Operating Systems (OS)s

Syntax

- $\alpha|\beta$: either α or β
- $[\alpha]$: α is optonal
- $\{\alpha\}$: α can occout zero or multiple times.
- ... : Further arguments, options, ... are possible.

Definition 0.1 Institute of Electrical and Electronics Engineers (IEEE)**:** Is an organization composed of engineers, scientists, and students that developers uniform standards.

Definition 0.2 Portable Operating System Interface (POISX)**:**
Is a family of standards, specified by the IEEE[def. 0.1], in order to have uniform API's (as well as ancillary issues, such as command-line shell utilities) accords different unix operating systems.

Unix

Unix is a portable, multitasking, multiuser, time-sharing operating system (OS) written in C, that can be considered as the mother of all operating systems.

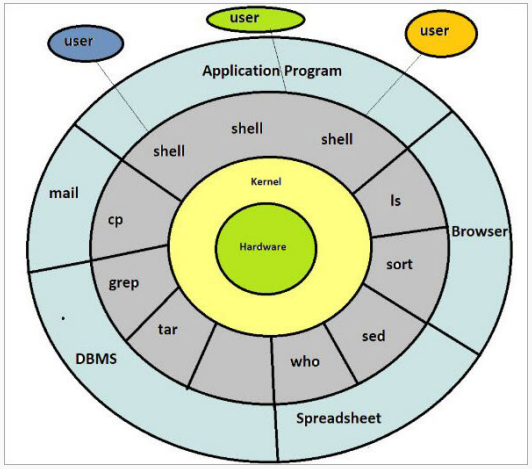


Figure 1: Basic Unix architecture

Categories of Unix Systems

- **Unix**: A family of operating systems. This family includes both UNIX operating systems and Unix-like operating systems.
- **Unix-like operating systems**: These look and operate like Unix, but haven't been certified as compliant.
- **UNIX operating systems**: These have been certified as compliant to the standards i.e. macOS is a UNIX 03-compliant operating system

Note

The UNIX OS is POSIX-compliant **??**.

Unix vs. Microsoft

Unix treats the command line as a first class citizen while the GUI is (even to this day) often capable of only a subset of features. This is reversed in Windows (though improving with PowerShell and Server Nano). The GUI is first class in Windows while the command line is limited.

History taken from [?]

The origins of Unix date back to the mid-1960s when the Massachusetts Institute of Technology, Bell Labs, and General Electric were developing Multics (Multiplexed Information and Computer Services), a time-sharing operating system for the GE-645 mainframe computer. Multics featured several innovations, but also presented severe problems. Frustrated by the size and complexity of Multics, but not by its goals, individual researchers at Bell Labs started withdrawing from the project. The last to leave were Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna, who decided to reimplement their experiences in a new project of smaller scale. This new operating system was initially without organizational backing, and also without a name. The new operating system was a single-tasking system. In 1970, the group coined the name Unics for *Uniplexed Information and Computing Service*, as a pun on Multics. The operating system was originally written in assembly language, but in 1973, Version 4 Unix was rewritten in C.
Once *Unics* could support multiple users it was renamed to *Unix*, where no one knows how the final name came up.

Linux

GNU/Linux is an open source implementation of the Unix and Posix specifications that comes in many distributions and has diverged in many ways from Unix.

History

Richard Stallman was looking to create a truly free and open source alternative to the proprietary Unix system. He was working on the utilities and programs under the name GNU, a recursive acronym meaning "*GNU's not Unix!*". Although there was a kernel project underway, it turned out to be difficult going, and without a kernel, the free and open source operating system dream could not be realized. It was *Linus Torvald's* work—producing a working and viable kernel that he called Linux—that brought the complete operating system to life. Given that Linus was using several GNU tools (e.g., the GNU Compiler Collection, or GCC), the marriage of the GNU tools and the Linux kernel was a perfect match. **Thus** Linux (From **Lin**us and Unix**)** is nothing but a UNIX clone which was written by Linus Torvalds and some other hackers from scratch, in order not to infringe property rights of Unix.

Linux vs Unix

People have come to use the term Unix to mean Unix operating systems other than Linux but Linux can be considered as a unix like operating system.

1. The Standard C Library

Definition 2.1 The Standard C-Library**:** Is a C-library of standard functions that can be used by all C programs and sometimes by programs in other languages.

Definition 2.2 glibc**:** This is the C library that is nowadays used in all major Linux distributions. It is also the C library whose details are documented in the relevant pages of the man-pages project (primarily in Section 3 of the manual).

2. Important Linux/Unix Commands

Description 2.1 (getconf).
Prints the values of POSIX ([def. 0.2]) or X/Open path or system configuration variables to the standard output.
`getconf [options] system_var`

Description 2.2 (chroot).

Working with Files

0.1. Obtaining the file type

```
file (file|achieve)
```

Archives

Definition 3.1 Achieves**:** An Archive or archive file is nothing else then a file that is composed out of multiple other files along with some metadata. Archive files are used to collect multiple data files together into a single file for easier portability and storage, or simply to compress files to use less storage space.
Archive files often store directory structures, error detection and correction information, arbitrary comments, and sometimes use built-in encryption.
Archive files are particularly useful in that they store file system data and metadata within the contents of a particular file, and thus can be stored on systems or sent over channels (i.e. email) that do not support the file system in question.
There exist different archive formats that support different features.

add Beyond archival purposes, archive files are frequently used for packaging software for distribution, as software contents are often naturally spread across several files; the archive is then known as a package. While the archival file format is the same, there are additional conventions about contents, such as requiring a manifest file, and the resulting format is known as a package format.

Classical Archive Files *.a/*.ar/*.lib

Definition 3.2 The Archiver **ar:** The archiver is a Unix utility that maintains groups of files as a single archive file. Today, ar is generally used only to create and update static library `.o,.obj`.
The file format is not standardized and several variants exit.

```
ar [options] libname.a file1.o file2.o ...
```

Attention

In the Linux Standard Base (LSB), ar has been deprecated and is expected to disappear in a future release of that Standard.

ZIP

2.1. Creating ZIP Files

```
zip -9 -r file.zip folder/
```

2.2. Unzipping ZIP Files

```
unzip achieve.zip/
```

3. Tar Files

Definition 3.3 Tar**:**

add

3.1. Unpacking tgz Files

```
tar xvf achieve.tar
```

4. TGZ

Definition 3.4 TGZ**:** A TGZ file is a Unix .TAR archive compressed with Gnu Zip (.GZIP).

4.1. Unpacking tgz Files

```
tar zxvf achieve.tgz
```

File Conversion

5. Pandoc

Definition 3.5 Pandoc**:** Is an all purpose Haskell library for converting from one markup format to another.

```
pandoc [options] file.md
```

`-t`: type to convert to

`-o`: output filename

Booting

Definition 4.1 Booting**:**

Definition 4.2 BIOS (basic input/output system)**:**
Is the very first program (firmware) that is executed once the system is switched on. In most cases it is stored in a flash memory in the motherboard itself and independent of the system storage.
It is used to perform hardware initialization during the booting process (power-on startup), and to provide runtime services for operating systems and programs. **Obtaing information about the current firmware**:

```
efibootmgr -v
```

Note

The BIOS will soon be dead as Intel has announced plans to completely replace it with UEFI on all their chipsets by 2020.

Definition 4.3
UEFI (Unified Extensible Firmware Interface)**:** UEFI or EFI is a new model for the interface between operating system and firmware. UEFI is a more modern solution than BIOS, supporting larger hard drives, faster boot times, more security features, and—conveniently—graphics and mouse cursors.

Definition 4.4 Master Boot Record (MBR)**:**
Is the first 512 bytes of a storage device. It contains an operating system bootloader and the storage device's partition table ([def. 5.8]). It plays an important role in the boot process under legacy BIOS systems.

Definition 4.5 RAM Disk/Drive**:** Is the usage of RAM storage as disk drive.

Definition 4.6 Initial root filesystem/Initial ramdisk**:**
The initial root filesystem is known as the initial ramdisk because the filesystem lives in a disk image[def. 5.5] created by the kernel in RAM. An disk image of the initial root file system (along with the kernel image) must be stored somewhere accessible by the Linux bootloader or the boot firmware of the computer.
This can be:

- the root file system itself
- a boot image on an optical disc or a USB stick
- a small partition on a local disk – called boot partition (usuallly using ext2 or FAT file systems)
- a TFTP server (on systems that can boot from Ethernet).

The bootloader will load the kernel and initial root file system image into memory and then start the kernel, passing in the memory address of the image. At the end of its boot sequence, the kernel tries to determine the format of the image from its first few blocks of data, which can lead either to the initrd or initramfs scheme. initrd (initial ramdisk) is a scheme for loading a temporary root file system into memory, which may be used as part of the Linux startup process. initrd and initramfs refer to two different methods of achieving this. Both are commonly used to make preparations before the real root file system can be mounted

make this better

1. Boot Loader

Definition 4.7 Boot/Bootstrap Loader**:** Is a piece of software started by the firmware (BIOS or UEFI) that is responsible for loading the kernel with the wanted kernel parameters, and initial RAM disk based on configuration files. Thus it is a piece of software executed during start up, that reads the operating system kernel into memory and then passes control to it.

Note

A boot loader must be able to access the kernel and initramfs image(s), otherwise the system will not boot. Thus, in a typical setup, it must support accessing `/boot`.
That means it must have support for everything starting from the block devices, stacked block devices (LVM, RAID, dm-crypt, LUKS, etc), including the file system on which the kernel(s) and initramfs image(s) reside.

Which Bootloader am I using
`efibootmgr -v`
under `Linux Boot Manager` we can see which boot manger we are using.

1.1. systemd-boot

Definition 4.8 systemd-boot/gummiboot**:** Is a simple UEFI boot manager which executes configured EFI images. It is included with systemd, which is installed on an Arch system by default.

Notes
- Was previously called gummiboot.
- It is simple to configure but it can only start EFI executables such as the Linux kernel EFISTUB, UEFI Shell, GRUB, or the Windows Boot Manager.

1.1.1. Bootctl

Definition 4.1 bootclt**:** Can check the EFI boot loader status, list available boot loaders and boot loader entries, and install, update, or remove the systemd-boot(7) boot loader on the current system.

2. Boot Manager

Definition 4.9 Boot Manager**:** Is a program that lets the user choose the operating systems if multiple are available. After selecting an operating system the boot manager loads and executes the boot loader of that operating system.

Mass Storage Management

Definition 5.1 Secondary Storage**:** differs from primary storage in that it is not directly accessible by the CPU. The computer usually uses its input/output channels to access secondary storage and transfer the desired data to primary storage. Secondary storage is non-volatile (retaining data when power is shut off).

Definition 5.2 Character Device/Special File**:** provide unbuffered, direct access to the hardware device. They do not necessarily allow programs to read or write single characters at a time; that is up to the device in question. Word-sizes are usually 1 to 8 Bytes.

Definition 5.3 Block Device/Special File**:** a block device is a kind of file which represents a device of some kind, with data that can be read or written to it in blocks of fixed size i.e. 512 bytes or a multiple of that.
They provide buffered access to hardware devices s.t. we can fill the buffer with arbitrary word-sizes.
This provides some abstraction from their specifics.
- In practice, a block device usually behaves just like a regular file
- They can be created anywhere in the filesystem `mknod`
- Regular reside in `/dev`
- **Examples**: Hard Drives, Partitions, DVDs, SSDs, RAM

We can use `lsblk` in order to print all available block devices (except RAM disks) in a tree-like format:

```
lsblk
```

Corollary 5.1 Physical Block Devices: Is a physically attached block device such as a HDs represented by a block device.

Corollary 5.2 Virtual Block Devices:
add

Note
- The downside is that because block devices are buffered by the file system, the programmer does not know how long it will take before written data is passed from the kernel's buffers to the actual device, or indeed in what order two separate writes will arrive at the physical device.
- If we read directly from this block devices then we read the data directly i.e. from the hard disk; completely bypassing the file system code in the kernel, possibly not up-to date due to caching of the file-system.
  Attention: when writing to it we will possibly corrupt the file system, as it does not know about it.

Definition 5.4 Image**:** Is a compressed, self-contained piece of software that has to be unwrapped in order to be used.

Definition 5.5 Disk Image**:** Is a copy of the entire contents of a storage device, such as a hard drive, DVD, or CD. The disk image represents the content exactly as it is on the original storage device, including both data and structure information.

1. Partitions

Definition 5.6 Partition**:** Block devices can be divided into one or more regions/logical disks called partitions.
This allows the OS to manage information in each partition separately.

Definition 5.7 Disk Partition**:** disk partitioning or disk slicing is the creation of one or more regions/disk partitions on a hard disk or other secondary storage, so that an operating system can manage information in each region separately. They usually reside under `/dev/`

Definition 5.8 Partition Table**:** A partition table is a 64-byte data structure that provides basic information for a computer's operating system about the division of the hard disk drive (HDD) into primary partitions. This division is recorded in the partition table, usually found in sector 0 of the disk.

1.1. MBR Partition Table

Definition 5.9 MBR Parition Table (see [def. 4.4])**:** Is the partition table used by BIOS firmware systems in order to save information about the hard drive data. MBR uses 32-bit entries in its table which limits the total physical partitions to 2TB ($2^{32}512B$).

1.2. GUID Partition Table

Definition 5.10
Universal/Global unique identifier (UUID)/(GUID)**:**
Is a 128-bit number used to identify information in computer systems. When generated according to the standard methods, UUIDs are for practical purposes unique. This is due to the fact that the likelihood of a duplicated UUID is very small, A sample of $3.26 * 10^{16}$ UUIDs has a 99.99% chance of not having any duplicates. Thus, anyone can create a UUID and use it to identify something with near certainty.
Format: `123e4567-e89b-12d3-a456-426655440000`
GPT uses 64-bit entries in its table which dramatically extends the support for size possibilities of the hard drive.

Definition 5.11 GUID Partition Table (GPT)**:** Is the partition table used by UEFI firmware systems in order to save information about the hard drive data. It is a standard for the layout of partition tables, that relies on UUIDs ([def. 5.10]).

1.3. fdisk

Definition 5.1 fdisk**:** Is a program in order to manipulate the disk partition table. It supports GPT, MBR, Sun, SGI and BSD partition tables.
To open a disk:

```
fdisk /dev/disk
```

Note
fdisk supports GPT since util-linux 2.23.

1.4. Operations on Disk

`p`: print the partition table

`F`: List Free unpartitioned space

`l`: list known partition types

1.4.1. Deleting Partitions `p`

Open dialogue in order to delete disks using `d`
1. Select the partion(s): `3` or `1,2` or `6-2`
2. Reload to partition table in order to verify what you did `p`
3. Save your changes: `w`

1.4.2. Creating New Partitions `n`

1. Enter Partition Number:
2. Enter First Sector: (can be seen from partition table)
3. Enter Last Sector:
   - + symbol to specify a position relative to the start sector/size measured in:
     - sectors
     - size kibibytes (K), mebibytes (M), gibibytes (G), tebibytes (T), or pebibytes (P)

The Kernel

Obtaining Machine Information

2.1. CPU Informations

- `lscpu`: outputs important information about the machine and CPU.
  or alternatively
- `cat /proc/cpuinfo`

2.2. Cache Information

- `grep . /sys/devices/system/cpu/cpu0/cache/index*/*`
  or alternatively using getconf
- `getconf -a | grep CACHE`
- `lscpu --caches`
- `lstopo`

Attention: `getconf -a | grep CACHE`: returns `ALL-SIZE` cache which is the cache of the sum of all processors and not of a single one.

2.3. Memory Information

- `cat /proc/meminfo`

2.4. Obtaining OS-Information

Obtaining the OS-Version
`uname -a [further_options]`

3. The Device Mapper

Definition 5.12 Device Mapper**:** is a framework provided by the Linux kernel for mapping physical block devices onto higher-level virtual block devices.
It forms the foundation of:
- the logical volume manager (LVM) [def. 5.13]
- software RAIDs
- dm-crypt disk encryption
- additional features such as file system snapshots

4. Logical Volume Management (LVM)

4.1. Logical Volume Manager (LVM)

Definition 5.13 Logical Volume Manage (LVM)**:** Is a logical volume manager for the Linux kernel; it manages disk drives and similar mass-storage devices for virtual volumes by using the kernels device mapper [def. 5.12].

4.2. Physical Volumes (PVs)

Definition 5.14 Physical Volumes (PVs)**:**
Is the actual physical storage of logical volumes.
It is a mass storage device or a partition ([def. 5.7]) represented by a block device ([def. 5.3]).
- `sudo pvs`: display in one line information about the pv's
- `sudo pvdisplay`: display nicely in multi-line output
- `sudo pvscan`: scans all supported LVM block devices in the system for physical volumes.

Definition 5.15 Physical Extent (PE)**:** A physical volume consists of many small pieces called physical extents.
It is thus the the smallest allocatable size of the physical volumes and hence also the smallest allocatable size in a LV. We may think of it as sector for HDs or partitions.
It is not possible to change the size of the PE's after the creation of a PV.
The standard size defaults to 4 MBytes.

4.2.1. Creating Physical Volumes

Creating a partion

```
pvcreate /dev/diskX
```

Where `X` is the lvm partition that we created using definition 5.1.

4.3. Volume Groups (VGs)

Definition 5.16 Volume Groups (VGs)**:** Is a container for one or multiple physical volumes (PVs), from which we may create logical volumes (LVs).
- `sudo vgs`: display in one line information about the pv's
- `sudo vgdisplay`: display nicely in multi-line output
- `sudo vgscan`: scans all the disks for volume groups, displays them and rebuilds the LVM cache file.

4.3.1. Creating Volume Groups

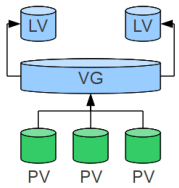
Creating a vg

```
vgcreate volum_grou_name /dev/diskX
```

Where `X` is the lvm partition that we created using definition 5.1.

4.4. Logical Volumes (LVs)

Definition 5.17 Logical Volume (LVs)**:**
Logical volumes or simply volumes are a kind of partition within a volume group (VG) and can thus be treated as such.



This means we:
- may create file systems on them
- may mount/unmount them

There exist three kinds of LVs:
1. `Linear Volumes`: are plain normal partitions
2. `Stripped Volumes`–RAID0: data is split or stripped evenly between two disks ⇒ faster reads and writes
3. `Mirrored Volumes`–RAID1: data is mirrored between two disks ⇒ safety against data loss

Displaying information about LV's:
- `sudo lvs`: display in one line information about the LV's
- `sudo lvdisplay`: display nicely in multi-line output
- `sudo lvscan`: scans all the disks for LV's and displays there status, location and size
- `df -h`: find out available and used space

Note
- It is not possible to create logical volumes without volume groups.
- RADI0/1 requires two physical disks

List divices capable of being used as physical volumes
`lvmdiskscan`

Data Loss!: Make sure to target the correct partition, otherwise we may end up with data loss.

4.4.1. Creating Logical Volumes

Creating LV's

```
lvcreate options volume_name
```

Hint: use

In order to use the rest of the free volume group we can use: `-l 100%FREE`

`(-L|-size) size[unit]`: Specify the size of the volume to be created

`(-n|-name) string`: Specify the name of the volume we are creating

`(-l|-extents) number[%]`:

4.4.2. Renaming LV's

Renaming LVs

```
lvrename path_to_lvold path_to_lvnew
lvrename volume_group_name lvold_name lvnew_name
```

4.4.3. Shrinking LV's

Attention:

- While enlarging a file system can often be done on-line (i.e. while it is mounted), even for the root partition, shrinking will nearly always require to first unmount the file system so as to prevent data loss.
- Make sure your FS supports what you are trying to do.

Warning: `lvresize` resizes an LVM volume, but it doesn't care at all what lies within it.
Thus if we simply reduce a LV without resizing its file system first, we will trash the file system. The file system can be resized first by using `resize2fs` or by supplying the `-resizefs` flag to `lvresize` which in turn will use `fsadm` to resize the fs adequately.

Resizing LVs and file systems in one go

Note: `full_path_to_lv` corresponds to `/dev/vg_name/lv_name`.

1. Start arch from a terminal only and change into the root directory:
   ```
   cd # Note all further commands are executed from /
   ```
2. Unmount the file system we want to reduce:
   ```
   umount path_to_fs # i.e. /home
   ```
3. Check the integrity of the fs:
   ```
   fsck -f full_path_to_lv
   ```
4. resize the LV `lv` by `X` GB and reduce the file system correspondingly. `lv` may either be:
   - `full_path_to_lv`
   - `vg_name/lv_name`

   ```
   lvresize *L XG --resizefs lv
   ```
   ```
   mount path_to_fs # i.e. /home
   ```

`-L *XT` : Change size of LV by `X` units of type `T` by:

- empty: change size to `XT`
- `+`: increase the size by `XT`
- `-`: decreasing the size by `XT`

Note

In order to extend the root volume group you need to specify the path:

```
lvresize +L XG --resizefs /dev/vg_name/lv_roo_name
```

Scalability

In a lvm all elements PV's, VG's, LV's can be resized apart from PE's.

The File System

Unix-like operating systems create a virtual file system, which makes all the files on all the devices appear to exist in a single hierarchy

Definition 5.18 The Root File System /:
The root filesystem, represented by the slash symbol by itself (/), is the primary filesystem (top in the hierarchy) under which all other files/filesystems live.

Information about the file system

- `df -hT`
- `lsblk -f`

5. Mounting

Unix-like systems assign a device name to each device for example `/dev/sdaX`.
However this is not how the files on that device are accessed. Instead, to gain access to files on another device we make those them available in the directory tree by mounting ([def. 5.19]) the device somewhere in the directory tree.

Definition 5.19 Mounting:
Is the process of attaching a file system to a directory (mount point) and make it available to the system.
The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.

```
mount options server mountpoint/
```

Definition 5.20 Mount Point: Is the directory onto which we mount a given file system.

5.1. The Filesystem Table /etc/fstab

In many situations, file systems other than the root need to be available as soon as the operating system has booted.
Hence all Unix-like systems therefore provide facilities for mounting file systems automatically at boot time.

Definition 5.21 File System Table (fstab): Is the file `etc/fstab`, that can be used to define how disk partitions, block devices, or remote filesystems should be mounted into the filesystem automatically.
Each line describes a file system Each filesystem is described in a separate line:

```
# <device> <dir> <type> <options> <dump> <fsck>
```

At boot time these definitions will then be dynamically converted into systemd.mount arguments (for each file system in the file) and then mounted by systemd.

`device`: describes the block special device or remote filesystem to be mounted by UUID [def. 5.10]

`dir`: the mount directory

`type`: type of the filesystem

`options`: associated mount options

`dump`: If the filesystem should be checked by the `dump` utility. This field is usually set to 0, which disables the check.

`fsck`: sets the order for filesystem checks at boot time using `fsck`. For the root device it should be 1. For other partitions it should be 2, or 0 to disable checking.

5.1.1. genfstab

Definition 5.22 Generate fstab genfstab:
Is a script to automatically generate the `/etc/fstab` file during arch linux installation

```
genfstab [options] root
```

`-U`: Use UUIDs for source identifiers

Example 5.1 :

```
genfstab -U /mnt >> /mnt/etc/fstab
```

6. Other Filesystems

6.1. The Common Internet File System (CIFS)

Definition 5.23 The Common Internet File System (CIFS):
is a network file-sharing protocol form of SMB??

Arch

Basics

Definition 5.24 deamon: is a program that runs continuously in the background and exists for the purpose of handling periodic service requests that a computer system expects to receive. The deamon (program) forwards the requests to (real) program/process as appropriate.

8. Systemd

Definition 5.25 systemd: systemd is a suite of basic building blocks for a Linux system.
It provides a system and service manager that runs as PID 1 and starts the rest of the system. It provides the ability to enable, start, stop and disable services.

Note

The name systemd stems from the Unix convention of naming daemons by appending the letter d.

8.1. Systemd/User

Definition 5.26 systemd/user: offers *users* the ability to manage services under the users control with a per-user systemd instance, enabling users to manage their own services. This is convenient for daemons[def. 5.24] and other services that are commonly run for a single user s.a. ssh-agents.
User units are located in the following directories (ordered by ascending precedence):

- `/usr/lib/systemd/user/`: units provided by installed packages
- `~/.local/share/systemd/user/`: units of packages that have been installed in the home directory
- `/etc/systemd/user/`: system-wide user units placed by the system administrator
- `~/.config/systemd/user/`: where users put their own units

add more details

8.1.1. pam enviroment

Definition 5.27 pam_enviroment: Environment variables can be made available through use of the `pam_env.so` module. Where user defined environment variables belong into: `~/.pam_enviroment`

add somewhere PAM and more details on ~/.pam_environment

Package Management

Definition 5.28 The Official repositories: Contains essential and popular software, readily accessible via pacman

Definition 5.29 Patch: Patch is a Unix program that updates text files according to instructions contained in a separate file, called a *patch file*.
The patch file (also called a patch for short) is a text file that consists of a list of differences and is produced by running the related diff program with the original and updated file as arguments. Updating files with patch is often referred to as applying the patch or simply patching the files.

Definition 5.30 Ports collections/ports trees/ports:
Ports are package-management systems within the unix-derivate world.
A port collections refers then to the collection of makefiles and patches in order to create binary packages.

pkgname.pkg.tar.xz

8.2. Arch Build System (ABS)

9.1. Pacman

Pacman is a simple package manager that consists of:

- binary package format
- an easy-to-use build system

The goal of Pacman is to make it possible to easily manage packages and to keep track of all dependencies.
Pacman keeps the system up to date by synchronizing the clients local package lists with a mirror server.

Removing Packages `-R`
Remove package and leave all its dependencies installed

```
pacman -R package_name
```

Options

- `-Rs`: Remove package and all dependencies not required by any other package.
- `-Rs`:

Query Packages -Q

List All Installed Packages

```
pacman -Qqe
```

List All Foreign (Aurman) Installed Packages

```
pacman -Qm
```

9.1.1. Cache

Definition 5.31 Paccache: Pacman stores its downloaded packages in `/var/cache/pacman/pkg/`, including old or uninstalled versions. The old packages (excluding the k recent) ones can be deleted using:

```
sudo paccache -rk nb
```

Note

The size of the package cache packages can be queries using:

```
sudo ls /var/cache/pacman/pkg/ | wc -l
```

as well as the number of packages:

```
du -sh /var/cache/pacman/pkg/
```

9.2. Arch User Repository (AUR)

Linux Applications

Screen Capture

1.1. Screenshots with GRIM

```
grim screenshot.png
```

`-o`: specify output device.

`-g`: specify region e.g. as "10,20 300x400"

1.1.1. Slurp: Select a Region By Mouse

```
grim -g "$(slurp)" screenshot.png
```

1.2. Screen recordings with wf-recorder

```
wf-recorder options
```

`--file=filepath`: specify audio input device

`--audio=input_device`: specify audio input device

Note

The avialable audio devices can be listed with:

```
pacmd list-sources | grep "name:"
```

Portable Document Format (PDF)

Searching within PDFs
pdfjam – Combining PDFs

Definition 6.1 pdfjam: Is a pacman package and a linux shell script for manipulating PDF files.

2.0.1. Joining PDFs

`pdfjoin` The pdfjam package provides a command in order to join multiple files into a single PDF file:

```
pdfjoin files [options]
```

`files` can be:

- A regex expression s.a. `*.pdf` or
- Multiple files `1.pdf 2.pdf 3.pdf`

`-outfile/-o`: specify output filename

`-rotateoversize`: Can be set to true or false in order to allow/avoid rotations

X Dekstop Group (XDG)

Definition 6.1 XDG: Is also know as and develops a set of standards common across different desktop environments.

3. Desktop Entries

Definition 6.2 prorgam.desktop**:** standard for applications to integrate into application menus of desktop environments. This files are usually stored in:

- System wide programs:
  - /usr/share/applications/
  - /usr/local/share/applications/
- User-specific programs: ~/.local/share/applications/

4. Qt Software

Definition 6.3 Qt Software: Qt is a widget toolkit for creating graphical user interfaces.

4.1. Enviroment Variable Embedded Linux Systems

QT_QPA_PLATFORM: On Embedded Linux systems, there are multiple platform plugins that you can use:

- xcb: the X11 plugin used on regular desktop Linux platforms.

Secure Shell Protocol (SSH)

5. Setting up SSH

5.1. Creating SSH Keys

Definition 6.2 ssh-keygen: can be used in order to create a private and public ssh-keypair:

```
ssh-keygen options
```

where

- this command should usually be run under ~/.ssh
- the public key has the extension .pub

Note

If ~/.ssh does not exist we need to create it mkdir ~/.ssh

-t rsa: specify the cryptographic algorithm to be used

-C: provide an additional comment i.e. an email address

5.2. Using SSH Keys

Definition 6.3 ssh-copy-id: In order to use our ssh key we need to copy the **public** key into .ssh/authorized_keys on the remote server (create it if it does not exist) we can do this using:

```
ssh-copy-id -i ~/.ssh/keyname.pub username@host
```

-i: specify the identity file to copy to the server

Note

If this does not work for some reason we can also do this manually: cat ~/.ssh/keyname.pub

Note: https vs. ssh

If we using a git remote with https this is not enough. We have to change the remote to the ssh link in order to make use of it. This can be done using git remote set-url and the ssh url.

5.3. SSH Config

Definition 6.4 config: If we want to uses multiple ssh keys, use aliases for the server and more we can create a .ssh/config file to specify which server should be accessed how and should use which ssh key:

```
Host aliases
    HostName hostname
    User username
    IdentityFile .ssh/keyname
```

aliases**:** Specifies aliases/shortcuts that we can use with ssh in order to log into the server

HostName**:** Specifies the real hostname Either as name or IP address.

AddKeysToAgent no**:** Can be set in order to add ssh keys automatically to a ssh-agentdefinition 6.6 (if one is running)

ForwardAgent no**:** Can be used in order to automatically forward ssh keys see alsodefinition 6.5

5.3.1. SSH Forwarding

Definition 6.5 SSH key Forwarding: Allows us to use our local ssh keys on another server and thus adds a layer of security as we do not need to store them directly on that server:

```
ssh -A options -A server
```

add ssh-agent in order not to retype k

5.4. SSH Agent

Definition 6.6 ssh-agent: is a key-manager for SSH, that can hold keys and certificates unencrypted and ready to be used in memory. The reason why the ssh-agent is secure is that it neither:

- writes any key to disk
- nor does it allow exporting of our private keys

In other words private keys are only usable for signing. The ssh-agent does this by running in the background (separately from ssh).

```
eval $(ssh-agent options)
```

Why eval?

The command ssh-agent outputs environment variables needed to connect. As ssh-agent can only change modify its own (or its child) environment variables but not its parent process environments we start it using eval.

Definition 6.7 ssh-add: Is the command that we can use in order to add/remove/list/...ssh-keys of a running ssh-agentdefinition 6.6. In order to add a key to the ssh-agent we can simply run:

```
ssh-add [options] [.ssh/keyname]
```

-l: List all added private keys

-d: delete the specified key .ssh/keyname

-X: Lock the ssh-agent

-t life_seoconds: set the maximum life time of a key:

```
ssh-add -t x(s|m|h|d|w) [.ssh/keyname]
```

Note

Starting the ssh-agent can be automated as well as adding keys see also section 3

File Transfer using SSH

6.1. Secure copy protocol (SCP)

In order to copy files from/to the server we may use scp:

```
scp [optional] source destination
```

-r: Allows to copy entire directories

bashExample scp -r source/folder username@host:destination

Note: graphical user interfaces

we may use graphical interfaces s.a. MacFUSE, Macfusion, Cyberduck, Filezilla.

6.3. Remote Sync (rsync)

Is more efficient than scp as it only transfers files that have change and uses an optimized data transfer algorithm:

```
rsyc [optional] source destination
```

-r, -recursive: sync files and directories recursively

-z, -compress: compress file data during the transfer

-a, -archive: achieve files and directories while synchronizing (-a equal to following options -rlptgoD)

-v, -verbose: verbose output.

-q, -quiet: suppress message output.

-n, -dry-run: perform a trial run without synchronization

-h, -human-readable: display the output numbers in a human-readable format

-p, -progress: show sync progress during transfer

Good standard combination Listening 6.1

```
rsync -avrzp source destination
```

Note: use of / at the end of paths

- source:
  - /: rsync will copy the content of the last folder.
  - no slash: rsync will copy the last folder and the content of the folder.
- destination:
  - /: rsync will paste the data inside the last folder.
  - no slash: rsync will create a folder with the last destination folder name and paste the data inside that folder.

6.4. Filesystem in Userspace (FUSE)

- **Mounting the file system:**

```
sshfs pollakg@euler.ethz.ch:[foldertomount] mountpoint
```

  - foldertomount: e.g. /cluster/ or /cluster/home/pollakg/.
  - mountpoint: can be an arbitrary non-existing folder: e.g. ./euler.
- **Unmounting the file system**: umount mountpoint

Content Blockers

7. Dans Guradian e2guardian[AUR]

DansGuardian is no longer maintained but an actively developed fork is e2guardian which shares similar functionality.

GNU Make

Definition 7.1 Make: Is a build automation tool that simplifies and automates building executable and libraries. It builds executables from source code by using user-defined *makefiles*/*Makefiles* that specify how the program should be built.

Make and bash

Make is compatible with bash commands.

Definition 7.1 Make Rule/Target:
Make consists of rules/targets that specify:

1. **dependencies/prerequisite**: are *other targets* that our target depends on.
2. **system_commands/recipes**: specify what actually has to be done (after all dependencies have been satisfied).

```
target: [dependency1 dependency2]
  command1 command2
```

Note

The order of dependencies and system commands of a target are not guaranteed and may even be run in parallel.

1.1. Executing Makefiles

```
make [options]
```

`target:` Specifies which target to run

Note

If no target is specified, then the first target in the *makefile* will be run, except for targets starting with a dot see **??**.

`-f file`:
Specifies which *makefile* to use if multiple makefile exist

`variable[:/?/+]=value`:
Assign values to variables on the command line (see also section 2)

`-B/-always-make`:
Force remake i.e. consider all targets out of date, regardless of the status of their prerequisites.

1.2. How does make decide what to rebuild?

Definition 7.2 Normal Prerequisites: All dependencies that are younger/newer/have been modified after our target are considered out-of-date and will be rebuild.

Note

We may use `ls -t` in order to list files by modification timestamp (newest first).

2. Macros

Definition 7.2 Macros/Variables: Although it is easy to think of makefile variables as traditional programming language variables, there is a distinction between a macro "variable" and a "traditional" variable. A macro variable is expanded "in place" to yield a text string that may then be expanded further.

Definition:

```
variable = value [value2 ...]
```

Usage:

```
$(variable)
```

`:=`: Variables are immediately expanded when they are defined and not when they are used.

`+=`: Values are appended to variables that already have a value.

`?=`: Only assigns a value to the variable if it does not already have a value.

Variable Name Conventions

- `CC/CXX`: C/C++ compiler
- `CFLAGSCXXFLAGS`: C/C++ compiler flags/options
- `LDLFLAGS`: linker flags/options
- `EXEC`: executable name
- `SOURCES`: source files `.c/.cpp`
- `OBJECSTS`: object files `*.o`
- `INCLUDE_PATHS`: include paths
- `LIBS`: libary directories

2.1. Special Macros

- `$@`: Name of current `target:`
- `$^`: Name of all the `dependencies` separated by spaces.
- `$?`: Names of all `dependencies` that are newer/younger than the target separated by spaces.
- `$<`: Name of the first dependency `dependency1`.
- `$|`: List all order only prerequisits[def. 7.5].

2.2. Implicit/Built-in Variables

Definition 7.3 Implicit Macros: Make contains a database of build in variables for convenience. The list of available variables can be displayed (in a directory without makefile) using:

```
make -p
```

Property 7.1 `${RM}`: translates to `rm -r`

2.3. Make Wildcards

Definition 7.3 Makefile Wildcards `%`:
Is a special make wildcard that allows to write general targets that match many different files.
Specifying the wildcard target:

```
[regex]%[regex]
```

Calling the matched wildcard in a rule:

```
$*
```

see also examples 7.1 and 7.2.

Definition 7.4 Wildcard Matching: allows us to match wildcards in the targets as well as in the dependencies:

```
[regex]%[regex]: [regex]%[regex]
```

see **??**

2.4. Special Directives

Override: Lets us ignore assignments in the makefile if they are already set as command line arguments:

```
override variable = value
```

2.5. Phony Targets `.PHONY:`

Definition 7.4 Phony Targets: Is a `target:` that is not related to a real file but is just a recipe to be executed. Usually make executes targets that do not create a `target:` file all the time.
However if a file called `target:` will ever be created, make will no longer run the corresponding recipe (if it has no prerequisites). The solution to this a special target called `.PHONY:`. By making our `target:` a prerequisite of `.PHONY:` it will always be build:

```
.PHONY: clean
clean:
  rm *.o temp
```

3. Order-Only Prerequisites

Definition 7.5 Order-Only Prerequisite: If we want to have certain special dependencies, that are build before others we can use *order-only prerequisites* – like a bootstrap.
Order-only prerequisites are build before other normal prerequisites and they the do not force the target to be rebuild if they are out of date:

```
target: dependencies | order_only_dependencies
```

see example **??**

4. Includes

Definition 7.6 Includes: allows us to include other files into our makefile:

```
[-]include filename
```

`-`: do not abort with an error if the file does not exist.

Include files in make are often called `config.mk`

5. C++ Header File Dependencies

C++ header files have to be stated explicitly, even tough the compiler can deduce them by itself. The reason for his is that otherwise make would not rebuild the `target` if only the header file changed:

```
file.o: file.cpp header.hpp
    command
```

add multiple makefiles in directories

6. Examples

Example 7.1 Make Anything:

```
EXEC=main
SOURCES=$(*.cpp)
OBJECTS=$(SOURCES:%.cpp=%.o)
```

Example 7.2 Make Anything:

```
% :
  gcc -o $* $*.cpp
```

and call it using `make filename`

Example 7.3 Order Only Dependencies: Directory timestamps change even if a file in it gets changed ⇒ without order-only prerequisites the target would always be out of date if the directory changes:

```
targer: dependenies | dir
   command
dir:
   mkdir $@
```

Virtual Private Network (VPN)

1. VPN Cisco (VPNC)

Definition 8.1 **VPN CISCO (VPNC):** Is a VPN client for Cisco hardware VPNs.
For each vpn you create a `.conf` config file inside `/etc/vpnc`.
In order to start a vpn connection you run:

```
sudo vpnc name
```

In order to disconnect you run:

```
sudo vpnc-disconnect
```

Docker

0.1. Virtual Machines (VM)

Definition 10.1 Virtual Machine (VM): A virtual machine is an virtual emulation of a physical computer system. It thus allows us to run different (virtual) operating systems on a given physical operating system and hence to run software that could not run on the given physical operating system otherwise.

Explanation 10.1 (VM). *In order for a virtual machine to run it needs*
- *its own virtualized hardware resources*
- *an underlying (Guest) OS on which it runs*

Definition 10.2 Guest Operating System: Is the operating system that runs on the VM.

Definition 10.3 Hypervisor/Virtual Machine Monitor (VMM): Is a software firmware or hardware that creates and runs VMs by virtualizing the underlying hardware/server.

Drawback:
- Uses a lot of RAM
- Uses a lot of CPU

Advantage:
- All OS resource available to OS
- Established security tools
- Better known security controls

Note: why are VMs so resource heavy

Each VM needs to run a virtual copy of all the hardware that the guest operating systems needs to run, as it shares the virtualized resources with the original operating system. Nevertheless, its still cheaper then buying additional computers.

1. Docker

Definition 10.1 Container: Are only virtualizing the underlying operating system and run as process instead of virtualizing also the hardware.
This makes them much more lightweight in comparison to VMs.

Explanation 10.2 (Container). *Containers are running on top of a physical server/deamon*[def. 5.24] *and its host OS. Each container shares:*
- *the host OS kernel and usually also*
- *the binaries*
- *libraries*

Drawback:
- GUIs don't work well

Advantage:
- Sharing the OS resources without the need to repeat them
- Are super lightweight (only some MB)

Definition 10.2 Docker: Is a way of packaging up an application and all of its dependencies, including those at the operating system level s.a. operating system libraries, into what is known as a container – thus, it acts as a lightweight kind of virtual machine.
However, they do not run as a full virtual machine but as a normal process in the background in a "chroot" fashion.

```
docker info
```

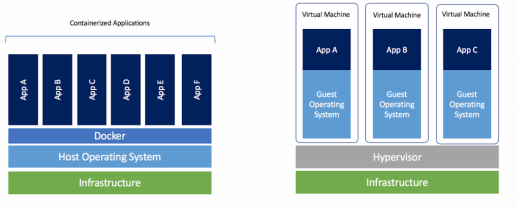


Figure 2: Docker vs VMs

Installation

Docker runs in the background as daemon and thus must be started/enabled by using systemctl.

1.1. Docker Images

Definition 10.4 Docker Images: Is a immutable executable package that includes instructions, as well as executable code, the OS libraries, dependencies, and tools to run an application (see also[def. 5.4]).

```
docker run [options] image_name [command]
```

this will load the image into a container[def. 10.6].

Notes
- Docker Images are is highly dependent on the OS kernel.
- `docker run` will download a matching image from *docker hub*

1.1.1. Options

`command`: Can be used in order to overwrite `CMD` of `Dockerfiles` see **??**.

`-d image_id`: run the image with the given image id

`-it`: run container interactively

Definition 10.5 Base Images: Docker images may be based of other docker images i.e. an Apache docker image may be based of a Ubuntu docker image. Thus docker images may be comprised of multiple layers, where each layer depends on the layer below. The first docker images (which is usually an OS) is called the *base image*.

1.1.2. Commands

Listing all images on the system:

```
docker images
```

Get image details:

```
docker inspect image_name
```

Removing an image:

```
docker rmi image_id
```

1.2. Docker Containers

Definition 10.6 Docker Container: Is an instance of an image that runs the images. The containers currently running on the system can be listed using:

```
docker [options] ps
```

the can be stopped using:

```
docker stop docker_id
```

Note

All running containers can be stopped using:

```
docker stop $(docker ps -q)
```

1.2.1. Options

`-a`: shows all containers (also those not currently running).

`-q`: only display numeric ids (quiet)

1.2.2. Command

`docker stats`: prints the current stats of all the containers running

1.3. Docker Files

`Dockerfile`

Definition 10.7 Dockerfile: Is a simple text file that server as recipe in order to create docker images[def. 10.4]. A docker image can be create from a docker file by running:

```
docker build [options] [dir]
```

where `dir` is the directory where the image resides

1.3.1. Options

`-t image_name:tag_name`: lets us give the image a name and a tag in order to identify it more easily

1.4. Commands

`FROM`: tells the docker from which base image[def. 10.5] to create the image from.

`MAINTAINER`: is the maintainer of the package

`RUN`: Is used to specify which actions to perform while building a docker image.

`CMD`: Is the command used to execute when the built docker image is launched and should be the last command of a docker file:

```
CMD command [param1 param2 ...]
```

Security

1. pwgen

Definition 11.1 pwgen: Is a command line password generator

```
pass options length
```

-c/--capitalize: include at least one capital letter in the password

-n/--numerals: include at least one number in the password

-y/--symbols: Include at least one special symbol in the password

-r <chars>/--remove-chars=<chars>: Remove characters from the set of characters to generate passwords

-s/--secure: Generate completely random passwords

2. GNU Privacy Guard

Definition 11.2
Open Pretty Good Privacy (OpenPGP): OpenPGP is a key-based encryption method used to encrypt files so that only their intended recipient can receive and decrypt them. OpenPGP is used widely to secure e-mail communications, but its technology can also be applied to FTP.
OpenPGP works by using two cryptographic keys to secure files. A Public Key is used to encrypt the file so that only its corresponding Private Key can decrypt it.

Note

Unlike SSL and SSH, OpenPGP is not a type of connection, but a method of encrypting a file prior to uploading it. As such, OpenPGP Mode can be used in conjunction with standard FTP, SSL or SSH connections.

Definition 11.3 GNU Privacy Guard (gpg): GnuPG is a set of programs for public key encryption and digital signatures that implements the OpenPGP standard[def. 11.2]

```
gpg --full-generate-key
```

Definition 11.4 Secret And Public Keys: For cryptography we need a public key (for encryption) and a secrete key (for decryption).

Definition 11.5 Master/Primary-keys: The primary OpenPGP key-pair created by GnuPG is used for signing in order to verify the user-ID.

Definition 11.6 Sub-keys: The OpenPGP sub-key-pair created by GnuPG is used for encryption/decryption and is signed by the primary key[def. 11.5] in order to verify the user identity.

2.1. Listing Keys

Listing Public Keys

```
gpg (-k/--list-keys) options
gpg --list-keys
```

Sample Output

```
pub   encryptionAlg.encryptionBits date_of_creation [usage]
      key-id
uid   [time_of_validity]   owner (comment) <owner mail>
sub   encryptionAlg.encryptionBits date_of_creation [usage]
```

Listing Secrete Keys

```
gpg (-K/-list-secret-keys) options
```

Usage

- pub – **pub**lic primary key
- sec – **sec**ret primary key
- sub – public **sub**-key
- ssb – secret **sub**-key

--with-keygrip: list the name of the secrete key file under `/.gnupg/private-keys-v1.d`

Key Types/Usage

Character	Meaning
S	Signing
C	Certification
E	Encrypting
A	Authentication

2.2. Delete Keys

```
gpg --delete-keys key-id
```

```
gpg --delete-secrete-keys key-id
```

Note

When deleting own keys we need to delete the private key first

3. Encrypt Files

Definition 11.7 Encryption: Encrypt a `file` into a `file.gpg` file using one of your public keys, where recipient is the corresponding e-mail address of this public key user.

```
gpg -r recipient (-e/--encrypt) options file
```

4. Decrypt Files

Definition 11.8 Encryption:

```
gpg (-d/--decrypt) options file.gpg
```

5. pass

Definition 11.1 pass: Is a command-line password manager that stores passwords (or any other sensitive information) in GnuPG encrypted files[def. 11.3] organized in a Password-store usually (a simple directory structure) stored in `/.password-store`. The filenames are usually the title of the websites or resources that require the password.
In order to retrieve passwords into our clipboard:

```
pass options name
```

--multiline/-m: Allows to add additional information s.a. username, url,...

--clip/-c: Copy the password into the clipboard

Avoid running pass without the -c flag, otherwise your password will be printed into the console!

List Available Passwords

```
pass
```

5.1. Creating a new password store

```
pass init [-p path] gpg-id
```

gpg-id is:

- the id (email) of an existing gpg key
- a new id, which will create a new gpg key

5.2. Adding New Passwords

```
pass insert name
```

name should be a descriptive hierarchical name i.e. `archlinux.org/wiki/username`

5.3. Generating&Adding Passwords

```
pass generate options name length
```

name should be a descriptive hierarchical name i.e. `archlinux.org/wiki/username`

--clip/-c: do not show password on command line but copy it into the clipboard

--in-place/-i: do not start a prompt but just replace the first line of the password file with the new generated password, keeping the remainder of the file intact ⇒ can be used to overwrite old passwords see also `--force`

--force/-f: Do not prompt when overwriting an existing password.

5.4. Editing Password Files

```
pass edit name
```

this allows us to add further information about websites a.s. usernames and URLs, where the format is "YAML like":

```
URL: website
Username: username
Secret Question 1: answer
```

add automation with git

add automation with ssh

5.4.1. Extensions

There exists a lot of extensions for pass:

- pass tail: display password meta data but not the password
- passmenu: uses dmenu to select passwords

5.5. Renaming/Copying

```
(mv/cp) [options] oldPath newPath
```

Display Manager

1. Simple Desktop Display Manager (SDDM)

Open A Terminal CTRL+ALT+FX