rm**), appends its input (the list) and runs the whole thing as a command.
A final example: **ls | grep .jpg | sort > pictures.lst** will write a sorted list of all files with the name containing ".jpg" into pictures.lst. Note that this would also include the file thisIsNotA.jpg.doc.

Environment (global) variables
Environment variables are simply variables with a name and a value, which are available to the whole system. They provide a way of sharing configuration data between applications. For example an environment variable may be a path to some relevant files or the name of your default text editor. The command **printenv** will list all currently set variables.
It is possible to run certain applications in a modified environment, meaning an environment where some variables are different. This is done with the **env** command. Let us list some interesting environment variables you may find on your machine:

PATH stores a list of all folders that are supposed to contain executable files. These are `usr/bin`, `usr/local/bin`, etc. So this is why your console knows where to find the Firefox executable, when you type firefox as a command.
MANPATH lists all folders in which the man command will look for manual pages if you for example type `man env`.
PWD indicates the working directory of the currently used terminal. Every time you use the cd command, the PWD variable changes to the new working directory.
BROWSER contains the path to your default web browser.

Environment variables are called global if they apply to all users on the system and local if they apply only to the current user. Global variables should be saved in one of the three files `/etc/profile`, `/etc/bash.bashrc` or `/etc/environment`. Local variables on the other hand should be declared in other files in your home folder like for example `.profile` or `.pam_environment`.
The **export** command is used to set environment variables. For example to include another folder into the directories list of the PATH variable we can type **export PATH=${PATH:}/home/my_folder/**. This simply adds the string `/home/my_folder/` to the original value of the PATH variable separated by a colon.

As you can see ${...} is used to access the value of the variable in between the curly brackets. For example `cd ${HOME}` will actually execute as `cd /home/Peter`.

Scripting: sh and python

Scripts are text documents that can be executed. The main difference between a script and a program is that the script is not compiled but executed. This means that it does not directly run on the processor but it is read by another program, called the **interpreter**, which will execute the commands specified. This makes scripts run slower than actual programs, but you can edit them at any time. Scripts are extremely useful for small tasks that you perform often, like doing a backup, installing updates etc.
A script is a file that is associated to an interpreter. If you want your file to be run by bash, the interpreter is sh. To execute it, type 'sh mybashscript.sh'. Another scripting language is Python which is so powerful that it is actually considered a programming language. The corresponding command would be 'python mypythonscript.py', sometimes 'python3 mypythonscript.py' for version 3 of the python interpreter.
You can also pretend that your script is an actual program by giving it execution privileges ('chmod a+x myfile', remember?) and simply run it with './myfile'. However, the system then does not know which interpreter to pick. This is why in that case you need to make the first line of your script be '#! /path/to/the/interpreter'. For example, for bash scripts, use '#!/bin/bash'. Then you can start you script on the next line.
Both bash and python are HUGE. We cannot teach you much about it here, but there are great tutorials out there which will help you to learn it if you want to. Here are two examples to get you a feel though:
If-statement in bash:

```
if [ "$some_variable" = y ] then
        other_variable = 5
else
        other_variable = 6
fi
```

The same if-statement in python:

```
if some_variable == y:
        other_variable = 5
else:
        other_variable = 6
```

For-statement in bash (this will print wine, then beer):

```
for i in {"wine","beer"}
do
        echo "$i"
done
```

Same thing in python:

```
for i in ["wine", "beer"]:
print(i)
```

Python is more intuitive, but bash scripts have the advantage that they can run on pretty much any Linux system whereas the python interpreter often has to be installed first. Yet, there are many, many other interpreters you can use, just pick your favorite.

.bashrc: Configuring bash

Every time bash starts up, it executes all commands in the user's .bashrc. This is a hidden file in your home folder, e.g. '/home/peter/.bashrc'. Ubuntu ships you a preconfigured .bashrc already. Go ahead and have a look at it using less or nano.
As you see, .bashrc is actually a script. It lets you configure bash. The most important command there is **alias** which lets you define a name that will be substituted by something else when you type it. For example, "alias la='ls -a'" means that whenever you type la, what will be run is ls -a. You can even write something like "alias ls='ls --color=auto'" where ls itself gets replaced by ls --color=auto. Now, you will not be able to access ls without the --color=auto option any more as the string ls gets replaced every time you write it, so be careful.
Such aliases only hold until the console is closed, then they are forgotten. However, as you read above, all commands in the .bashrc script are executed every time bash is started. This means, if you write an alias command into this file, the alias will be active all the time.
Remember: .bashrc is a regular script. You can do anything in there, even set environment variables.

ssh: Accessing a Linux computer remotely

SSH stands for secure shell. It is a network protocol that is used to securely exchange date between computers. SSH is often used to remotely access other computers over the internet. This means accessing and operating a different machine through your own interface. ETH offers students remote-usage of Linux machines on a server called Optimus. This is for example useful when you need to run calculations on a Linux machine and do not have one or when you simply require more computing power. When connected with the ETH server you can directly use the applications installed on this server.
As an example we will show you how to log into the optimus server:
In your console type "**ssh** user@optimus.ethz.ch" where 'user' is your ethz username.

Then you will be asked to type in your password. Once logged in, you can access and operate on the files that are stored on your ETH Linux account. To log out, you can either type logout or use CTRL+d. In case ssh is not installed, you can install it with "sudo apt-get install ssh".
If you need to copy files to or from the server you can use the the **scp** command. It is used exactly like cp, but copies the files over the ssh protocol. You will also be asked to type in your password.
Attention: Do not confuse sh (Linux shell) and ssh (remote access to a Linux computer)!

VPN

VPN stands for virtual private network and is used to create a direct connection to a remote network. This may allow students to connect and surf through the ETH network even when they are not at ETH. For example you can use this when you are in the library of a different university or have access to any Swisscom Wifi but still have to register or pay to actually use the Wi-Fi. VPN allows you to bypass this by directly connecting to the ETH network. Note that this bypass is only possible because Swisscom and ETH work together, explicitly allowing you to get around the payment. It will not work on most other Wi-Fi-Hotspots other than Swisscom or with other VPNs than ETH.
Although ETH advises you to install the official Cisco VPN client, there is an open-source alternative called openconnect. You will need to install plasma-nm-openconnect for KDE or NetworkManager-openconnect-gnome for GNOME and NetworkManager-openconnect. To connect, simply create a new connection (type openconnect) to sslvpn.ethz.ch, the rest works automatically.

Users and groups

This chapter tells you how to manage users on a Linux machine. It also introduces important theoretical concepts that you will use in the next chapters.
Users are the people who use a computer. They have a user name which is either their real name or a fictional one. Users can also be grouped together into so called groups. A group can consist of 0, 1 or more users, and every user can belong to multiple groups. The concept of users and groups is important for managing the permissions to access files and folders which we will cover at the end of the next chapter. Usually every Linux user have his own home folder, a folder where he has the sole right to save his files and that other users cannot access. Root is a special user who has the right to manipulate all files on the system. To run any command under the name of the root user, you can use the **sudo** command.

Managing users and groups
When you are on a Linux machine you can use the **who** command to list all users that are currently logged into that machine. This might be interesting when working on servers that support multiple users at the same time while on your personal machine you will usually be the only user logged in.
The command **useradd** is used to add a new user to the system. **-m** indicates that you want the user to have his own directory in /home. **-g** defines the name of the group the new user will belong to. If nothing is specified, by default Linux will create a new group with the same name as the new user and add the user to this group. **-G** can be used to already add the new user to additional groups. The parameter expects a list of group names separated by comas. Finally the **-s** parameter is used to specify the default shell of the user. This might be /bin/bash. Let us create a new user account for Peter:
`sudo useradd -m -s /bin/bash peter`
This creates an account for the user Peter who initially only belongs to the group peter. Now to set a password for Peter we must use the **passwd** command. "passwd peter" will prompt the user to add a new password for Peter.
Deleting users can be done with the **userdel** command. The command will only remove the account and some account related files but it will not remove any files from the user's home directory unless **-r** option is added. -r will remove all private files of the user.
Note: To find out to what groups you belong, open a terminal and type: groups
Note: On openSUSE, you can also use YaST to do this using a GUI.

More about sudo
So far, you have used sudo only to become root. However, sudo is actually a program that allows you to become a different user. The syntax of this would be **'sudo -u otherusername mycommand'**. Similarly, the option -g can be used to run a command as a specific group. If you omit all options, the default action is to become the user root. To see the full story about sudo, type 'man sudo'.
The sudo command can be configured in many ways. **/etc/sudoers** defines which users are allowed to use the sudo command. **DO NEVER edit this file with a normal text editor.** There is a special command called **visudo** that can be used to securely edit the sudoers file. If for any reason you manage to save some invalid syntax into this file, the whole file may become invalid which will lead to the inability to call the sudo command ever again. And without the sudo command you will not be able to repair the file. Visudo is safe because will complain when it sees wrong syntax, preventing you to write your malicious changes to disk.
Of course, you need to be root to edit the sudo configuration file. Therefore the command is 'sudo visudo'. If you are asked what editor you want, type nano, vi or whatever editor you prefer. You will then see the configuration file. Everything is commented so that you can see what each option does. If you do not understand something, use your favourite search engine to get more info. If you are unsure, experiment using a virtual machine or an installation media in live mode first, because you can easily lock yourself out if you mess up the sudo config file.
Lines starting with a # are commented out and will be ignored (for sudo it is like they do not even exist). The sudo config file tells you to uncomment certain lines if you want to use specific features. This simply means to remove the # at the beginning of the line.
One of the most interesting lines is:
Uncomment to allow members of group wheel to execute any command
%wheel ALL=(ALL) ALL
This defines that any user in the group 'wheel' may become root by using sudo typing his own password. When you remove this, you will not be able to use sudo any more! This means that if you do not know root's password, you have locked yourself out of your system.
Here, you can configure sudo to give sudo rights only to specific users or other groups than wheel. You can even let users become root without typing their password (security risk)!
You are probably happy with the sudoers file openSUSE provides you by default. In this case, you can call this paragraph curiosity and you will not need to worry about it. However, there are distros which do not come with sudo. There, you have to login as root directly, then install sudo, configure it and add yourself to the wheel group

manually. If you want to make your own very fancy sudo configuration, go ahead and happily do that. You will find much more information about sudo and visudo online.
Note: sudo only works in the console. Imagine you have a launcher on your desktop that starts your favorite program, call it foo, and you want foo to be run as sudo every time you double-click that launcher. If you set 'sudo foo' as the command to be run, it will not work because you would have to enter your password, but there is no console open to type it into. This is where the command gksudo becomes handy. It does the same thing as sudo, but instead of asking you for entering your password in the console, it opens up a window where you can enter it. The command would then be 'gksudo foo'.
Note: There is a fun hidden feature in sudo that will insult you (but in a nice way) when you mistype your password at sudo prompt. For example, instead of printing "Please try again", it could say something like: "This mission is too important for me to allow you to jeopardize it". There is a large pool of funny insults that are randomly displayed every time you type the wrong password. To enable this feature, ucomment 'Defaults insults' in the sudoers file (using visudo).

Some predefined Linux groups
wheel is an administration group. Users who belong to the wheel group have additional system privileges and control the access to the sudo command.
root members have complete control over the system.
wireshark is a group used by the software Wireshark, an application that monitors the network. When installing the software, users should be added to the wireshark group to be able to capture network packets instead of running Wireshark with the sudo command.
Users belonging to the group **vboxusers** are able to run virtual machines under VirtualBox.

Creating groups
New groups can be created using the **groupadd** command. For example **groupadd adults** will create an initially empty group called adults. With the methods explained in the chapter about file permissions, you could now change the permissions of some files so that they can only be opened by members of the adults group.

Adding users to groups
There are two commands used to add users to other groups. **usermod -aG** will the user to a list of groups. For example "usermod -aG audio,video peter" will add the user peter to the groups audio and video. On some distributions, it is necessary to belong to these groups to be able to use the computers audio and video capabilities. Another way to add a user to a specific group is the **gpasswd**. "gpasswd --add peter audio" will add peter to the audio group. To delete a user from a group, type "gpasswd -d audio peter".

Dealing with files and folders

This chapter is about concepts that are useful when handling files and folders on Linux. We will first give you some theoretical knowledge about data management in general. Then, you will learn what you find in which folders on openSUSE. Finally, we will show you powerful concepts and their corresponding commands for dealing with files efficiently.
Note: This chapter is probably the hardest one of this course and may seem and very technical to a beginner. However, it is also one of the most important topics we are talking about. What you will learn here is crucial for understanding and solving various problems related to the installation and even the daily use of any Linux system.
In the section "Theory", we will have a macroscopic look at drives in general. This information will help you managing your hard drives and USB sticks. Also, it is very useful for installing operating systems.
In the following sections, we will see how files are organized in openSUSE and Linux in general. You will learn what it means to mount drives and you will understand the concept of file and folder privileges.

Theory

Partitions, file systems, partition table: Theory
Consider a so-called block-device like a USB key, a hard disk drive (HDD) or a solid state drive (SSD). All of these allow you to store files and folders and they all share the same way of organizing data. We are going to give you a quick overview of this extremely sophisticated topic. As we are only talking about software concepts, forget what kind of device of the ones listed above you are using; let us just name it a drive. Every drive contains disk space subdivided into so-called partitions. A partition is a slice of the available space that will contain your data. Most drives that you know only have a single partition. If you have two partitions, your file manager (Windows users: Windows Explorer, Mac users: Finder) will show up two disks, but they are on the same device! You can imagine your available space like a cake which you can cut into pieces. If you have a 500 GB hard disk, you can divide it into a 200 GB partition for your OS and a 300 GB partition for your videos. Note that every drive that can store data contains at least one partition.
Every partition has a so-called file system (FS), also called formatting. This is a table that stores the file name, its location (the folder it is in), its size, its owner, permissions and much more. As a consequence, the name of a file is not stored at the same place as the actual data it contains. All this info sits in a different location, managed by the file system. The file system therefore always takes away a few percents of your disk space, this is why you can never actually store 500 GB of data on a 500 GB drive. There are many types of file systems, each with advantages and drawbacks. We will list the most important ones below. Most FS are made for storing files and folders, but there are special ones like the "swap" partition. Swap-formatted partitions are unable to contain files. However, they can store portions of your RAM. Every Linux system typically has a swap partition that is once or twice as big as the available RAM. When you have too many programs open and your machine runs out of memory, the system would usually crash. But thanks to swap, unused parts of your memory (like a loaded picture you are not using right now) can be moved from the RAM to the hard drive. This is called "swap out", which explains the name of that particular partition. Of course, writing parts of your RAM to disk is a horribly slow process which will make your system freeze for a few seconds. When you switch windows, requiring some swapped out data, the system needs to swap it back in and swap out now unused data, causing more lag. This is annoying, but at least your system does not crash. Note that swap is only used when the computer does not have enough RAM. If it happens often, consider buying more RAM.
In brief, devices contain partitions which store your data which is organized by the file system. But how are the partitions organized, is there a sort of file system for them? The answer is yes, it is called the partition table. This table tells the system which partitions are on a device and where to find them. Just like file systems, there are different types of partition tables. We will consider two below.

BIOS and two types of partition tables
Again: The partition table organizes partitions, slices of your hard disk space. They are recognized by the computer itself. Every PC has a little software, called the BIOS which is installed directly on its mainboard. The BIOS is not an operating system. It checks the computer's health and handles basic tasks like controlling the fan speed. The most important task of the BIOS at startup is finding an operating system to boot (meaning: starting the operating system after the computer was turned on). For this, it will read the partition table and check which partition the OS is on.
There are two major types of partition tables:
1. MBR: This is the "good old" format, which was used on PCs until about 2012. In 2015, almost all computers still support it using so-called "legacy mode". MBR allows you to have up to 4 primary partitions. Or you can have up to 3 primary and 1 extended partition. The extended partition serves as a container for so-called logical partitions and can contain a large number of them. A little amount of space in the

front is called the Master Boot Record (MBR), which tells a starting computer where it can boot from. Only one operating system can be referenced to. However, there are programs like GRUB which allow you to have multiple operating systems (called multi-boot) on your computer. They are a little OS that displays a table of options to boot and then forward to the actual operating system.
2. GPT: With the arise of a new type of BIOS, the UEFI-BIOS, a new type of partition table was introduced: GPT. Before, a special version of this format was used by Macs. GPT allows you to have a large number of partitions on the same device. There are no more extended or logical partitions. An MBR does not exist. Instead, there is a partition called "EFI" which saves information about bootable partitions. This allows multi-boot even without boot managers like GRUB.
Caution: Moving or altering partitions is a dangerous (and sometimes very time-consuming) process: If anything goes wrong, you might lose all data on the disk! Therefore, always make a backup before playing around with partitions.
Note: You cannot simply switch from MBR to GPT and vice versa. You have to wipe the entire disk and reinstall all contained operating systems.

The most important file systems

There are many types of file systems. Here is a list of the ones you might encounter the most:
1. FAT32: Ancient file system that can be read and written natively by Linux, Windows and Mac. It is standard on smaller USB keys. However it only supports files of a maximum size of 4 GB. It does not support file ownership or privileges. Also, this FS suffers from fragmenting, meaning that files are spread all over the hard disk. If it gets too slow, you have to run a defragmenting utility.
2. exFAT: More recent alternative to FAT32, will hopefully soon replace it. It is widely compatible with most modern operating systems, but it does not have the limitations of FAT32.
3. NTFS: FS made for Windows in the late 90's. It supports ownership and file privileges to some extend and the maximum file size is large enough to fit any of your files. Unfortunately, this FS suffers from fragmenting. Also, Macs can only read, but not write to NTFS partitions. You can install the Tuxera driver, but it is very slow.
4. HFS/HFS+: FS made for Macs. Windows does not support it at all, Linux can only read but not write. No defragmentation needed, file ownership and privileges supported.
5. ext2/3/4: FS made for Linux. Windows and Mac do not support it unless you have the right driver (and it is hard or impossible to get one now). ext2 is outdated, ext4 the current version and ext3 the "good old" version. Does not suffer from fragmentation, file ownership and privileges supported.
6. Btrfs: Brand new FS which supports cool features like taking snapshots of folders in a fraction of a second. No defragmentation needed. Btrfs is officially marked as stable but it is still quite young and therefore somewhat unstable. It is getting more and more popular, file ownership and privileges are supported.
Your Linux operating system must be installed either on a ext2-4 or btrfs formatted volume; usually, you will pick ext4. The EFI partition of a GPT disk is formatted in FAT32.
Note: A very powerful, advanced and yet user-friendly program for handling partitions and formattings is **gparted**. There are also console tools for every single task; check online if you want to learn more about them.
Caution: Modifying partitions is dangerous and can cause total loss of files: either because you accidentally deleted or formatted the wrong partition or due to hardware or software failure. Therefore, always back up your data before doing such modifications.

The openSUSE file hierarchy

You now know how files are organized on a drive. The following section aims to provide some overview about how the files are organized in the folder hierarchy in openSUSE. The organization may vary between different distributions and therefore the following is only guaranteed to be true for openSUSE. Note that many of these folders present usage is for historical reasons; the files are not strictly organized thematically. Some new distros come with a completely new and different file hierarchy in order to clean things up again.
Note: There is no need to remember or even fully understand the file hierarchy. This section shall give you a feeling about where files are typically located, as this is very different from other OSes.

The root directory

The root directory, written as "/", is the top of the hierarchy. Everything that is part of the file system is in somewhere below the root directory. When you use the **cd** command to call a directory with its full address, the address needs to start with '/. If you just type 'cd /' the current directory in the terminal will be changed to the root directory. This is valid on any Linux system.
Note: There is nothing like C:\ in Linux. Everything is under /, even USB disks which you usually find in /media/ (see below).

/bin and /sbin

The /bin directory contains executable files (programs). It contains only the most critical programs for system start up. Similarly, /sbin contains programs which usually

are only available to the root user.

/boot
The Linux kernel and other files that are needed by boot managers like grub to boot up the system.

/dev
Every device in Linux is treated as a special file. It will be then opened for access by mounting it (a section about mounting is coming up).

/etc
This directory contains configuration files for programs and other things shared by all users.

/lib
System libraries used by programs in /bin and /sbin.

/lib64
Like lib, but for 64-bit systems

/home
This is where the users data is located. /home contains a folder for every user. If your name is Peter you will probably find a folder called Peter. The visible sub folders inside are Desktop, Documents, Downloads, Music, Pictures, Public, Templates and Videos. However there are vastly more hidden folders inside your home folder. In many file browsers you can use Ctrl+h to make them visible. In Linux every file or folder that starts with a '.' is meant to be invisible.
The hidden files and folders are mostly created and used by programs to store your settings. For example there is a folder called .mozilla where you can find your Firefox bookmarks and history.

/mnt
Contains directories for deprecated mount points, used to make other devices visible in the file system hierarchy. For example, after executing the command 'mount /dev/cdrom /mnt/cdrom' you can access the content of the cdrom drive at /mnt/cdrom. We will talk more about mounting below.

/usr
The /usr hierachy contains most of an openSUSE system. Also, it contains most of the programs installed on your system. It has subdirectories bin, lib and sbin, which contain the programs and libraries not contained in /bin, /lib and /sbin. The share directory contains data shared among programs such as fonts, icons and documentations.

/opt
/opt contains directories that are structured like /usr. This is used for software that uses it's own version of some of the parts of your system, which is mainly proprietary software.

/var
The /var hierachy contains lots of variable data your system produces. For example /var contains your printer's queue, your event log and your package database.

/srv
If you install a web server on your machine, this is where the web sites go.

/run
The /run hierachy contains lots of internal data needed for running services on your system to talk to each other, for example pid files which tell programs which pid a given service jas.
Note: To find out where a program is installed, type 'whereis myprogram' or 'which myprogram'.

Mounting and Unmounting

When you plug a USB stick into your machine, it will show up as a file in /dev/. For example, if you have an internal hard drive, its second primary partition will be /dev/sda2. The first extended partition of your USB stick is typicalle /dev/sda5. This varies between systems!
You can not do much with such a file other than reading or writing bits. This will be discussed in "bitwise copying" later. So, how do you get to see your folders on the stick? The answer is mounting. By mounting your drive, you open up its device file and make its contents accessible for typical operations.
First, you need to find out which device your drive is. This can be done using the command "lsblk" or "sudo fdisk -l". All partitions of all recognized devices will show up. Compare the sizes to find out which one is your desired device.
Now we are going to mount your drive. Assume you want to access the first primary partition on the second device. Then, your device would typically be /dev/sdb1. You could mount it to wherever you want. By convention, we usually mount to /mnt/. Create a folder to which you want to mount to (for example using the mkdir command. For creating folders in /mnt/, you need to be root, so do not forget sudo). The folder we want to mount to should be empty. Let us say we do 'sudo

mkdir /mnt/mydrive'. Then we can type 'sudo mount /dev/sdb1 /mnt/mydrive'. If the command was successful, the folder /mnt/mydrive/ now contains the content of your drive.
When you are done using your drive, close all the programs accessing it and unmount it: Either type 'sudo umount /mnt/mydrive' or 'sudo umount /dev/sdb1', both have the same effect. When the command is done, you are safe to remove your drive (make sure you have unmounted all partitions on that device!). If you have an open file or folder on the drive, umount will complain and refuse to work. This is also the case for consoles which are still running on the drive. Typing cd will navigate away from it, allowing you to unmount it.
On openSUSE, USB keys are automatically mounted for you when you plug them in. They typically go to /run/mount/username/drivename/. For unmounting a drive, you can click the little eject-button in your file manager. Note that it will refuse to do it when you mounted a device with "sudo mount"; you have to unmount these using "sudo umount".
You can mount anything to anywhere. Mounting may seem painful at first sight, but it is an extremely powerful concept. You can mount internet servers, cds, files etc. and they will all show up the same way, allowing you to forget where the data actually is (in your computer or thousands of kilometers away). Also, there are many options for mounting (for example read-only or read-write). To know more about mounting, check the manpages or online - it is a huge topic.
It is possible to automatically mount drives at startup. To do this, use the file /etc/fstab. For example, it allows to automatically mount a network share as soon as it becomes available. That way, when you get home, it automatically becomes available. fstab is well documented online and it is worth reading more about it.

Hardlinks, symbolic links

Links are a concept you might have already encountered with Windows or Mac, when you wanted to place the link to an executable file or a directory onto your desktop for faster access. Linux distinguishes between hard links and symbolic links (also called soft links).
Hard links refer to the specific physical location of a file while symbolic links only contain the symbolic path to a file (i.e. /my/file/path.txt). If that file is moved, the symbolic link becomes invalid while the hard link still points to the physical location of the file. Because folders are abstract structures that rely on their path name, hard links cannot refer to folders but only files. Symbolic links can of course link folders as they work by specifying the file path.
Hard links can be created with the **ln** command. **ln -s** on the other hand is used to create a symbolic link. For example "ln -s dir sym_link" will create a symbolic link to dir with the name sym_link and "ln test.txt hard_link" will create a hard link to test.txt with the name hard_link.

File ownership and privileges

You will probably think that you might rarely use the information in this section, as it allows you to do very sophisticated and advanced file protection. However the moment will come when you will be very thankful for knowing this stuff. Motivating example: You run 'sudo su', do some administrative stuff and then return to your home folder /home/peter while still logen in an root. You enter 'nano shopping-list' and save a list of groceries you want to buy in the afternoon. You turn off your computer and have breakfast. Suddenly, an important product you need to buy comes into your mind. You turn your PC back on and type 'nano shopping-list'. You add the missing entry and try to save... Permission denied! Why? Read ahead to understand what happened there.
As you have read above, Linux' file systems support file ownership. This means that every file and folder has a user and a group it belongs to. Only the owner has the right to set the privileges of the file (except for root, who can do anything on any file of course). The owner of a file can make someone else be the owner of it, but you cannot make someone else's file yours unless you are root.

Privileges

There are mainly three types of privileges: Reading, writing and executing.
Read: For a file: View its contents. For a folder: See its contained files, but not their size, owners or privileges. Also, you cannot cd to a read-only folder.
Write: For a file: You can rename, move or delete the file and edit its contents. For a folder: You can add or remove files to/from the folder and change the privileges and properties of it is contained files or folders.
Execute: For a file: You can run the file as a program. Under Linux, there is no .exe extension for programs - instead, the executable privilege is set. For a folder: You can cd into the folder and view the contained files' size, owners and privileges.

Owner, group, others

Every one of the three privileges can be assigned to three types of people: the owner (the person the file belongs to), people in a specified group and everyone else. You will therefore have 3x3=9 properties: Read privileges of the owner, read privileges of the users in the group, and so on until execute privileges of the others.
Owner: This is either the user that created the file or who received it via chown (seen later).
Group: Every file is also associated to a group. On Ubuntu, if your username is,

say, peter, there is also a group with the name peter, containing only the user peter. When you create a file, its group will be set to the group peter. The group privileges specify the rights that all users in that group have on that file.
Others: Everyone who is neither the owner or in the group of the file.
This becomes useful if you have, say, a series of documents that you want to share with other people on your computer (which could be a server accessible from the internet). You want to be able to read and edit the documents from your account, and there are a few people who shall be able to read (but not write) them. Every one else should not be do anything with those files. In this example, you would set owner to read-write, group to read and others to nothing. If you set the group to "readers", you can give a user read privileges to all those documents by simply adding that user to the "readers" group.
Note: By default, when you create a new file under Ubuntu from your account "peter", the file has owner peter, group peter and privileges as follows: read-write for the owner and read-only for group and others.
Now, you should be able to understand what happened in the motivating example from above. Well, after doing your administrative tasks, you forgot to exit from sudo mode. Therefore, you were still in sudo when creating your shopping list and, of course, that file now belongs to root and not to you (peter)! In the next paragraph, you will see how to use the chown command to become owner of the file.
Sooner or later, you will run into a problem of this kind, and then you will be thankful for understanding what is going on.

Commands for dealing with privileges
If you want to know the rights of an existing element, use the command 'ls -l' or 'ls -la' if you want to see hidden files too. You will get an output like: "drwxr-x--x 2 peter editors 4096 2. Feb 09:20 myfolder/". The d stands for directory, as this is a folder. For files, there will be a dash ('-') instead of the d. Then come the rights for the owner (user): they are rwx, so the owner can do everything. The group has r-x, which means that they can read and execute, but not write. Finally, others have privileges --x, allowing them only to execute. The owner of the folder is peter and the group is editors. The filesize of the folder is 4096 bytes (which is the default size of any folder on this kind of system) and it was last modified on February 2nd, at 09:20 AM. Finally, the name of the folder is myfolder. There is a slash ('/') because it is a folder.
Now we will see how you can edit the privileges of a file or folder. You only need to know two important commands: chown and chmod.
1. chown: This command is used to change the owner and/or the group of a file. The syntax goes as follows (citation from the man pages): 'chown [OPTION] ... [OWNER][:[GROUP]] FILE'. Consider your shopping list from the motivating example above. To make the file yours, cd to its folder and type: 'sudo chown peter shopping-list'. Note that you must do sudo unless you are the current owner of the file. If you want to change the file's group, put a colon in front of the group name: 'chown :name-of-the-new-group shopping-list'. No sudo required, since you made yourself the owner in the first chown command. Finally, you can change owner and group at the same time, by typing '[sudo] chown peter:name-of-the-new-group shopping-list'.
2. chmod: Now that you have set user and group of the file, you are ready to set their respective privileges. If you are not the owner of the file, you must execute with sudo. There are two ways to set privileges:
2a. chmod using absolute values: You will enter three digits to set privileges to a specific setting, ignoring current values. The first digit corresponds to the privileges of the owner, the second to the group's and the third to the other's privileges. Each digit is a number between 0 and 7. It represents the sum of some the values 4, 2 and 1 where 4 stands for read, 2 for write and 1 for execute. Let us take for example the digit 6. It can be split into 4+2, meaning that you can read and write, but not execute. 5=4+1 means you can read and execute, but not write. 2 stands for writing only. 3=2+1 means write and execute, but not read. 0 means no privileges at all and 7=4+2+1 means all privileges. Remember, there are three digits, so 'chmod 123 shopping-list' means that the owner can only execute, users in the group may only write and every one else can write and execute, but noone can read as none of the digits is larger than 4. 'chmod 754 shopping-list' means that the owner can do anything, users in the group cannot execute and everyone else can just read.
2b. chmod using relative values: Instead of forcing a new value which overwrites the old one, you can add or subtract privileges to/from the existing ones. We distinguish four letters: o for others, g for group, u for user (which is the owner, but the o is already taken for others) and a for all of them. Another three letters are introduced: r for read, w for write and x for eXecute. There are three operators: +, - and = which are self-explanatory. If you want to give the group the right to write (without changing the privileges for reading or executing, and neither those of the owner or the others), you can type 'chmod g+w shopping-list'. You can combine different settings: 'chmod a+rwx shopping-list' gives everybody the right to read, write and execute. Another example: 'chmod o=rw shopping-list' sets the rights for others to exactly read-write, no execute permitted. Finally, 'chmod u-w shopping-list' disallows the owner to write to that file.
Note: If you own a file, you can change its privileges in most file managers by right-clicking and then choosing "Properties".

The package manager

This chapter is all about installing, maintaining and removing software automatically. The package manager is probably one of the greatest strenghts of Linux systems compared to other OSes.

Theory

In most Linux distributions, you do not need to (un)install or update software by hand. Software is organized in so-called packages which are handled for you by your package manager. There are two types of packages: excecutables and libraries. The first are programs you can actually run. The latter, called libs, are used by executables. Note that the same library can be used by several programs (e.g. a sound library for decoding mp3 may be used by different players like vlc, clementine, rhythmbox etc.). This saves hard disk space.
We distinguish between updating (installing updates, replacing installed software by a newer version), upgrading (Update from one version of a distribution to another) and installing (acquiring new software).

Update

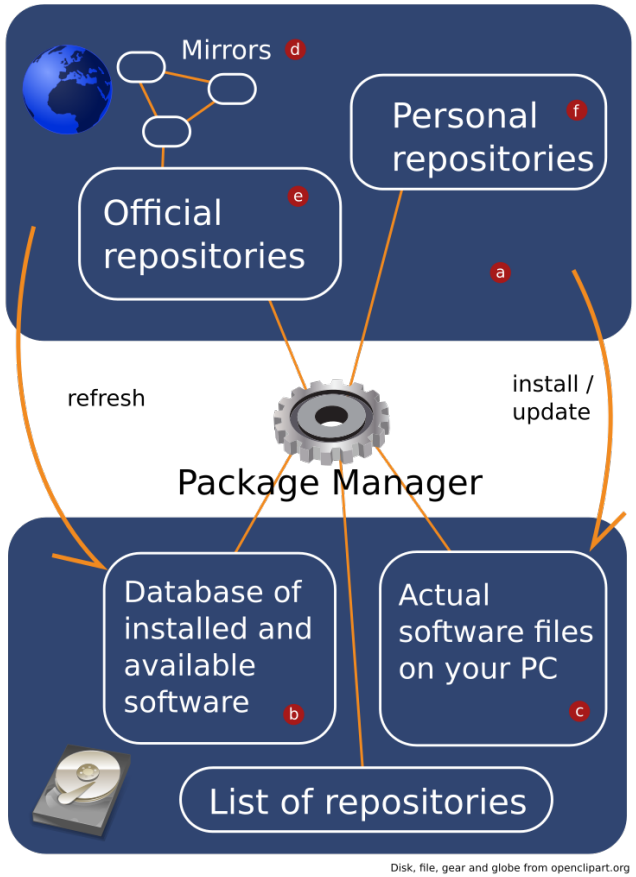

Figure 1 – *Schematic representation of the package manager's functioning.*

Have a look at figure 1 on page 29. The Package Manager (let us call it PM) has its own database (b) to keep track of installed and available software. To know which packages can be installed from the web and where to get them, the PM checks a list of servers, the so-called repositories (a). It then goes online and downloads the newest package information from these. This process is called "refresh". Note that this does not install any updates - that would be called updating - but it only refreshes the information the PM has about the available repositories.

Sources are accessed by channels, called repositories (repos) in general. These can be the official servers of your distribution (e) or external sources (f). The latter are often subdivided into personal repositories (k).
In brief, updating means that the PM collects a list of all packages that can be installed (collected and merged from all available software sources) and the most recent version number of each package. If the same package exists in different sources, the most recent version from the source you have used last for the package will be chosen. This is called vendor stickiness and prevents sombody from accidentially overwriting important packages with incompatible versions.
The package manager now knows which of the installed packages (c) are outdated and which software could be installed on request. On a typical openSUSE machine, there are tens of thousands of packages available and several hundreds installed.
Available packages are downloaded from big servers, so-called mirrors. For openSUSE, there are several mirrors per continent, all offering the same packages and synchronizing new versions between each other. This allows to distribute the workload and prevents the failure of a single server due to overload.

Updating
After updating your software sources, the package manager knows which of your packages (h) have a lower version number than the one available online (i). Updating means that the PM downloads the new packages from the mirrors and extracts them onto your file system (c), replacing your old software by the new one.

Upgrade
On version-based distributions like openSUSE or Ubuntu, a snapshot of time is taken, forming a distribution version. This is like Windows 8 vs. 8.1 or OS X Mavericks vs. Yosemite. However, distribution versioning goes much further than that, as it also concerns applications managed by your PM (and not just the OS). For example, Ubuntu 14.04 was released in spring 2014 and only contains software released by then or compatible with software released by then. For the next few years, there are security updates, but some packages might be kept at a maximum version. For example, GIMP will not be updated to more than 2.6.12 on an Ubuntu 12.04 setup. In contrast, Ubuntu 14.04 comes with GIMP 2.8.10. Each version of your distribution has a restricted range of versions for each program. This ensures that the packages on your computer are compatible between each other. In order to get a newer version of a program than the latest one your distribution version supports, you must either upgrade your entire distribution or switch to an unofficial repository where someone backported the program for your version. Upgrading your distribution usually consists of replacing your software sources by the ones for a later version (l) and then upgrading. The trick is that all packages are upgraded simultaneously, causing an enormous installation process of several thousand packages. Sometimes, you have to fix compatibility issues by hand after this. You then have a newer version of your distribution. There are also distributions which do not have distribution versioning. That model is called "Rolling Release". These always provide you the newest software available, but you must deal with compatibility issues yourself. Rolling Release Distros like Arch Linux or the "Tumbleweed"-branch of OpenSUSE are therefore typically used by more advanced users.

Dependencies
This is one of the most important tasks of the package manager. On Linux, packages can depend on each other. To understand this, let us have a look at the package file-roller, which installs "Archive Manager", a simple program for (de)compressing ('(un)zipping') files. This lovely piece of software only consists of a window with user interface elements like buttons etc. It is unable to handle any kind of archive. Wait, what? An archive manager that can not handle any archives? You heard right. Actually, file-roller is only a graphical user interface (GUI) for dealing with archives. It takes your request and translates it into a command that another program, which will deal with the actual archive, will understand. file-roller cannot display any buttons by itself either; it relies on external libraries for doing that. It cannot be run without those libraries, therefore they must be installed before running file-roller. That is why those libs are called dependencies of file-roller.
Now, there are programs like 7zip, the software you need to deal with .7z files, that you do not necessarily need for running file-roller (assuming that you never use .7z files but only .zip files and that the zip utility is already installed on your system). file-roller knows about 7zip and checks at each startup if that package is installed. If so, it offers you to open, extract and create 7z-archives. If 7zip is not found, file-roller pretends to know nothing about it. So you can extend file-roller's capabilities by installing supported archive tools. These are called optional dependencies, as you are free to install them if you need them.
The cool thing about dependencies is that everything can depend on anything and you can reuse the same package for tons of purposes. For example, when installing 7zip, not only file-roller, but also all other archive managers that you have installed and that support it will detect 7zip and offer you additional options. The bad news is that especially in Linux, this possibility is highly used and there can be what developers call the "dependency hell": A depends on B and C which depend on E and so on.
Luckily, you have a powerful friend here: the package manager. Each package comes with information about its dependencies which is automatically read by the PM. Thanks to this, you do not ever have to worry about dependencies in practice (unless someone messed something up). Just do not be surprised if the package manager

wants to install more than what you asked it for - it knows what it is doing. Note that optional dependencies are not automatically installed for you, as they are not necessary for running the program. You have to request their installation yourself.

Installing/Uninstalling
If you have understood everything so far, this is going to be simple for you. When the user requests the installation of a package, the PM checks in its database (b) if a package with the specified name exists. If yes, it downloads the necessary files from the mirrors (d) and extracts the packages onto your disk (c). Also, the PM checks the program's dependencies and installs them for you (unless they are already installed).
Uninstalling is even simpler: If the package is installed on your system, it will be removed. Then, the PM does something magic: if there is a package on your system that was installed as a dependency (meaning, you never explicitly asked for it to be installed) and if there are no more programs on your computer that depend on that package, it will be removed as well! The package manager is your friend!

Adding new Sotware Sources
Sometimes, you may want to install software that your distro does not provide. Developers can setup their own mirrors or use existing external services to distribute their software to package managers. On openSUSE, developers mainly use the openSUSE Build Service where everybody gets his/her "home" repo. Multiple developers can form Projects which can then provide updated packages for older distributions(say, the Mozilla project for Firefox or Thunderbird) or submit packages for inclusion in the next version of openSUSE.
Adding a Software Source is typically done in three steps: 1. Tell the PM that you want to use the repo, either via YaST or using zypper on the shell. 2. Update your package information. The PM will now also load the available packages from the new repository and be aware that they are available. 3. Ask your package manager to install the desired software. If you already installed it beforehand, you will need to force a vendor change(by clicking on "Switch ... to versions from this repository" in YaST for example).
Once you have added a Software Source to your system, it becomes fully part of your computer. You can forget where you got the software from - the package manager will take care of it as if it was an official source of your distro. Also, software that was added that way will be automatically upgraded together with all the other packages managed by the PM.
Caution: When you add a new source, you leave the trusted, official distribution sources. The provider of your source could potentially bring malware into your system, so make sure you only add Software Sources of providers you trust!

Practice: openSUSE

Updating and upgrading the system
Usually openSUSE will tell you when there is something to update. If not you can look for updates by using the shell:
`sudo zypper sh`
will give you a special "packet management" shell. It works like a normal shell would but has special commands for package management(type "help") to see them all. Here *sudo* specifies that we want to run the command with root privileges and *zypper* is the name of the packet manager. Type
`ref`
to manually refresh information about your repositories.
Now it is time to install the updates. Type:
`up`
The packet manager will show you all available updates, including new dependencies. If you want to install them, simply hit Return, otherwise type n
If you output looks like this

```
zypper> up
Loading repository data...
Reading installed packages...

The following 8 package updates will NOT be installed:
cairo-devel freetype2-devel libcairo-gobject2 libcairo-script-interpreter2 lib

Nothing to do.
zypper>
```

some updates are from different repositories(see vendor stickiness). If you are sure that you want to install them you can type
`dup`
to disable the stickiness checks and install them anyway.
Caution: You can interrupt the download process using Ctrl+c. However, **do not interrupt the package manager when it is actually installing**. Interrupting a running install process (also update or removal) might result in a broken package system or even cause failure booting the system.

Upgrading your entire distro
As you have read in "Theory" above, OpenSUSE supports jumping from one version to the next one. If a new version is available, you can download the installation image, boot it and use it to upgrade the existing system(the installer will ask you if it detects an existing openSUSE installation) or you can perform the upgrade from

your running system. If you want to perfrom it from the running system, you will need to do the following:
`sudo zypper sh (gets you a zypper shell)`
`mr -adR (disables all existing repositories)`
Now your PM has forgotten all currently used repositories. To add the new distributions' repos, use the following(replace <version> by the version number of the distribution you want to upgrade to, ex. 13.2):

```
ar -n "openSUSE-<version> OSS" http://download.opensuse.
    org/distribution/<version>/repo/oss/ repo-<version>-oss
ar -n "openSUSE-<version> Non-OSS" http://download.
    opensuse.org/distribution/<version>/repo/non-oss/ repo
    -<version>-non-oss
ar -f -n "openSUSE-<version> Updates OSS" http://download.
    opensuse.org/update/<version>/ repo-<version>-update-
    oss
ar -f -n "openSUSE-<version> Updates Non-OSS" http://
    download.opensuse.org/update/<version>-non-oss/ repo-<
    version>-update-non-oss
clean -a
```

At this point, the PM knows about the new distribution. Now you can simply type:
`ref`
`dup`
Notice the similarity to normal updates.
ATTENTION: This will take a long time and download several hundred MBs. Do not ever do this on a laptop without plugging in the power cable!

Installing and uninstalling software
To install a specific software you have in mind, for example Firefox, you can run Apper if you want a graphical frontend. But, of course, you are smart and you know that opening up an app store takes up resources for nothing. Therefore, you choose to do the exact same thing in the console: simply type 'sudo zypper install MozillaFirefox'. The package manager will look for any package with the name MozillaFirefox. Then it will tell you how big it is and ask you whether you want to install it, which you can answer by typing 'y' or 'n'. If the package has unmet dependencies, i.e. software that is needed to run the software you want to install, those packages will already be in the list the PM asks you to confirm.
You can even search for packages in the terminal. The corresponding command is 'zypper search your-favorite-package-name'

The same way you install software you can also remove them. For example to remove Firefox type 'sudo zypper remove MozillaFirefox'.
If you are unsure about how a certain software package is called, but you know the first few letters of it, you can use the tab tab method to let the package manager show you all software that starts with these letters.
Caution: Again, do not interrupt the package manager when it is installing or removing software. It may only be interrupted with Ctrl+c while it is downloading.

Adding new Software Sources
The links to the software repositories the System uses are located in '/etc/zypp/repos.d'. People who want to distribute software that is not yet in the official openSUSE repositories can use the build service. If you want to add someone's home or a project repository, you can use "zypper ar -f <URL> <Name>" or YaST->Manager repositories.

Installing software manually

The package manager can only deal with packages. However, a program does not necessarily come in a package - this is only the case if someone packs it up. If there is no package for your program, you have to install it manually.
Note: The package manager does not know about the software you have installed manually. You need to manage and update it yourself. We consider two cases:

Compiling from source

This is the typical way to install open source software without package manager. The developer just provides you the source code, but no executable which you could run. You therefore have to take the developer's role and compile the code yourself.
First, always check on the developer's website if there is any information about the compilation provided. Sometimes they publish instructions for compilation, like: "First, install the following dependencies: ..., then type automake && make". This is the dream case, be thankful to the programmer. Note that you already know that for installing a dependency, 'sudo apt-get' will do the job for you.
If there is no such information provided, welcome to dependency hell. The typical way to install has five steps:
1. Extract the source files into a directory of your choice (e.g. your Downloads folder).
2. Open up a terminal and cd to that folder, then type './configure'. As you have learned above, this command will execute the script called 'configure' in the current folder. That script is provided by the developer and will check if you have all dependencies installed. If not, it will fail and print what dependency could not be resolved. This can be a long and painful process, as configure only prints out one unmet dependency at a time. Also, the name of the missing dependency may not be the name of the package you need to install. As a rule of thumb, if the library xx is missing, you usually need to type "sudo apt-get install libxx". For example, if configure complains that openal could not be found, install the libopenall package. Then run './configure' again and be ready for the next surprise.
3. Once the configuration was successful, there should be no more problems (unless the developer is either incompetent, lazy or sadistic). Simply type 'make' and look at the beauty of the extremely verbose gcc output busting your terminal. Depending on the size of the program you are compiling, this can take a very long time and use quite some CPU power.
4. The program is now ready for running, but it is still in your temporary folder (e.g. Downloads). You would have to cd to that folder and run it with ./programname every time you want to run it. For that reason, you might want to copy it to your system folders. Usually, the makefile contains a section that tells make how to install it. Therefore, smile and type 'sudo make install'. You need sudo because you are now writing to /usr/bin/ or somewhere similiar.
5. This is it! By typing programname, you should be able to run your program now. You can delete your temporary folder now. Or keep it if you want to reuse it (for reinstalling or uninstalling the program automatically for example).

Self-extracting executable

Proprietary, closed-source software is often not distributed as a package, but as it is not open source, it cannot be compiled either. An alternative approach is therefore often chosen by the developers: they provide you an executable that will install your actual program.
This method is like a setup.exe under Windows. Linux people do not like it, because it lets them no control of what will be installed where. Still, if you want a particular software, you are sometimes forced to do it that way. A typical example for this is Matlab.
Such self-extracting executables always come with installation instructions. The typical way to install them is:
1. If the file you downloaded is an archive (.zip, .tar.gz, etc.), you have to extract it first.
2. The install instructions will tell you that you need to run a particular file, let us call it install.sh for example. Make sure that this file is executable. Remember the privileges part from above: the appropriate command for letting you execute the file is: 'chmod a+x install.sh' (you have to cd into the containing folder first of course).
3. Now run the executable: './install.sh'. If the executable requires root rights (e.g. if it wants to install things into /usr/bin/), execute it with sudo: 'sudo ./install'.
Caution: You have no idea what that executable does. Make sure you trust the developer and read the install instructions through before running it.

Maintenance

Linux only requires some maintenance usually. You basically need to do 2 things: Installing updates and backing up your files. Both are critical for your security and have to be done regularly.

Updating your system

This is probably what you will be loving your Linux most for. Any software managed by the package manager will be updated by it automatically every time you perform an upgrade. The chapter about the package manager above explains how to do that. You should install updates at least once a week. A good trick to not forget it is writing yourself a script or alias that will install updates and then poweroff the computer automatically. If you take on the habit to use that command instead of just the poweroff command every time you want to shut down your computer, you will check for updates very often without any additional work. You can even forget about updates, because they have become part of your daily workflow. Note that you cannot check for updates too often on a Linux machine. The package manager will simply go online and check if there is anything to be installed. If you check for updates five times an hour, chances are good that there are none available and nothing will happen.

An example for an bash alias for such a command would be (add it to your .bashrc): 'alias up='sudo zypper ref && sudo zypper –quiet –non-interactive update && sudo poweroff''. Every time you type 'up' (here, meaning update and poweroff), this will check for updates, install them if any and then poweroff the computer.

Note: From time to time, you should still do a manual update, just to make sure everything works fine. The options only answers yes to regular questions. If your package manager encounters a problem that requires your interaction they will cause it to skip the update, but your computer might shutdown too fast to read the error message.

Of course, the package manager cannot update software you have installed manually, as it has no knowledge of the existence of such software. If you have installed a program by hand, you need to recompile it from source (using the typical "./configure – make – sudo make install"-scheme, just like when you installed it).

Backup

Any storage media can fail at any moment. No matter how expensive or new your computer or hard disk is, it can lose your data at any moment. The only way to protect yourself is to do a copy of all the data that is important to you. Typically, the easiest way to achieve this is to use an external USB hard drive that has at least as much space as the sum of the space on the computers you want to backup on it. There are many ways to perform a backup and there really is not a way to tell which one is best for you. A good backup is performed at least every two weeks and includes all files that you would not like to lose. Also, an efficient backup does not contain trash files such as the content of your Downloads folder (which you would probably throw away soon anyway).

using rsync

rsync is a very powerful console tool for copying data. It will only copy the data that has not been changed, allowing you to do super-fast synchronization of your files from the computer to the external drive. The syntax of rsync is of the form 'rsync [options] source destination' where in this case source is your home folder on the computer and target the backup folder on your external hard drive. The most commonly used options of rsync are -avh, where -a stands for "keep the original file privileges and attributes", -v for "give me more output so that I know what's going on in detail" and -h for "express sizes in K/M/G to make it more readable". Further, the --delete option specifies that if a file is missing in the source, it will also be deleted from the target. If you want it to be into an other folder instead, use --backup and --backup-dir="path/to/backup/directory". Note that any file removed from your computer that was backed up on the hard disk will be moved to the backup-dir (which you can put anywhere). That backup-dir only contains the deleted files and will only increase in size, taking more and more space on your disk until you delete it manually.

If you have many large files on your computer, you might want to see the copy progress of each file. For this, use the option --progress. If you like to see statistics when rsync is done (like how many files have been found, transferred, deleted etc.), use the --stats option.

rsync even supports remote sources and targets. It has a very good manpage, so if you are interested we recommend you to read a little bit through 'man rsync'.

using something else

There are people out there who burn their personal backup every time to a blank CD or DVD. We do not understand why anyone would do that and consider it a waste of time and money.

If you only have few data in a single folder, you can as well copy it to a USB

stick. Remember: having a backup just means that your data is at two different, independent places at the same time.
Files stored in your Owncloud or Dropbox folder are automatically synched to a web server. This data is therefore backed up whenever you are connected to the internet. Note that Dropbox could theoretically access and read your files, so be careful!

Repairing a broken system using chroot

Repairing Windows overrides GRUB so that Linux cannot be booted any more. Moving boot partitions or messing up your boot partition will result in failure to boot your system. When playing around with the sudoers file or password, it is possible to lock yourself out of your own computer. All these accidents make your computer unusable, but all of them can be fixed with the chroot method.
Chroot basically lets you become root on a different system than the one that was booted. This means that you can boot from a live CD or USB stick and then login on the system that is on your hard disk without having to boot or know the password of the latter.
On Ubuntu, the process is described in `http://wiki.ubuntuusers.de/chroot/Live-CD`. This is a summary and translation of the indicated website. To understand what is going on, you must have read and understood the chapter "Dealing with files and folders".
For openSUSE, make sure that your installation media matches the system on your hard disk: openSUSE version and 32/64-bit must be the same. Boot your computer using the installation media but select "Rescue system" instead of "Installation". Your computer will boot in text-only mode, you can log in with "root" and no password.
We will assume that your target system is not encrypted. Run 'blkid' to find out on which partition(s) your system is located. Check the partition's size to get a feel for which one it is. It will be of the form /dev/sdxY where x is a letter and Y is a number.
Now, let us mount your system partition.
For MBR boot tables: if you have a separate boot partition, mount it as well: use 'mount /dev/sdzY /mnt/boot'. For EFI boot, you have to mount the EFI partition: 'mount /dev/sdwY /mnt/boot/efi'
We now need to mount five system folders into the right places. This would be five commands, but of course Linux allows us to summarize this: Simply type 'for i in /dev /proc /sys ; do sudo mount -o bind i /mnti; done'. If you read the section about scripting, you may have recognized the bash for-loop in this line.
Finally, if you need an internet connection, you will need to type "ip addr" to find out the name of your network card (starts with enp) and then "dhclient enpX" to obtain an IP-adress. Afterwards, the following is recommended: 'sudo mount -o bind /etc/resolv.conf /mnt/etc/resolv.conf'
We are done mounting and ready to change into the system on your disk using 'chroot /mnt /bin/bash'.
You are now logged in as root on your own system, not the live system any more. To fix the boot with GRUB, run yast and open the bootloader configuration. You do not need to change anything, just press save and the bootloader gets automatically rewritten.
You can now work like if you booted your system normally. This means that you can use nano to fix config files you broke, visudo to get a messed up sudoers file back to work or use passwd (or 'passwd user') to set a new password for a user in case you forgot it.
You cannot use the command poweroff in chroot mode. Instead, disconnect using 'exit' or Ctrl+d, then type poweroff.
Note: Anyone can chroot into your system without having to know your password. This should not be surprising to you, as you already know that anyone can access your files using a live system anyway. The only way to keep your data safe is to encrypt your hard disk.

Where to go now?

Explore! You will never be done learning about Linux, the topic is simply too huge. What you have learned in this course is kind of a survival kit that should get you through the worst in the beginning. However you can go much further, either by experimenting or by reading. The following websites are great places to visit:
- `activedoc.opensuse.org`: Official openSUSE documentation(used to be available as books)
- `community.ubuntu.com`: Official Ubuntu forums
- `archlinux.com`: The ArchWiki is possibly the best Linux wiki in English, much of the information is also useful for openSUSE.
- Wikipedia: Great resource, great explanations