Linux for Nutanix Administrators

Introduction

This course is constructed of a combination of theoretical parts followed by lab assignments.

Students will get introduced to various topics. Every topic is followed by a lab part when relevant.

Prerequisites

Students should have general knowledge of Operating Systems, Storage- and Networking concepts.

Lab Environment

Every student will have access to a Linux VM.

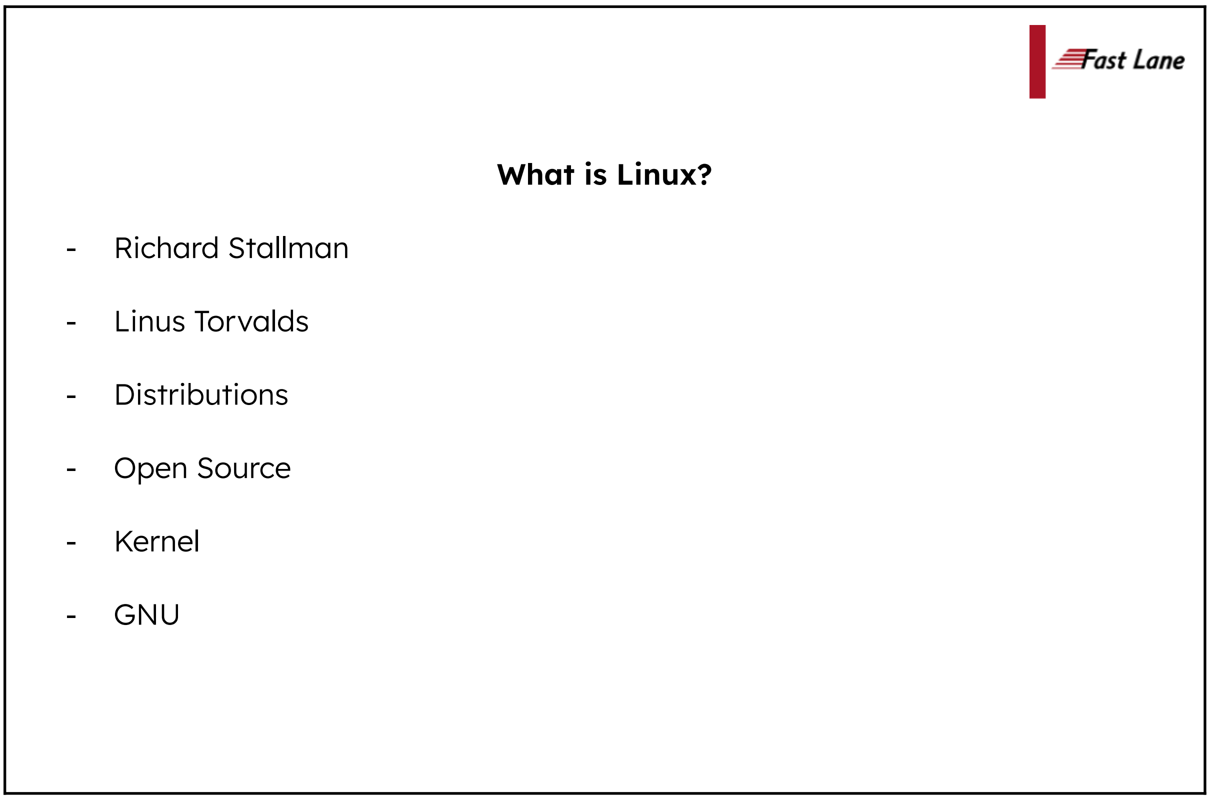
Purpose

The goal of this training is to prepare students to enter the Managing Nutanix Clusters course.

Topics

Accessing the Lab Environment

Consult your instructor on how to login to the lab environment.



What is Linux?

- Richard Stallman
- Linus Torvalds
- Distributions
- Open Source
- Kernel
- GNU

1. What is Linux

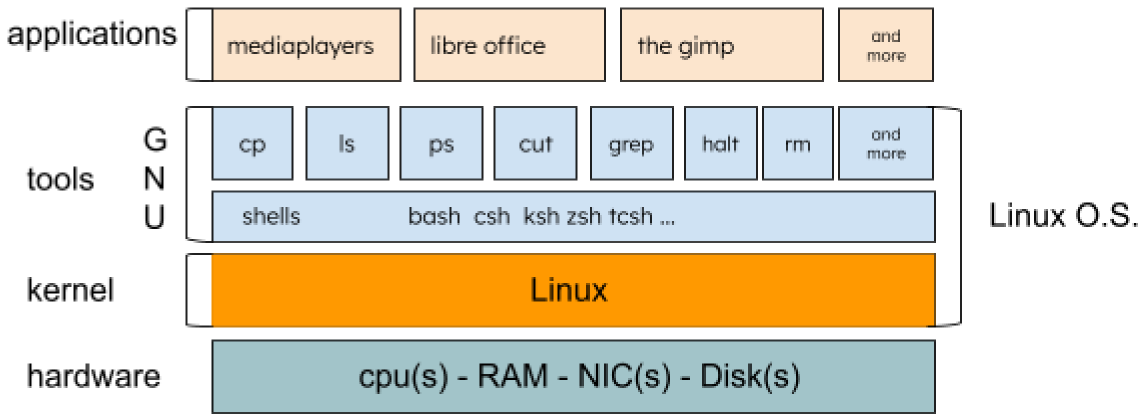
In 1983 Richard Stallman (an American programmer and much more) launched the GNU Project to write a Unix-like computer operating system composed entirely of free software.

In 1991 Linus Torvalds (a Finnish student) began a personal project to create a new free operating system kernel.

Over time, organizations and companies have adopted the Linux Operating System to distribute Linux in the form of - so called - distributions or distros.
Some important initial distros were Slackware, Red Hat and Debian. These were and still are used to create new distros by creating a fork of the original distros. And these forks are often used to fork new environments off of. A popular example of this is: Linux Mint is based upon Ubuntu and Ubuntu in term is based upon Debian.

What is a Linux-type Operating System?

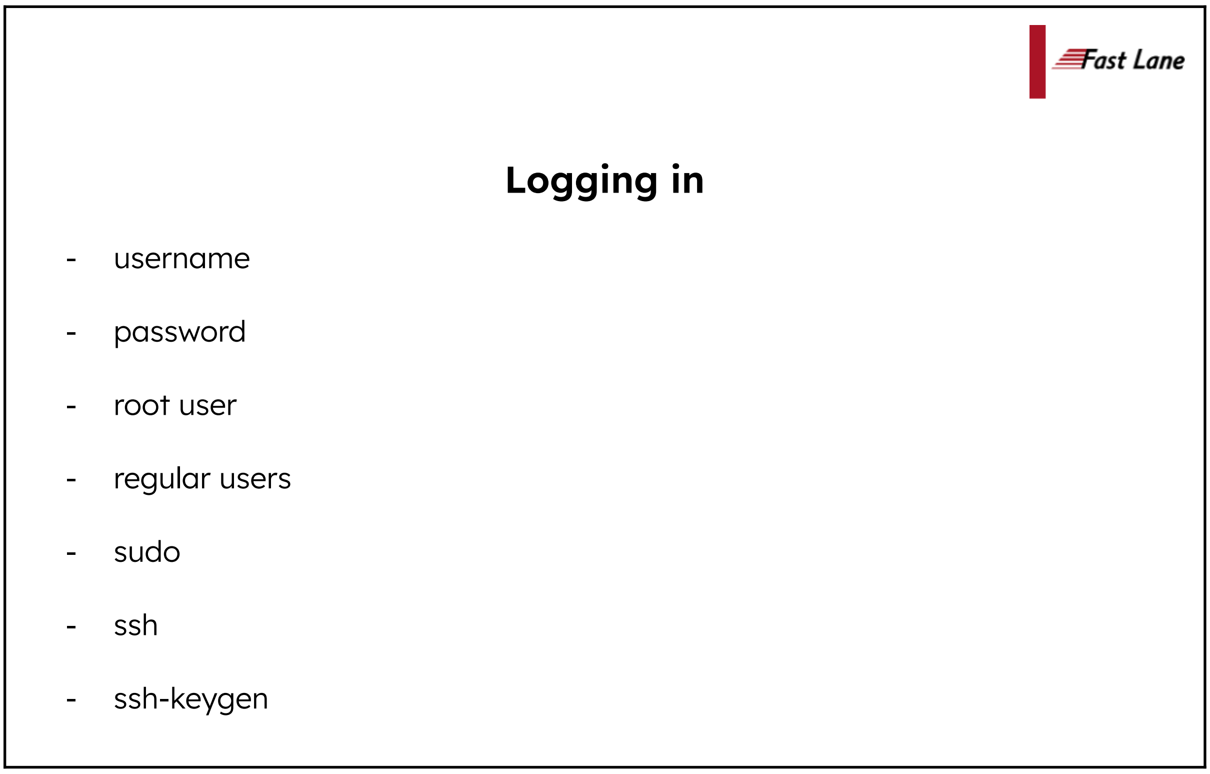
A Linux-type Operating System uses the Linux kernel at the heart of the Operating System, and is accompanied by tools to instruct the Kernel to perform tasks.
In this training we focus on the combination of the Linux Kernel in combination with GNU tools.



So, in short: The kernel is a core component of an operating system and serves as the main interface between the computer's physical hardware and the processes running on it. The kernel enables multiple applications to share hardware resources by providing access to CPU, memory, disk I/O, and networking.

Next to the **kernel** and **tools**, there usually is an **interface** to work with the tools. This interface in our environment will be a shell that offers us the possibility to instruct the kernel to perform actions. There are many shells available to work from. Bash is a popular interface. But you could use the C shell, Korn Shell or any other shell if you liked.

And of course, next to all of this, there are applications that you can run. Think of Database Systems, Mail Applications, Home Automation, Video Editors, et cetera.



Logging in

- username
- password
- root user
- regular users
- sudo
- ssh
- ssh-keygen

2. Logging in

In Linux, you can login via the console if you have direct access to the machine. If you do not have direct access to a machine you can login via the network.

The shell

When a user logs in, usually, a shell will be started for the user to work in. This is true for console logins as well as network logins.

A user has a user id (UID) and a group id (GID). This determines the permissions a user has or does not have. This user id and group id are connected to the shell the user works in.

The root user

The root user is a special case. This user is in charge of everything. The user id for this user is 0 (zero). A user that has or manages to get user id 0, will thus have root permissions.

It is not advised to login as root.

The sudo command

To get root permissions you can use the sudo command. (sudo stands for substitute user do).

Note: Not every user is allowed to get root permissions. This is determined by a configuration environment that allows a regular user to perform one, more or all commands with root privileges.

A user can run a single command by getting root permissions, or a user can start a shell with root permissions.

Running the sudo command will ask the user to type is **own password**. So the user does not need to know the root password.

Some examples:

Actions of a user	The commands to be used
1. Get your username	**`whoami`** `rocky`

2. Get your user id	**`id`**
	`uid=1000(rocky) gid=1000(rocky) groups=1000(rocky),10(wheel)`

3. Reboot with root permissions	**`sudo reboot`**
	`[sudo] password for rocky:`
	`connection to rocky closed`

4. Run a bash shell with root permissions	**`sudo bash`**
	`[sudo] password for rocky:`
	`[rocky]#` **`id`**
	`uid=0(root) gid=0(root) groups=0(root)`

5. Run a command as a user that is allowed to get root permissions.
The user in this case is gandalf
	`sudo reboot`
	
	`We trust you have received the usual lecture from the local System Administrator. It usually boils down to these three things:`
	
	`    #1) Respect the privacy of others.`
	`    #2) Think before you type.`
	`    #3) With great power comes great responsibility.`
	
	`[sudo] password for gandalf:`
	`gandalf is not in the sudoers file. This incident will be reported.`

ssh

To login from one Linux machine to another Linux machine you can use ssh (secure shell). This will allow you to login to another system using a username and password. It is also possible to login with a public key which should be present on the machine you want to login to.

ssh-keygen

Of course also need a public and private key yourself. To generate these you can use the **`ssh-keygen`** command. This command knows many options, but the one you usually use is **`-t`**. With this option you specify the type of key you want. In the example the rsa key type is used.

```
linux : ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):<ENTER>
Enter passphrase (empty for no passphrase): <ENTER>
Enter same passphrase again: <ENTER>
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:41UZY4ALTykw+FGyr6mVHGf8Gw0f4bBxB2sa1buAU7E root@rocky-81
The key's randomart image is:
+---[RSA 3072]----+
|   .+o.  o=++    |
|  . .+o +..=.+   |
|   ... =++E +.   |
|    .o  =X.+.    |
|    . = S.+. .   |
|   . B o * ..    |
|    *   + o      |
|   o     o       |
|  .     .        |
+----[SHA256]-----+
```

Your keys will by default be stored in the **`.ssh`** directory in your login directory.

If you work on a Windows machine you can use PUTTY to securely login to a Linux machine.

When you use ssh or PUTTY, you traffic across the network will be encrypted in transit.

Some examples:

Login using username:password	**`ssh Fatima@192.168.4.117`**
	`Fatima@192.168.4.117's password:`

Copy public key to other machine and them login without a password
 	
Now you can login without a password
	`ssh-copy-id Fatima@rocky-81`
	`(snipped)`
	`rocky@rocky-81's password:`
	
	`Number of key(s) added: 1`
	
	`Now try logging into the machine, with:   "ssh 'rocky@rocky-81'"`
	`and check to make sure that only the key(s) you wanted were added.`
	
	`ssh Fatima@rocky-81`
	`Last login: Fri Nov 17 05:59:05 2023 from 192.168.4.160`

Logging out

To logout of the shell you have a number of options.

1. you can use the **`exit`** command. This will exit the shell you are working in.
2. you can use the combination **`CRTL+D`**.
3. you can use the **`logout`** command.

Manual pages

If you need to find out how a command works you have the opinion to use the manual pages.

For example if you want to know how the **`ls`** command works and what its options are, you can run **`man ls`**.

Another very useful way is to use the --help argument to quickly find out about the command.

```
linux : shutdown --help
shutdown [OPTIONS...] [TIME] [WALL...]

Shut down the system.

Options:
     --help      Show this help
  -H --halt      Halt the machine
  -P --poweroff  Power-off the machine
  -r --reboot    Reboot the machine
  -h             Equivalent to --poweroff, overridden by --halt
  -k             Don't halt/power-off/reboot, just send warnings
    (snipped)
```

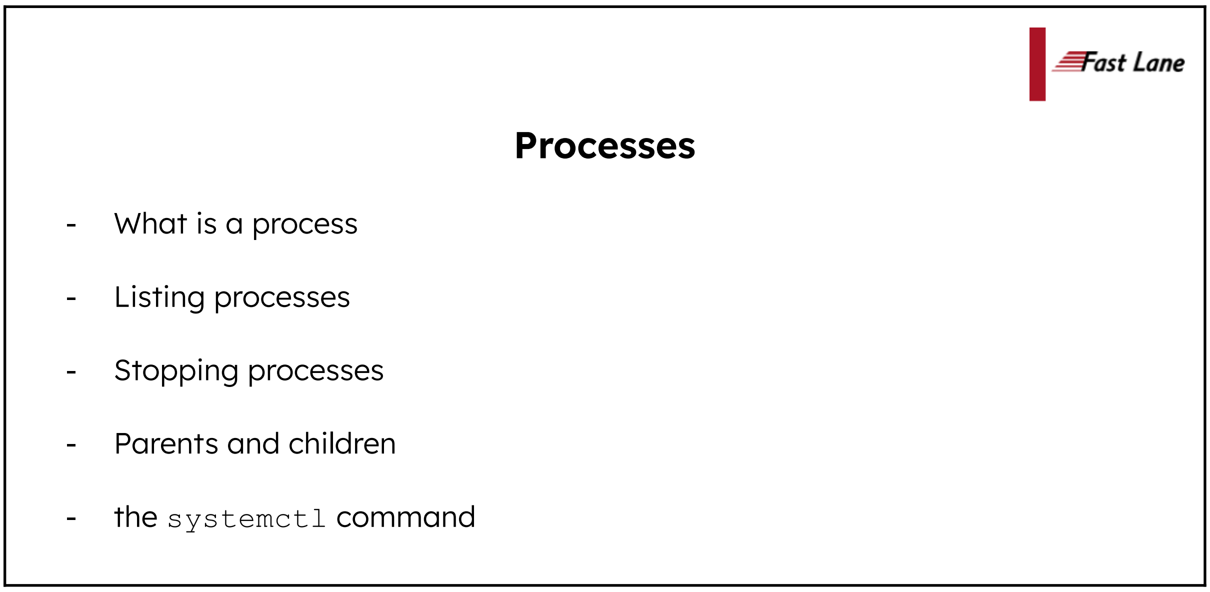
2. Logging in - Exercises

	Exercise 1
1-1	Login to your system with your **username** and **password**
1-2	What is your user id? Hint: use the **id** command.
1-3	Try to create a new user. Hint: use the **useradd yourname** command. If this fails. Can you explain why?
	Try to create a new user using sudo. Hint: use the **sudo useradd yourname** command.
1-4	Try to give the user a password. What is the response of the system? Hint: **passwd yourname** If this fails, can you explain why?
	Now use the sudo command to give the user a password. Hint: **sudo passwd yourname**

	Exercise 2
1-1	Login to your system with username and password
1-2	Run the **`exit`** command. This will log you out. Log in again as in 1-1.
1-3	Run the **`logout`** command. This will log you out. Log in again as in 1-1.

	Exercise 3
1-1	Login to your system with username and password
1-2	Use the **`manual pages`** to find out about the **`ls`** command.
1-3	Use the **`--help`** argument to find out about the possible options for the **`reboot`** command.

	Exercise 4
4-1	Login to your system with username and password
4-2	Use the **`ls`** command to list the contents of the **`.ssh`** directory.
	If the directory does not exist, this means there are no keys. You will then have to create new keys. Create new keys using **`ssh-keygen -t rsa`** (Accept all defaults by pressing the <ENTER> key. Three times.)
	List the contents of the .ssh directory.



Processes

- What is a process
- Listing processes
- Stopping processes
- Parents and children
- the `systemctl` command

3. Processes

What is a process

A process is a program that is executed in memory. Every program that you run will have its own process in memory. When you start an application, multiple processes may get started because an application may run multiple single programs that make up the application. In general when you run any of the GNU tools, each tool will by default be run in its own process which will be exited when the program is finished. So many processes are there only for the duration of the program.

So when a user runs a command, the user usually does that from a shell like the bash shell. This bash shell is connected to a terminal in which the user can type commands.

Other processes are not connected to a terminal. These processes get started when the system starts and are called **daemons**. They perform tasks like running a webserver, allowing network connections, managing a database, et cetera.

Listing processes

Some commands to view your processes:

ps

This command prints the status of one or more processes.

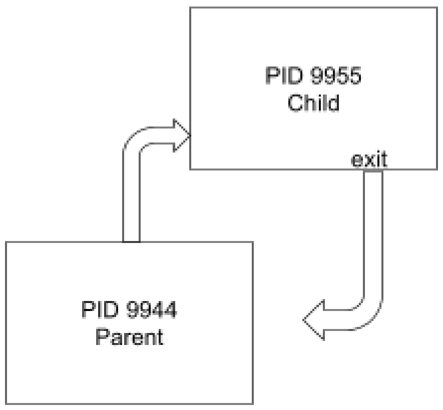
```
#list the process you work in at the moment.
linux : ps
    PID TTY          TIME CMD
  21440 pts/0    00:00:00 bash
  21460 pts/0    00:00:00 ps
```

The first column is the **process id**(PID).
The second column is the **terminal** it is connected to.
The third column is the elapsed time the process has run on the CPU.
The fourth column is the executable command.

```
#print a full listing of your process.
linux : ps -f
UID         PID   PPID  C STIME TTY          TIME CMD
rp         9944   9943  1 07:45 pts/1    00:00:00 -bash
rp         9955   9944  0 07:45 pts/1    00:00:00 ps -f
```

The full listing shows that a process can be the parent of another process (child). In this example the parent process is PID **9944** and the child process is **9955**.

parents and children



A single process can be a child process as well as a parent process. In fact, a process is an instance of a started program that can start other programs.

```
#print a full listing of all processes.
linux : ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Oct09 ?        00:00:04 /usr/lib/systemd/systemd
root           2       0  0 Oct09 ?        00:00:00 [kthreadd]
root           3       2  0 Oct09 ?        00:00:00 [rcu_gp]
root           4       2  0 Oct09 ?        00:00:00 [rcu_par_gp]
root           5       2  0 Oct09 ?        00:00:00 [slub_flushwq]
root           6       2  0 Oct09 ?        00:00:00 [netns]
root           8       2  0 Oct09 ?        00:00:00
[kworker/0:0H-events_highpri]
root          10       2  0 Oct09 ?        00:00:00 [mm_percpu_wq]
root          12       2  0 Oct09 ?        00:00:00 [rcu_tasks_kthre]
root            13       2  0 Oct09 ?        00:00:00 [rcu_tasks_rude_]
root            14       2  0 Oct09 ?        00:00:00 [rcu_tasks_trace]
root            15       2  0 Oct09 ?        00:00:02 [ksoftirqd/0]
(output snipped)
```

The above command gives a full listing of all processes running on your system.

top

This command displays and updates a sorted list of running processes.

```
linux : top

top - 17:58:51 up 15 days,  4:35,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 136 total,   1 running, 135 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
MiB Mem :   1274.1 total,    726.4 free,    344.6 used,    372.6 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.    929.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0  106644  17820  10668 S   0.0   1.4   0:03.20 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_fl+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker+
```

As you can see in the above output, the top command gives a lot of information about your system, next to listing processes.

How to stop a process

When a program is done with its task, it will **exit** and the process is stopped and cleared automatically. So, for example, if you run the **ps** command, the ps command runs and when it has performed its task, it exits.

kill

You can also stop processes by sending a *signal* to the process you want to stop. For example: imaging a process with PID 12345 has to be stopped, you can send signal 15 to the process or signal 9. Signal 15 is a friendly way of asking the process to stop. Signal 9 is an unfriendly way of simply stopping the process.

The command to do that is the kill command. To stop **PID 12345** you can run:

```
linux : sudo kill -9 12345
```

Note: The root user of a system has the right to kill any process. A regular user is by default only allowed to kill its own processes.

Daemons and Services

A daemon is a background, non-interactive program. It is detached from the keyboard and display of any interactive user. The word daemon for denoting a background program is from the Unix culture; it is not universal.

A service is a program which responds to requests from other programs over some inter-process communication mechanism (usually over a network). A service is what a server provides. For example, the NFS port mapping service is provided as a separate portmap service, which is implemented as the portmapd daemon.

A service doesn't have to be a daemon, but usually is. A user application with a GUI could have a service built into it: for instance, a file-sharing application.

Daemons and Services usually get started at boot time.

The program responsible for starting and monitoring daemons and services is the **systemd** program.

systemctl

The command to manage the systemd environment is the **systemctl** command. Systemctl is part of the systemd environment for creating and managing services. In this training we focus on stopping and starting existing services.

An example.

The task scheduler in Linux is `cron`. The service is named the `crond.service`. Based on configuration settings, the `crond.service` will perform tasks at certain moments. To list the status of the crond.service you can run the following command:

```
linux : systemctl status crond.service
● crond.service - Regular background program processing daemon
     Loaded: loaded (/lib/systemd/system/crond.service; enabled
Active: active (running) since Sun 2023-07-02 14:17:07 CEST; 4
months 3 days ago
    (output snipped)
```

Next to status information you can also stop, start and restart.

```
linux : sudo systemctl stop crond.service
linux : sudo systemctl start crond.service
linux : sudo systemctl restart crond.service
```

To prevent a service from being started at boot time, you can disable it. And of course to reverse that, you can enable the service again.

```
linux : sudo systemctl disable crond.service
linux : sudo systemctl enable crond.service
```

Variables

A variable is a placeholder for data in a process. A process can have many variables. Some are set by default, some can be created and managed manually. Variable are local to a process. If the process exits, all variables are gone.

A variable can be '*exported*'. This means that the variable will be present in the child processes as well.

To fill a variable you enter the variable name and then an equal sign, directly followed by a value. No spaces surrounding the equal sign. Variable names are case sensitive.

Some examples:

```
linux : name=fatima
linux : PS1="Myprompt : "
linux : counter=0
```

unset

To empty a variable you can unset the variable. For example, to unset the variable name, you run the following command:

```
linux : unset name
```

echo

To list the content of a variable you can use the echo command, and use a $ sign in front of the variable name.

```
linux : echo $counter
0
```

Task: determine the value of the following variables: **`HOME, PATH, PWD, USER, HOSTNAME`**.

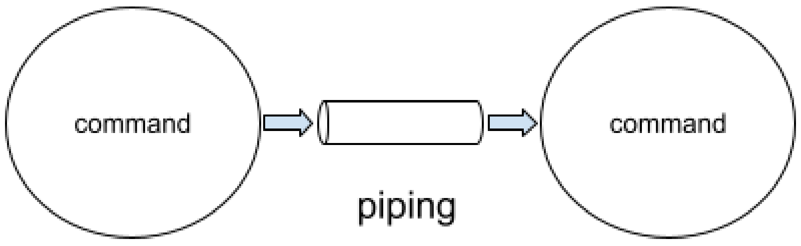
set

To list all variables you can run the set command.
This commands also displays functions. Functions are beyond the scope of this training.

```
linux : set
BASH=/bin/bash
BASHOPTS=checkwinsize
(snipped)
```

command line piping

When a command is executed, the output of the command will be written to the terminal. Sometimes you want to do some additional processing of the data before it is finally written to the terminal. This can be done by sending the output to an unnamed pipe instead of to a terminal. Another program can then receive the data from the unnamed pipe and do further processing.



For example:

You run the ls command. This will print the filename in a directory to the terminal.
If you want to know how many files there are, you can send the output to a program that will count the lines and print that to the screen. A command line pipe is created by the | sign. The format is:

```
command | command | command | command
```

Without command line piping	`linux : ls`
	`a  b  c`

With command line piping `(wc -l)` counts the number of lines.	`linux : ls | wc -l`
	`3`

output redirection

By default, commands will output to the terminal to which the bash shell is connected. This simply means that the output of a command will print to the terminal. Instead of printing output to the terminal you can save it to disk (for later use). This can be done by using the greater than sign: **>** but also by using two greater than signs: **>>**.

A single > means that the file will be created if it does not exist. It will be emptied first if it does exist. If you use two: >>, it will create the file if it doesn't exist or append to the file if it does exist.

Example:

List the contents of a directory.	`linux : ls`
	`a  b  c`

Output the contents to a file.	`linux : ls > files`
	`linux : cat files`
	`a`
	`b`
	`c`
	`files`

Append the contents to a file.	`linux : ls >> files`
	`linux : cat files`
	`a`
	`b`
	`c`
	`files`
	`a`
	`b`
	`c`
	`files`

Empty the file.	`linux : > files`

Note: there is also input redirection and double input redirection.

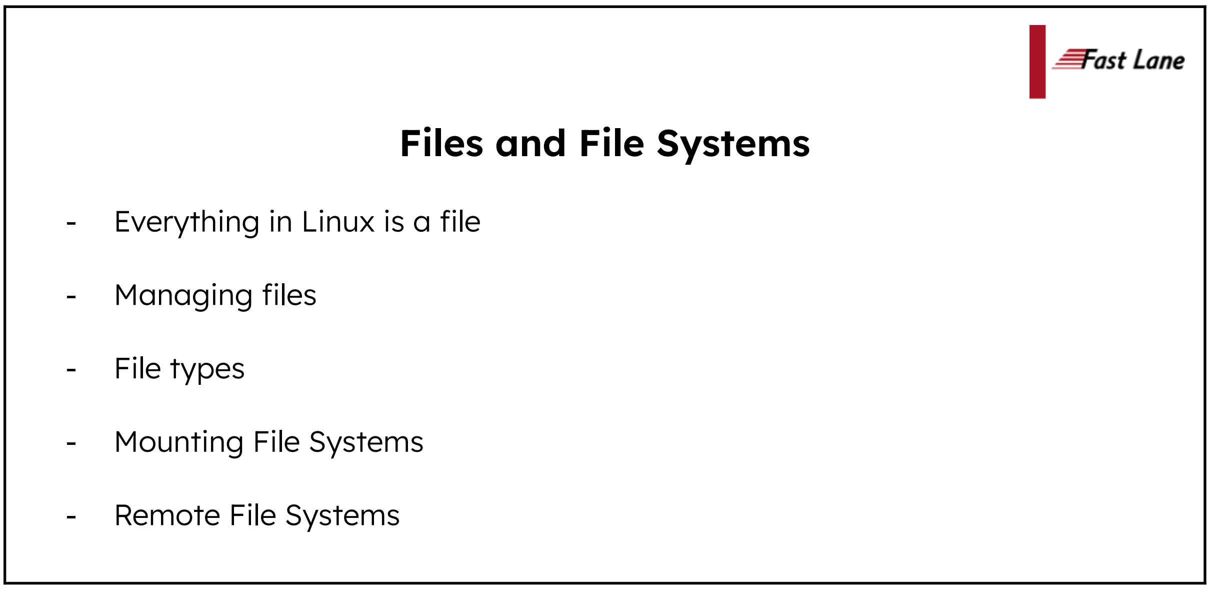
3. Processes - Exercises

	Exercise 1
1-1	Login to your system
1-2	Use the ps command to show a full listing of your own processes your processes. - What is the PID of the **bash** shell? - What is the PID of the **ps** command? hint: use the **-f** option.
1-3	Run the same command again. Is the PID of the ps command the same as the first time? Explain your answer.
	What is the Parent Process of the **ps** command?
1-4	Run the bash command. Run the **ps** command again to view a full listing of your own processes. What is the PPID of the **ps** command?
	Run the following command: **echo $$**. What does this tell you?

	Exercise 2
2-1	Login to your system
2-2	Run the following command: `while true; do find / ; sleep 1; done` This command lists all files on your system and continuously generates output. Open a new tab in your browser and login once more.
2-3	Run the **top** command.
	Which process is listed in the top processes?

	Exercise 3
3-1	login to your system.
3-2	Change your system prompt into the following: **mymachine :** hint: use the **PS1** variable.
3-3	Create a variable with the name “**letter**” with the value “**A**” Print the content of the variable using the **echo** command.
	From your current shell, start a new shell by using the **bash** command. Print the content of the variable letter, using the **echo** command followed by the variable name. *The variable should be empty.* exit the current shell.
	From your current shell, export the variable using the export command and the variable “**letter**”. Again, start a new shell from your current shell. Print the content of the variable letter, using the **echo** command followed by the variable name. *The variable should not be empty.*

	Exercise 4
4-1	login to your system.
4-2	Stop the crond.service by running the following command: **`systemctl stop crond.service`** What is the response you get?
4-3	Start the cron.service again using the sudo command in front of the actual command.
4-4	With the reboot command you can reboot a system. Instead of the reboot command use the shutdown command to reboot your system. Use the manual pages to find out about the correct option to use.



Files and File Systems

- Everything in Linux is a file
- Managing files
- File types
- Mounting File Systems
- Remote File Systems

4. Files and File Systems

Files

In Linux, everything is a file, and if it is not a file, it is a process. Files are not just text files, images and programs. Also disks, directories, terminals, et cetera..everything is a file.

case sensitive

Linux filenames are case sensitive: **Data** is not the same as **data**.

ls

The `ls -l` command will show you the properties of a file, including the file type. This is printed as the first character in the output of the ls -l command.

```
linux : ls -l
total 4
drwxr-xr-x 2 root root 4096 Nov  4 09:58 confdir
-rw-r--r-- 1 root root    0 Nov  4 09:58 emptyfile
lrwxrwxrwx 1 root root    7 Nov  4 09:58 linktoconfdir -> confdir
```

In the above example you see a directory (d), a regular file (-) and a symbolic link (l).
Other file types are: block device (b), character device (c), socket (s), pipe (p).

file

A nice command to determine what type of information the file contains is the **file** command.
.

```
linux : file *
confdir:     directory
confdirlink: symbolic link to confdir
emtpyfile:   empty
```

Linux File Commands

cp	copy	`cp file1 file2`
rm	remove	`rm file2`
mv	rename or move	`mv file1 file3`
file	list file type	`file file1`
touch	create file or update timestamp	`touch emptyfile`

Directories

The purpose of a Linux directory is to organize files. Directories can contain other directories. There are many directories. In this document we list some important ones.

/boot	kernel	
/usr/bin	binaries	/bin is a symbolic link
/usr/lib	libraries	/lib is a symbolic link
/usr/sbin	system binaries	/sbin is a symbolic link
/etc	configuration files	
/home	user login directories	
/var	logfiles	
/tmp	temporary files	
/mnt	to mount remote filesystems	

Navigating the File System

To navigate the file system you can use the **cd** command. When you enter the cd without any arguments, it will take you to your login directory. If you pass arguments to the cd command, you either use an absolute pathname or a relative pathname.

The difference between an absolute pathname and a relative pathname is really simpel. An absolute pathname starts with a **/** (slash). All other pathnames are relative.

Pathnames

/home	absolute
../data	relative
data	relative
..	relative
./prog	relative

Creating and removing directories

To create a directory you use the mkdir command. You always add a pathname as argument. This may be a relative or absolute pathname. For example:

```
mkdir /tmp/mydir
mkdir mydir
mkdir ../data1
```

If you create a path that does not exist yet, you can use the -p option to also create the in between directories.

```
mkdir -p /home/newdir/mydir
```

To remove a directory you use the **rmdir** command. To remove a directory, the directory has to be empty. If the directory contains files or other directories, you will get an error message.

If you want to still remove a directory that is not empty, you can use the **rm** command in combination with the **-r** option. This means that you will recursively remove the directory and whatever is in it. And if you want to ignore all questions you can use the -f option in combination with the -r option.

rm

```
rm -rf /home/newdir
```

Note: be careful with this, because this cannot be undone.

Editing files

For editing text and configuration files, it is best to learn how to use the `vi editor`, because it is “closest” to linux administration. Learning `vi` is a steep learning curve. Because of that, for this training we use `nano`, which is easy to work with.

To start nano, you run the nano command with an optional filename for an already existing file. The most important key-combinations are **`^O`** for writing and **`^X`** for quitting.

If you want to quit a modified file, you can either save it or not save it by using **Y** or **N** upon quitting.

Mounting File Systems

In this training we are not going to create file systems on disk. But we are going to mount and ISO filesystem as well as a Remote File System using the NFS protocol.

mount

Mounting an ISO file

Imagine you have an operating system image in the form of an ISO file. You want to access the ISO file for some reason. Maybe you need to install some software from it. You can mount the ISO file system in two steps.

1. Create a directory to mount it to. (mount point)
   `mkdir /mnt/cdrom`

2. Mount the ISO
   `mount -o loop /mnt/iso/rocky.iso /mnt/cdrom`

Note: a *loop* mount allows you to mount a file.

If you need to list your mounted file systems you can use the df -h command. This will print a human readable list of mounted file systems. Human readable means that sizes and capacities are printed in human readable format instead of in bytes.

```
linux : df -h
Filesystem                                         Size  Used Avail Use%
Mounted on
/dev/mapper/rl-root                                 36G   24G   13G  67% /
(output snipped)
/dev/loop0                                         1.5G  1.5G     0 100%
/mnt/cdrom
```

In the list you see that the root file system **/dev/mapper/rl-root** is mounted to **/** and that the **/dev/loop0** device (your ISO) is mounted to **/mnt/cdrom**.

Disk Based File Systems

Linux supports many types of Disk Based File Systems. These File Systems differ in many respects. The details of those differences are beyond the scope of this training. As the term says, these File Systems are on disk. In order to be able to create such a file system you need to know what your drives are on which to create these file systems.

lsblk

The `lsblk` command will list the block devices that are present on your system.

```
Linux : lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda            8:0    0     1G  0 disk
sr0           11:0    1 1024M  0 rom
vda          252:0    0    20G  0 disk
├─vda1       252:1    0     1G  0 part /boot
└─vda2       252:2    0    19G  0 part
  ├─rl-root 253:0    0    17G  0 lvm  /
  └─rl-swap 253:1    0     2G  0 lvm  [SWAP]
```

Note: the device names may differ per as created by the hypervisor.

mkfs

The `mkfs` command creates a filesystem on a device.

```
Linux : mkfs /dev/sda
```

Note: the default filesystem type is **ext2**.

Mounting a disk based File System

To mount a disk based file system, you need a mountpoint. This is a directory to which you mount the file system.

```
Linux : mkdir /mnt/smalldisk
Linux : mount /dev/sda /mnt/smalldisk
```

Mounting a Remote File system

To mount a remote file system you need three pieces of information:

1. What is the ip address or hostname of the remote server?
2. What is the directory on that remote server that you want to mount?
3. What is the local directory you want to mount the remote file system to?

An example:
The remote server ip address is `192.168.4.117`
The remote file system is `/SelfServiceContainer`
The local directory is `/mnt/remotecontainer`

```
linux : mkdir /mnt/remotecontainer
linux : mount 192.168.4.117:/SelfServiceContainer /mnt/remotecontainer
linux : df -h | grep remotecontainer
192.168.4.117:/SelfServiceContainer  1.1T  3.8G  1.1T   1%
/mnt/remotecontainer
linux :
```

Automatic mounts at boot time

/etc/fstab

The file /etc/fstab can be configured to mount file systems at boot time.

The following is a possible entry in the /etc/fstab file.

device	mount point	file system type	options	0	0
/dev/sda	/mnt/diska	ext2	defaults	dump or not	fsck order

When there is an entry in the /etc/fstab file, you can run the mount command with the device or the mount point as the argument.

In the about fstab example, to mount the file system, it would be sufficient to runt the following command:

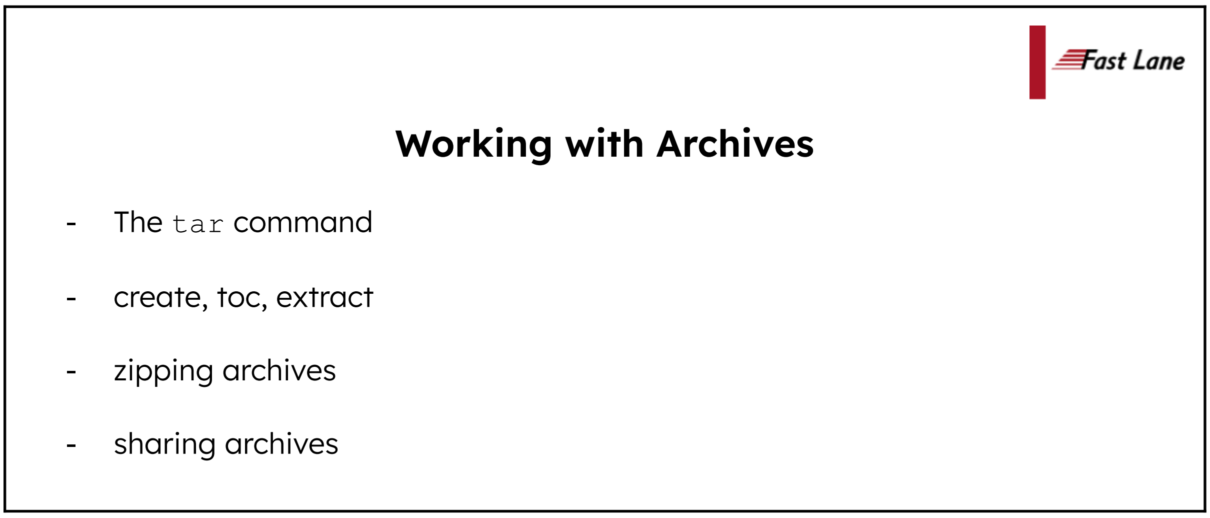
```
mount /mnt/diska
```

The system will check /etc/fstab and pick the device **/dev/sda** to mount to the mount point **/mnt/diska**

Files and Filesystems - Exercises

	Exercise 1
1-1	Login to your system
1-2	Use the `lsblk` command to list your blockdevices. Note: try the **`-f`** option to also see filesystems if they are present.
1-3	Pick any of the available disks that has no filesystem yet. Note: make sure that the disks are not in use. Disk **'vda'** is definitely in use. hint: /dev/sda is not in use.
	Create a default (ext2) filesystem on the disk. hint: use the `mkfs` command on device /dev/sda
1-4	Create a directory to be used as a mountpoint. For example: **`mkdir /mnt/data`**
	Mount the filesystem. For example: **`mount /dev/sda /mnt/data`**

	Exercise 2
2-1	Login to your system
2-2	Make sure the sda device is mounted at boot time. hint: edit the /etc/fstab file.



Working with Archives

- The `tar` command
- create, toc, extract
- zipping archives
- sharing archives

5. Working with archives

When you want to group files together in one file, or store files on disk as if on a tape device, or when you want to compress files into a single file and send this file to someone else via a network connection or by using a mail client, you can use the `tar` command.

Like other commands, `tar` knows multiple options, but only a few are very important.

tar

c	create	create a file
t	table of contents	list all files in the tarfile
x	extract	extract files
z	zip	create or extract zipped file
f	file	the name of the file to create or extract

Some examples:

1. List the contents of a directory and then archive these files into a single tarfile.

```
linux : ls
a  b  c  d  e  f  g

linux : tar cf collected.tar .
tar: ./collected.tar: file is the archive; not dumped
```

Note: the message means that the tarfile itself is not part of the created archive.

```
linux : ls
a  b  c  collected.tar d  e  f  g
```

2. Extract the files from the archive after having removed the actual files.

```
linux : rm a b c d e f g
linux : ls
collected.tar

linux : tar xf collected.tar

linux : ls
a   b   c   collected.tar d   e   f   g
```

3. Create a zipped tarfile and list its contents.

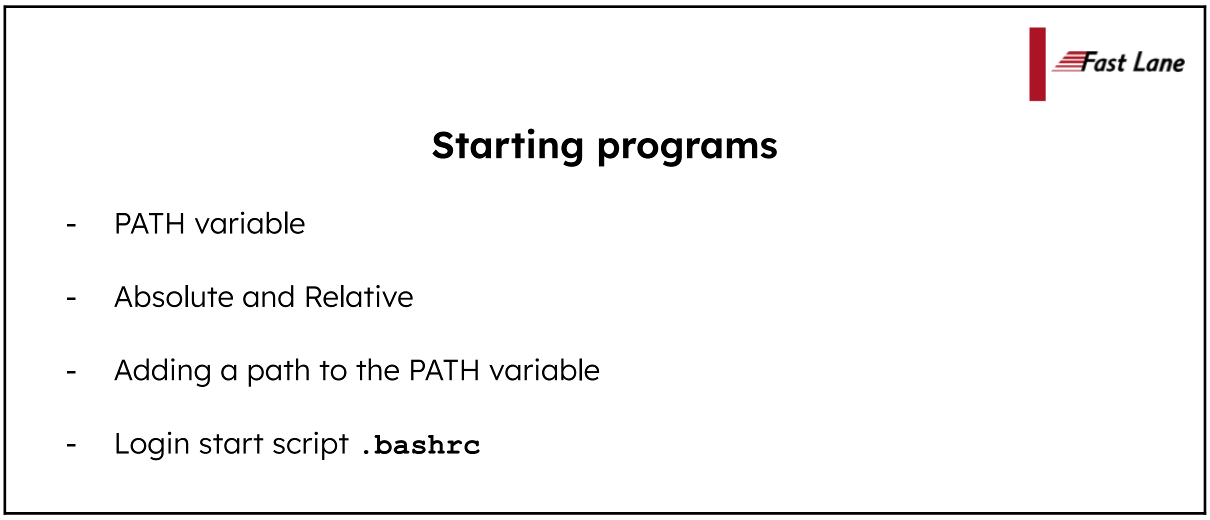
```
linux : tar zcf zipped.tar .
linux : tar tf zipped.tar
./
./e
./c
./g
./b
./collected.tar
./f
./d
./a
```

Sharing tarfiles

Tarfiles can be sent to others any way you like. For example, you can create a zipped file and send it via email or using the winscp or scp.

Working with archives - Exercises

	Exercise 1
1-1	Login to your system
1-2	Create a directory "collect" in your login directory. hint: use `mkdir`.
1-3	Copy all files from the /etc/ directory to the collect directory. hint: use cp
	In the collect directory create a zipped tarfile of all files in collect. hint: use tar zcf
1-4	Remove all files from the content directory except the tarfile. hint: use rm
1-5	Extract the tarfile to restore all files in the collect directory. hint: use tar zxf



Starting programs

- PATH variable
- Absolute and Relative
- Adding a path to the PATH variable
- Login start script **`.bashrc`**

6. Starting Programs

A Linux shell like the bash shell has some internal commands like cd, exec, pwd and more. These programs are executed inside the shell. Other programs (most programs) are not present in the shell but on disk. When you execute such a program from disk, the program will be executed in a child process and when the program is finished, it will exit. The child process will then be ended.

The PATH variable

The PATH variable is used by the shell to determine which directory to search through to find a program that needs to be executed. If the shell cannot find the program in any of the directories that are listed in the PATH variable, it will return a **command not found** error.

Instead of relying on the PATH variable you can also enter a relative or absolute path to start the program.

For example:
```
/usr/local/bin/myprog
./myprog
../myprog
```

If you want to change the PATH variable to widen the search for particular commands which are present in other directories than the default, you can change the PATH variable manually. Or you can change the PATH variable from within the startup script that the shell uses. If the bash shell is used to work in, then there is a **.bashrc** script in the user's login directory.

To list the contents of the PATH variable:

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
```

To change the PATH variable on the commandline:

```
PATH=$PATH:/home/john/myprograms
```

The above command will take the content of the current PATH variable and add your new directory to the list of directories.

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/home/john/myprogram
s
```

(just to make sure: it does not help to change to the directory in which the program resides. Because Linux does not look in the current working directory by default. It is also not best practice to add the current working directory to the PATH variable, for security reasons).

which

If you want to find out where a program is located you can use the `which` command. This command sequentially scans the PATH variable. If it finds the command it will list the program and the directory. If it cannot find the program it will say so and also tells you in which directories it cannot be found. If the command cannot be found, this does not mean that the command is not on disk. You may have to change the PATH variable.

```
linux : which reboot
/usr/sbin/reboot

linux : which REBOOT
/usr/bin/which: no Reboot in
(/root/.local/bin:/root/bin:/usr/sbin:/usr/bin:)
```

Next to simply typing the command to execute it, you can also use the path name of the program to start it. This way you are always sure exactly which command you run.

```
linux : /usr/bin/ls
a b c
```

find

If you do not know the location of a program or other file, but you need to find it, you can use the find command. The find command in its simple form:

1. where to start looking (what is starting pathname)
2. the name of the file

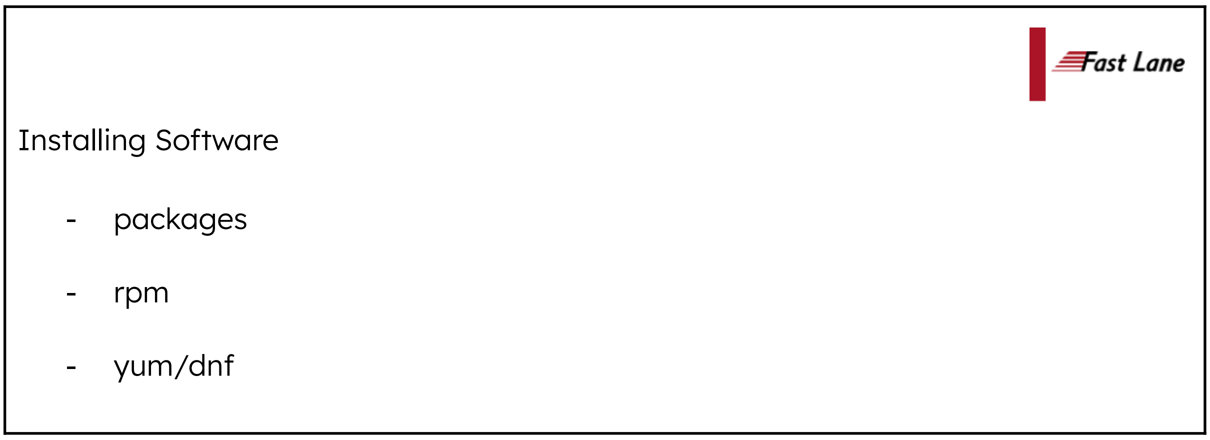
Example: start looking from the `root directory` for files with the name "`reboot`".

```
linux : find / -name reboot
/sys/kernel/reboot
/sys/fs/selinux/class/system/perms/reboot
/usr/sbin/reboot
```

Starting Programs - Exercises

	Exercise 1
1-1	What is the content of your PATH variable? hint: use the `echo` command.
	How can you determine which path the shell uses to execute the ls command? hint: use the `which` command.
	Can you also overrule the PATH variable? hint: use an absolute pathname.

	Exercise 2
2-1	Create a directory called `scripts` in your login directory. hint: use the `mkdir` command.
	Create a file that runs the `ps` command. The file should be created in your new scripts directory. The name of the file should be “`myprocs`” hint: use the `echo` command with `output redirection.`
	Make the file `myprocs` executable. hint: use the `chmod +x` command.
2-2	Add the new `scripts` directory to the `PATH` variable. hint: use the `PATH=$PATH:$HOME/scripts` construction.
	Add the above construction to your **`.bashrc`** This will set the new PATH every time you login. hint: use the `echo` command with `output redirection.`
2-3	Log out and log in again. Use the which command to check that you `myproc` script can be found. Run the `myproc` script without any arguments.



Installing Software

- packages
- rpm
- yum/dnf

7. Managing Software

Software in Linux environments is usually managed in the form of packages. In Red Hat based distributions this is in the form or RPMs (Redhat Package Manager) with the **`.rpm`** suffix. In Debian-based distributions it is in the form of DPKGs (Debian Packages) with the **`.deb`** suffix. The focus in this training is on RPMs.

RPM

An RPM package can contain an arbitrary set of files. Most RPM files are "**`binary RPMs`**" (or BRPMs) containing the compiled version of some software. There are also "**`source RPMs`**" (or SRPMs) containing the source code used to build a binary package.

Some commands to manage RPMs:

Install a package	**`rpm -ivh ksh.rpm`**
List package dependencies	**`rpm -qpR ksh-1.0.0~beta.1-3.el9.x86_64.rpm`**
List all packages	**`rpm -qa | more`**
Remove a package	**`rpm -e ksh`**
Remove without checking dependencies	**`rpm -e --nodeps ksh`**

Check the package in which a file resides.	**`rpm -qf /usr/bin/ls`**
	`coreutils-8.32-34.el9.x86_64`

List all files in a package	**`rpm -ql ksh`**
	`/etc/binfmt.d/kshcomp.conf`
	`/etc/kshrc`
	`/etc/skel/.kshrc`
	`/usr/bin/ksh`
	`(snipped)`

YUM[1]

The problem with singular packages is that there usually are dependencies, between packages. The package manager does not resolve these dependencies. The solution for this is Yum. YUM performs dependency resolution when installing, updating, and removing software packages. YUM can manage packages from installed repositories in the system or from . rpm packages. These repositories can be listed with the **`yum repolist`** command.

Some commands to work with yum.

Install a package	**`yum install -y ksh`**
Update a package	**`yum update -y ksh`**
Remove a package	**`yum remove ksh`**
Download a package to the current directory	**`yum install ksh --downloadonly --downloadaddir .`**
Check the package name of un uninstalled program	**`yum whatprovides ksh`** `Last metadata expiration check: 1:20:05 ago on Wed 29 Nov 2023 06:23:26 AM EST.` `ksh-3:1.0.0~beta.1-3.el9.x86_64 : The Original ATT Korn Shell` `(snipped)`
List all available packages	**`yum list all`**
List all installed packages	**`yum list installed`**
List all package groups	**`yum group list`** `Last metadata expiration check: 0:28:24 ago on Wed 29 Nov 2023 06:23:26 AM EST.` `Available Environment Groups:` `Server with GUI` `Server` `Workstation` `(snipped)`

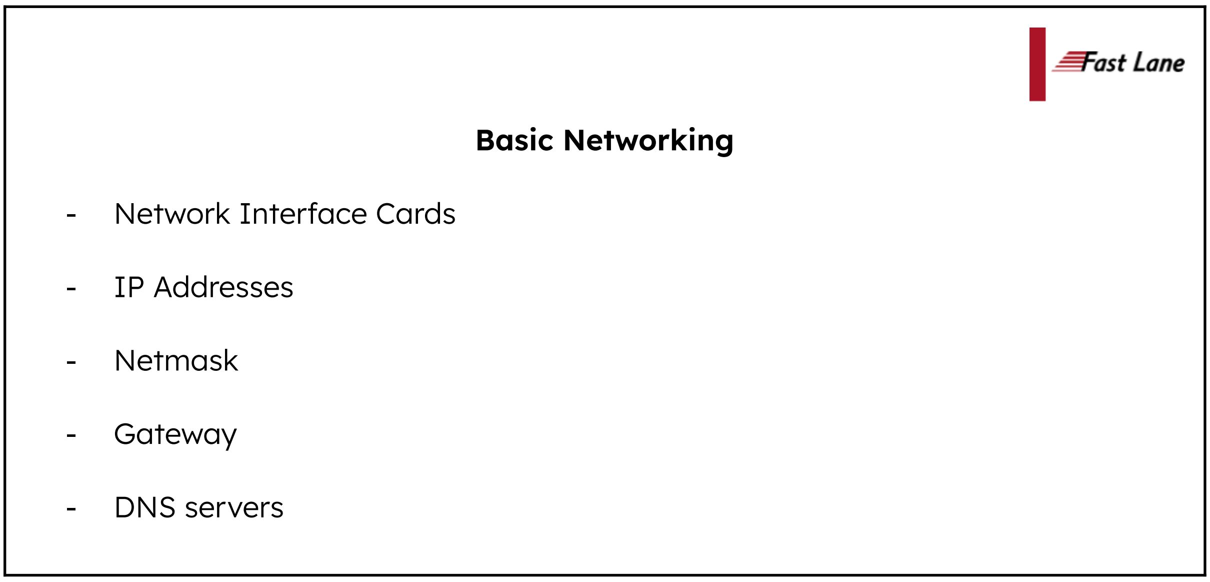
[1] Red Hat based systems now come with a updated version of YUM called DNF (Dandified YUM)

The configuration directory where yum searches for repositories on the Internet or local network is **`/etc/yum.repos.d`**. The so called *repo files* contain URLs and other configuration data.

Managing Software - Exercises

	Exercise 1
1-1	In this exercise you will install the vsftpd (very secure ftp server daemon).
	Check whether the vsftpd package is installed. hint: use `rpm -q`
	Check whether the vsftpd package is an available package. hint: use `yum list all`
	Check which package contains the vsftpd program hint: use `yum whatprovides`
1-2	Download the vsftpd package to your current working directory without installing it. hint: use `yum install --downloadonly`
	Check the success of the download. hint: use the ls command
1-3	Install the package using the rpm command.
	Check the success of the installation. hint: use the `rpm -q` command
	Remove the vsftpd package. hint: use the `rpm -e` command
1-4	Install the vsftpd package using yum install.
	Remove the vsftpd package using yum remove.

	Exercise 2
2-1	In this exercise you will update your system using `dnf`
	Run the following command: **`dnf update`** Before you enter “y”, answer the following questions: How many packages will be installed? Why are these packages going to be installed? How many packages will be upgraded? What is the download size? type **`y`**



Basic Networking

- Network Interface Cards
- IP Addresses
- Netmask
- Gateway
- DNS servers

8. Basic Networking

Network Interface Cards

To connect to a Local Area Network, a Linux machine needs a Network Interface Card. One can connect via WIFI or a physical LAN using network cables. In this module we focus on physical connections.

Despite all the examples you get: it is best practice to use the `nmcli` command.

Links

To list the physical of virtual Network interface cards, there are multiple commands you can use.

The following list is not complete.

lspci **Note: `lspci` is part of the package `pciutils`**	Is a utility for displaying information about PCI buses in the system and devices connected to them.
ethtool	Is a utility to query or control network drivers and hardware settings
ifconfig **Note: `ifconfig` is part of the package `net-tools`**	Is an outdated network configuration utility
ip	Recommended new network config utility
lshw	identify Ethernet interfaces and NIC hardware.
nmcli	List connections with the network manager command

Some examples:

```
linux : lspci | egrep -i --color 'network|ethernet'
01:00.0 Ethernet controller: Red Hat, Inc. Virtio network device
(rev 01)
```

```
linux : sudo ethtool enp1s0
Settings for enp1s0:
      Supported ports: [  ]
      Supported link modes:   Not reported
```

```
     Supported pause frame use: No
     Supports auto-negotiation: No
(snipped)
```

```
linux : ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.4.189  netmask 255.255.255.0  broadcast
192.168.4.255
        ether 52:54:00:43:15:fc  txqueuelen 1000  (Ethernet)
        RX packets 750568  bytes 100404584 (95.7 MiB)
        RX errors 0  dropped 87818  overruns 0  frame 0
        TX packets 5372  bytes 622021 (607.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions
0
(snipped)
```

```
linux : ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:43:15:fc brd ff:ff:ff:ff:ff:ff
```

```
linux : sudo lshw -class network
  *-network
       description: Ethernet controller
       product: Virtio network device
       vendor: Red Hat, Inc.
       (snipped)
```

```
linux : nmcli connection show
NAME                 UUID                                  TYPE      DEVICE
eth0                 5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03  ethernet  eth0
eth1                 9c92fad9-6ecb-3e6c-eb4d-8a47c6f50c04  ethernet  eth1
lo                   bc208bf8-d19f-4989-8ab5-2342d57a3e84  loopback  lo
```

Setting an IP address and Netmask

To set an IP address more an more distribution use the NetworkManager service[2].

[2] ifcfg-<nic> files are outdated.

On Red Hat based distributions you can still use the interface scripts in /etc/sysconfig/network-scripts.

Here is a simple example for the eth1 device.

#filename: /etc/sysconfig/network-scripts/ifcfg-eth1	
DEVICE=eth1	the name of the network interface card
BOOTPROTO=static	can be static, none, or dhcp
ONBOOT=yes	interface is configured at boot time
NETMASK=255.255.255.0	first three bytes form the network part
IPADDR=192.168.5.10	the ip address
GATEWAY=192.168.5.254	sets the default gateway for the interface

You can also use the **`nmcli`** tool or **`nmtui`**, to edit connections.

The following command will set the ip address of `eth1` to `192.168.5.20`

```
linux : sudo nmcli con modify eth1 ipv4.addresses 192.168.5.20/24
```

ip address - command

With the **`ip address`** or **`ip a`** command, you can list all ip addresses on your system.

```
linux : ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
(snipped)
```

ifconfig - command

The ifconfig command is deprecated, but you can still install it and use it. It is part of the `net-tools` package.

Many admins still like to use this tool.

The `ifconfig` command will list all ip addresses or you can temporarily change the ip address of an interface on the fly.

In the following example the ip address of eth1 is listed, then modified and listed again.

The modification will not be permanent. If you want to make it permanent you either use nmcli, nmtui of the ifcfg-eth1 file in /etc/sysconfig/network-scripts.

```
linux : sudo ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.4.10  netmask 255.255.255.0  broadcast
192.167.4.255
      (snipped)

linux : sudo ifconfig eth1 192.168.5.100

linux : sudo ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.5.100  netmask 255.255.255.0  broadcast
192.168.5.255

Note: Setting the ip address with the ifconfig command is not
permanent. It will not survive a reboot.
```

DNS Nameserver

To easily configure the DNS servers, you can create use the /etc/resolv.conf file or add the DNS servers to the ifconfig-<nic> file in /etc/sysconfig/network-scripts. As a third option you can use the `nmcli` command.

```
linux : cat /etc/resolv.conf
nameserver 8.8.8.8


linux : cat /etc/sysconfig/network-scripts/ifcfg-eth0
(snipped)
DNS1=8.8.8.8

linux : nmcli con modify eth1 +ipv4.dns 8.8.4.4
linux : nmcli con show eth1 | grep -i ipv4.dns:
ipv4.dns: 8.8.8.8,8.8.4.4
```

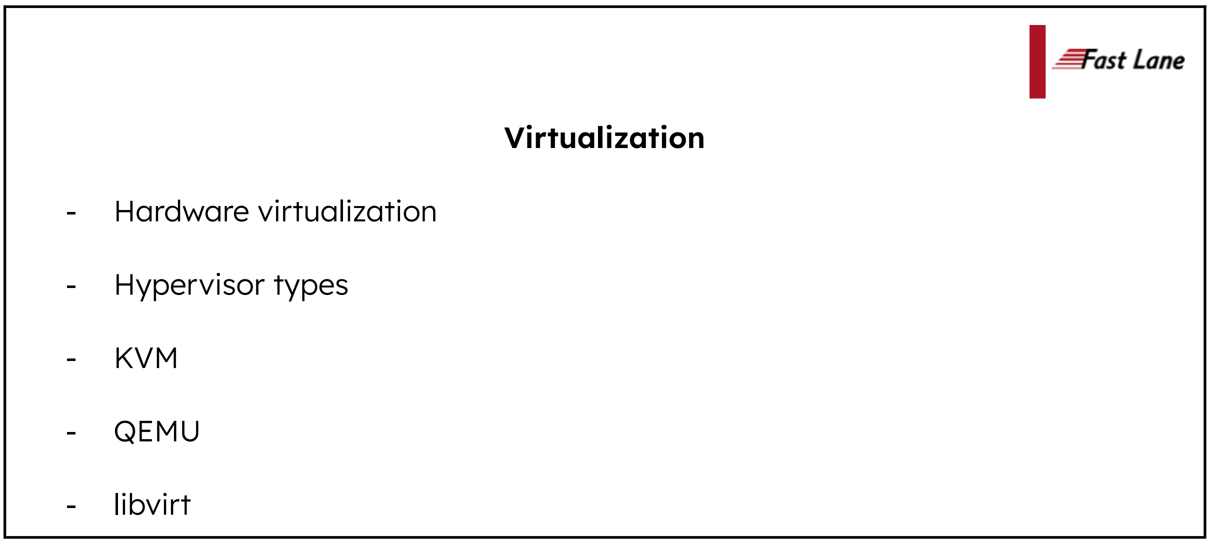
Basic Networking - Exercises

	Exercise 1
	In this exercise you will work with links or nics.
1-1	List your network interface cards hint: use the `ifconfig` command.
	List your network interface cards hint: use the nmcli command.
	List all properties of the eth0 interface hint: use the `ethtool` command.

	Exercise 2
	In this exercise you will list your ip configuration
1-1	List the ip address of eth0 hint: use the `ifconfig` command.
	List your network interface cards hint: use the nmcli command.
	List all properties of the eth0 interface hint: use the `ethtool` command.

	Exercise 3
	In this exercise you will set your ip configuration for eth1
1-1	Set a new static address for the eth1 interface.

	hint: use the **`nmcli con modify eth1 ipv4.addresses`** command.
	Add DNS server 1.1.1.1 to the eth1 connection hint: use the **`nmcli con modify eth1 +ipv4.dns`** command



Virtualization

- Hardware virtualization
- Hypervisor types
- KVM
- QEMU
- libvirt

9. Virtualization

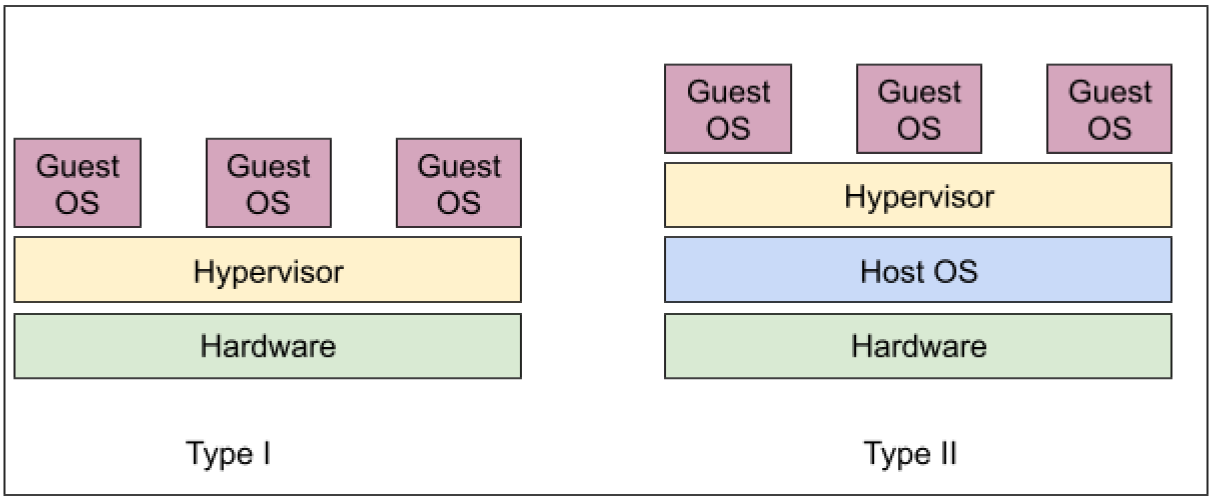
Virtualization is a technology that allows multiple operating systems or applications to run on a single physical machine. It creates a virtual (rather than actual) version of a resource, such as a server, storage device, network, or operating system. This virtualized environment enables greater efficiency and flexibility in managing and utilizing computing resources.

There are several types of virtualization, like Hardware Virtualization, Desktop Virtualization, Application Virtualization, Network Virtualization and Storage Virtualization. In this introduction to Virtualization we will focus on Hardware Virtualization.

Hardware Virtualization

Hardware virtualization provides hardware support to on or more guest Operating Systems. Two important kinds of hardware virtualization are Type I and Type II hypervisors.

- Type I – Runs on top of the hardware (bare metal hypervisor)
  - VMware vSphere with ESXi, KVM, Microsoft Hyper-V, Xen
- Type II – Runs on top of an installed Operating System.
  - VMWare Workstation, Oracle Virtual Box, Parallels.

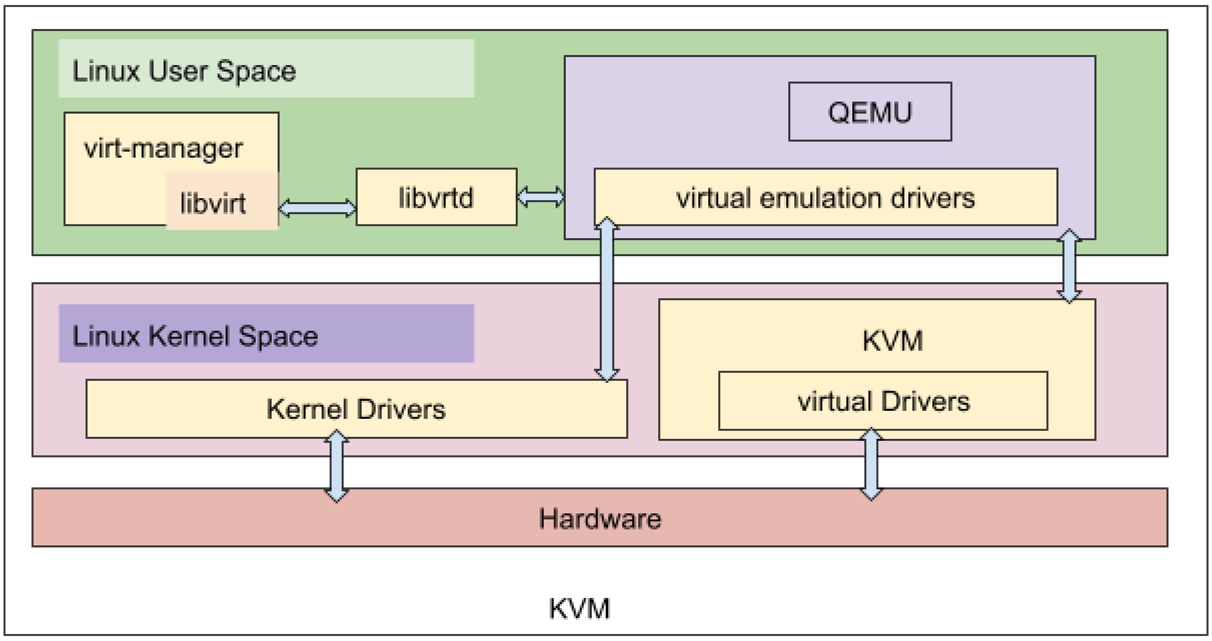


KVM

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions.

With KVM you can run unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc.

The workings of KVM is a complex structure of software relations.



This module has no exercises but a demonstration.