

# Linux System Administration

PM-TRTNB219

Zsolt Schäffer, PTE-MIK

# Preamble

Lecture 1

Chapter 1

## Course details

## Contact information

- Code: PM-TRTNB219
- Tract: 2x90 min every week
- Location: computer lab A101
- Course leader:  
Prof. Dr. Péter Iványi  
office B140
- Lecturer:  
Zsolt Schäffer  
office B103  
[schzsolt.pmmik@gmail.com](mailto:schzsolt.pmmik@gmail.com)

# Objectives

- Conceive an approach of to how administer Unix-like systems
- Practical knowledge of general administration of Linux based servers (setting up networking and users; providing database, web and network services, virtualization basics, etc.)
- Pick up synthesized knowledge to troubleshoot service issues on your own
- Preparation for the optional Red Hat Certified System Administrator and Red Hat Certified Engineer (ex200, ex300)

# Requirements to apply

- Accomplishment of course 'Operating Systems' (PM-TRTNB230)
  - Basic knowledge of basic shell command and concepts (e.g. cat, echo, redirects, envir. vars)
  - Knowledge of file system manipulation commands (eg. ls, cp, mv, rm, chown, chmod, etc.)
  - Being familiar with at least one console mode text editor (vi is standard)
  - Knowledge of basic Unix tools, like: more, sort, grep, mount, etc.)

# Requirements to pass

- Participate on **at least 70%** of lectures and lab practices
- There will be about 5 short MCSA tests at the start of randomly chosen lectures throughout the semester. Achieve an average of **at least 65%** by the end of the course
- A practice exam will take place on the last lecture of the semester. You'll have to set-up or correct the settings of a prepared virtual PC in accordance to written instructions. Achieve **at least 50%** on this final exam
- Both types of test can be repeated separately if necessary (but only 2 times at most in a single semester). Supplement exam appointments will be provided after the study period.

# What do Linux server admins do?

- Set up services provided to other nodes on the company/institutional network or to the public. (Like file storage, block storage, e-mail, database, web applications...)

This is achieved via configuring DEAMONS  
→ lectures 9-11

This is the most 'in sight' part, the thing visible to users/customers.

Several other skills are required towards this:

- Configure basic system settings. Set up a suiting environment for daemons. (Install packages, add users, configure network, set up hard disks and file systems)

→ lectures 3-5,7

# What do Linux server admins do?

- Schedule maintenance and backup tasks  
→ lecture 8
- Harden security to protect the running environment to make the provided services more robust  
→ lecture 4
- Troubleshoot boot and service issues, analyze logs  
→ lecture 6,8
- Set up virtualization, uniform environments, mass deployment (only partly covered in this course)  
→ lecture 12

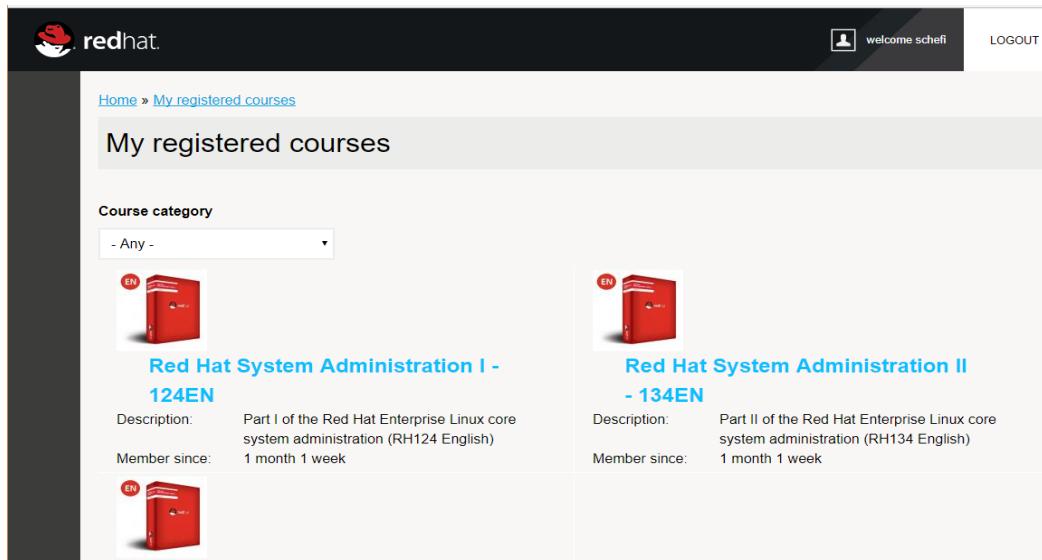
# Intro to the Red Hat Lab environment

Lecture 1

Chapter 2

# Sign-up to an RHLearn account

- Open <https://rhlearn.gilmore.ca/> in your browser
- Create a new user account. (License key will be provided to your e-mail)
- Once logged in, select My Courses on the Dashboard
- All learning material will be available to you for 6 months after redeeming the key



The screenshot shows the 'My registered courses' section of the Red Hat Learn platform. The top navigation bar includes the Red Hat logo, a 'welcome' message, and a 'LOGOUT' link. The main content area is titled 'My registered courses' and features a 'Course category' dropdown set to '- Any -'. Below this, two course entries are listed:

Course	Description	Member since
<b>Red Hat System Administration I - 124EN</b>	Part I of the Red Hat Enterprise Linux core system administration (RH124 English)	1 month 1 week
<b>Red Hat System Administration II - 134EN</b>	Part II of the Red Hat Enterprise Linux core system administration (RH134 English)	1 month 1 week

# Classroom Lab environment

- The lab consists of Classroom server and several student computers.
- Each student computer is a host for two virtual machines. Hands on practice experience throughout the course happens by manipulating these virtual computers. You'll have root access to them.
- The virtual machines are called serverX and desktopX, where X is a number denoting the seat you are in.
- The virtualization host can only be accessed as a restricted user, but you can still use the web browser, take notes, etc...
- The virtual machines are snapshoted, they can be reset to a ready made starting condition if something goes really wrong.

# Logical layout

- Each student is provided with both a virtual server and client computer in order to be able to set up and test network services throughout the course.
- Some tasks only require one the virtual computers, some need both.
- The IP address of your personal virtual **server** is **172.25.X.11** where X denotes your seat number. Hostname resolving to this IP is **serverX.example.com**
- The IP address of your personal virtual **desktop** is **172.25.X.10** where X denotes your seat number. Hostname **desktopX.example.com** resolves to this IP address.

# Controlling the virtual machines

- You can access the console of the virtual computers with clicking on the icon on your hosts dekstop. This opens up a VNC like connection (spice protocol) to the screen of the VM.
- Clicking the icon starts the VM if it isn't already running.
- You can control the VM through its graphical console (spice window) or using the rht-vmctl command on the host computer.
- You can restore the VMs to the starting state using the command line.

# Controlling the virtual machines

- Also consider the following rht-vmctl commands:
  - start, stop
  - reset, save
  - fullreset
  - view

Host shell command line

```
# rht-vmctl start server  
  
# rht-vmctl start desktop  
  
# rht-vmctl start all  
  
# rht-vmctl reset server
```

# Reminder of some general OS and basic Linux concepts

Lecture 1

Chapter 3

# In this chapter

- Kernel space, user space, system call, driver, daemon
- Scheduler, task, process, thread, interrupt, signal
- Standard streams, file descriptor, redirects
- “Everything is a file file” - file-centric concept of \*nix
- Terminal, console, teletype, pty, command line, shell
- How to get help?

# Kernel space

- Kernel space and user space separation is a way of sandboxing applications, in order to prevent them from doing globally harmful things like crashing the machine.
- CPUs usually have hard-wired features to support privilege distinction. Intel x86 CPUs have 4 hardware privilege rings. Linux kernel runs in ring 0. User applications run in ring 3. 1 and 2 is not used in Linux.
- Code running in kernel space has access to everything, including the whole range of physical memory and all bus I/O addresses, meaning it can send and receive data directly to any hardware.
- Kernel code consist of the kernel binary itself and several loadable extensions, called kernel modules.

# Kernel space

- A specific functionality of the kernel can usually be set to 3 different states at compile time: disabled, enabled, or module. Not everything is selectable as a module.
- Modules can be loaded or unloaded at runtime from binary files called kernel object (\*.ko). They are stored under /lib/modules/linux-[kernel version] directory.
- You can compile your own kernel module against the source code of the very specific version of the desired kernel. Kernel object files are usually not compatible across different kernel versions.
- Kernel objects can implement device drivers, filesystem drivers, cryptography functions, and a lot more, like intermediary layers between other kernel modules. There is a dependency tree involved.

# User space

- User space applications are all the other software code that does not belong to the kernel. This isn't restricted to just applications with a user interface (like a spreadsheet editor), but a lot more.
- An executing instance of a program is called a process. Each user space process has its own virtual address space, separated from all other processes.
- All processes have a unique numerical identifier (PID).
- All processes have a user ID (who own the process) and also an SELinux context. (More on this in lecture 3 and 4)
- The virtual address space of each process is mapped to physical RAM ranges (usually non-contiguous) and/or swap space. This is also done with the help of hard-wired CPU features (Memory Management Unit). Only kernel can manipulate the MMU mappings.

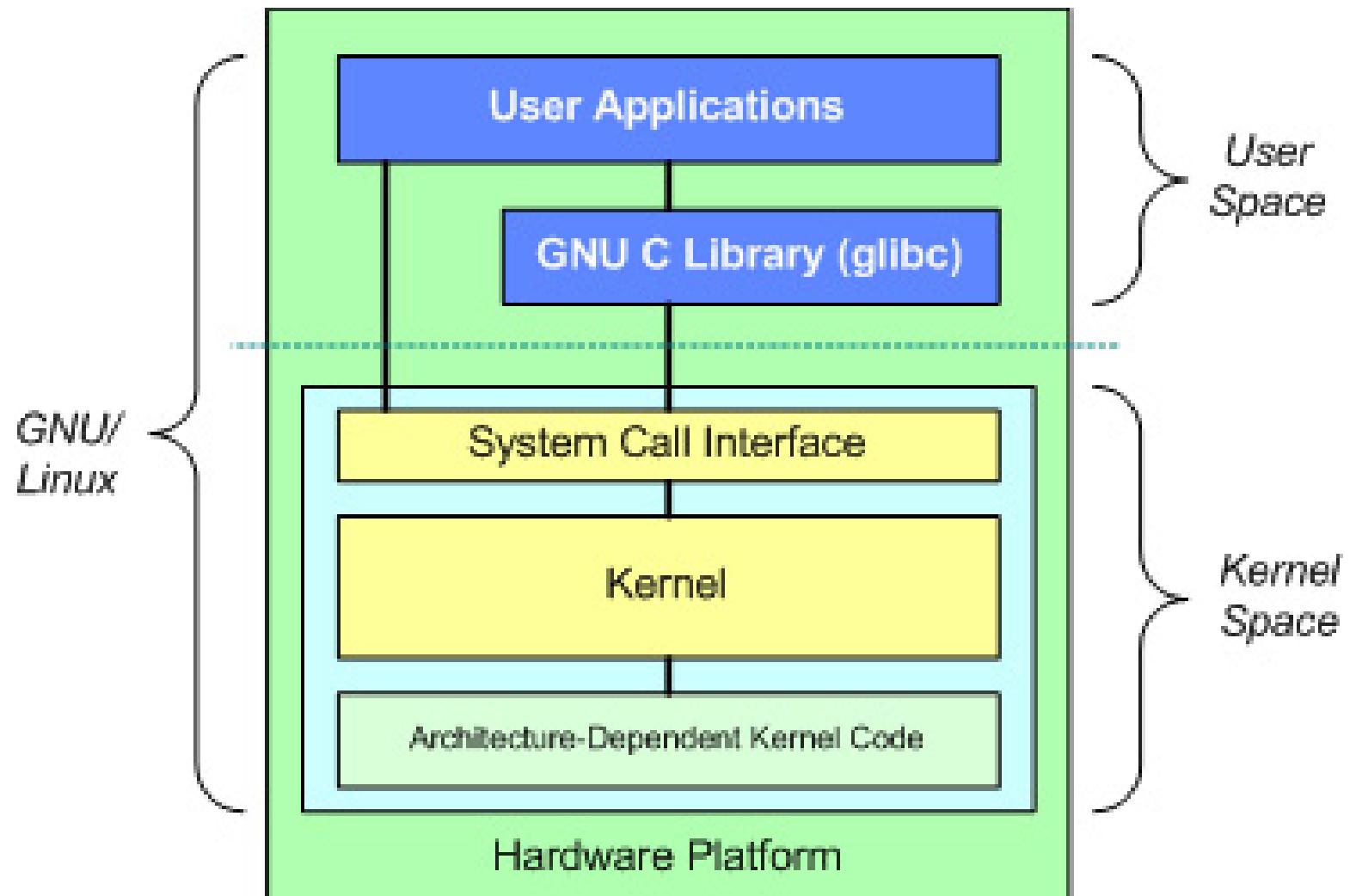
# Scheduler, processes

- Userland processes don't have access to I/O buses either. Hardware is access through device files and system calls.
- Processes don't get unrestricted access to CPU resources either. Time slices are given to them by the kernel's scheduler. It decides which process to give the next time slice to, based on a configurable algorithm.
- Switching between processes involves saving and loading the execution state of the CPU several times a second (context switching). This is done by the kernel.
- Kernel itself doesn't behave like a process, it doesn't get scheduled. After init is complete it either just services interrupts or runs system calls upon userspace requests.
- More on schedulers, tasks, processes, threads, interrupts, signals will be discussed in lecture 4.

# System calls

- System calls are requests in a Unix-like operating system by an active process for a service performed by the kernel, such as input/output (I/O) or process creation, opening of a file, memory allocation, etc.)
- Kernel version can change frequently in a system. The stable interface of requesting kernel functions to be performed is the standard C library (libc).
- Libc versions are widely compatible (within major versions), even vendor inter-operability is possible (glibc ↔ eglibc are ABI compatible, uclibc ↔ glibc are only API compatible)
- E.g.: calling the printf() function from a program works on a wide range of systems without recompiling. Several kernel layers are involved, we get expected results independent of actual output device. (serial port, VGA console, SSH session or X11 terminal window)

# Fundamental architecture of Linux



<https://www.ibm.com/developerworks/library/l-linux-kernel/figure2.jpg>

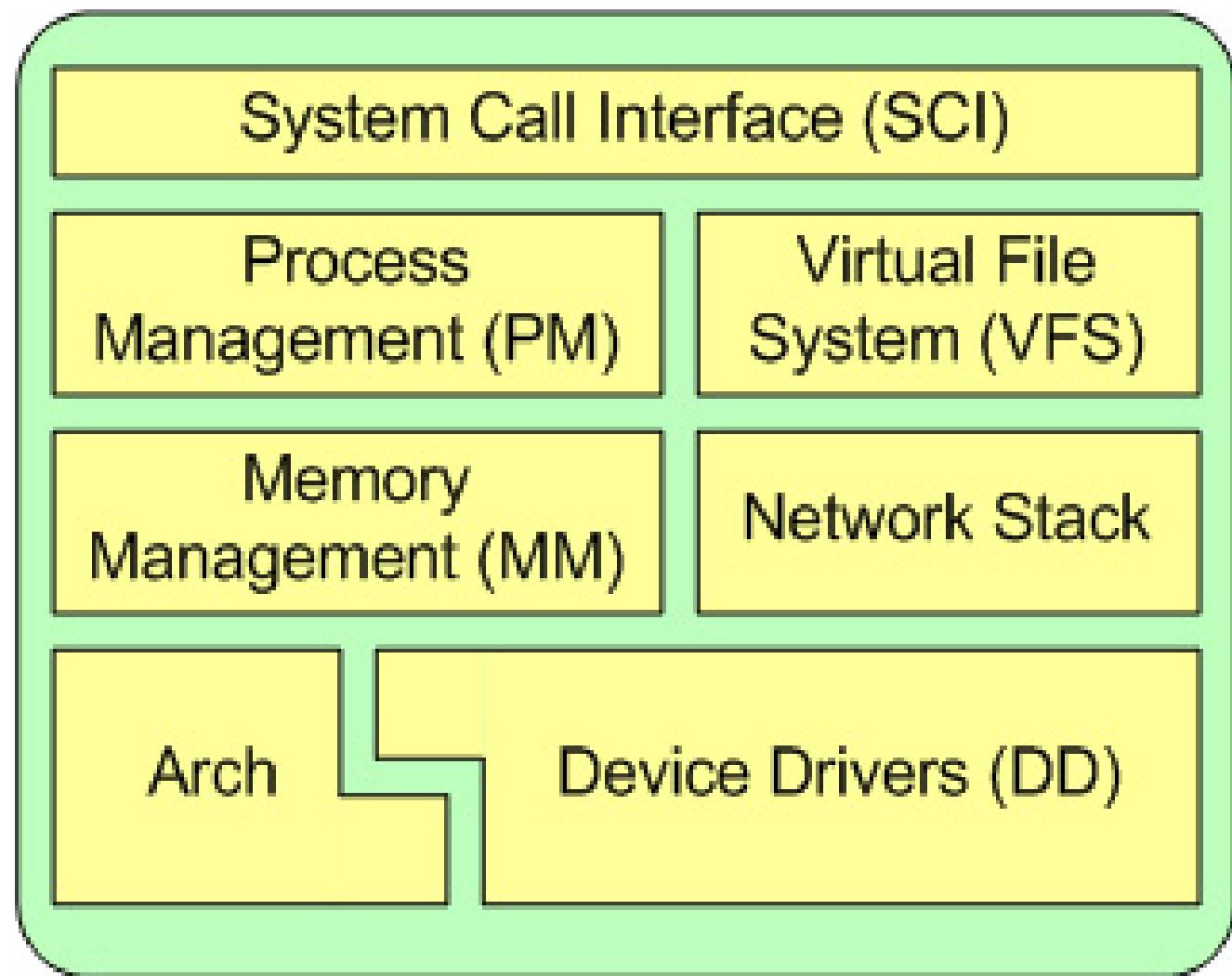
# Hardware devices

- Access to hardware for userland processes is provided via device files, usually located in the /dev directory or using the sysfs filesystem. (More on this in lecture 5)
- For e.g. reading input from a serial port is performed by opening the /dev/ttyS1 device, then performing an tcsetattr() call to set baud rate and other serial preferences. After that the port can be written or read just like a text file. E.g. `read(12,buffer*,10);` This line of C code reads 10 bytes from file ID 12 (the serial device in this example) to an array pointed by buffer\* pointer.
- You can get snap images from a webcam device or read the orientation sensor of your handheld device in a similar way.
- E.g. turning an LED on or off in a Raspberry Pi is done by writing a 0 or 1 to a file like `/sys/class/gpio/gpio17/value`

# Drivers

- Simply put drivers are kernel objects that configure, initialize and perform raw I/O to hardware devices and represent the device as a file to userspace programs.
- A driver is programmed by implementing interrupt handlers and file operation functions in C code. (like open, seek, read, ioctl, etc.) These functions are compiled against a specific kernel source code into a ko file.
- The hooks of your implemented functions are imported by the kernel and executed appropriately every time a userspace process wants to access the device.
- The user process only performs file operations, subject to standard filesystem permissions. Actual bus I/O is performed in kernel space by functions compiled into the kernel object. The application makes the request by asking the kernel to call the read() or other function implemented in the given device driver.

# Major subsystems of the Linux kernel



<https://www.ibm.com/developerworks/library/l-linux-kernel/figure3.jpg>

# Daemons



- Daemons are userspace processes. Just like the ones discussed earlier.
- The special thing is that they don't interact with a user directly. Instead their input (rules of how they should operate) are read from a config file and their output is written to a log. Other than that, daemons only communicate to other processes on the same host or over the network.
- Therefore they are often referenced as background processes. This is not quite accurate, since any other process can be put in the background too.
- Usually both the config and log contains only human readable text. (This is strong Unix concept)
- The names originates in ancient Greek, it means "personal protecting spirit", like a guardian angel.

# Daemons

- Daemons are strictly focused on doing one very specific (sometimes even simplistic) task, but with the goal of doing it most proficiently. (This is a strong Unix concept.)
- Daemons are the most prevalent way of providing automated services. (For e.g. serving .jpg and .html files from a web server or providing network addresses in reply to DHCP requests.)
- Hence daemons are a very important part of this subject. There is a daemon for every task somewhere.
- It is impossible to cover them all. What will be covered (through several examples) is the concept itself to configure services through text files and troubleshoot through logs. (More in lecture 9 and 10)

# Standard streams

- Remember the example of the `printf()` function. The application calling this function is completely unaware of the output devices. For e.g. it can be a graphic terminal on HDMI screen or a serial port on the same computer like a Raspberry Pi or a remote terminal through a network session. It still works in all cases without recompiling or modifying the a program.
- The kernel contains the required layers for a `printf()` call to become actual output.
- But how is this interfaced to the application? How is it standardized?
- As discussed earlier, applications do only file operations instead of directly manipulating the hardware.

# File descriptors

- When a process executes the `open()` syscall to open a file, the function (on success) returns a positive integer, this will be a file descriptor.
- This number uniquely identifies the file within the specific process which opened it. File descriptors are not globally unique, but process are sandboxed anyway. This info is public, it is exposed through `/proc/[PID]/fds` dirs.
- Every process gets three file descriptors opened upon their creation. No manual opening is necessary. This is handled by standard libc during application initialization.
- These file descriptors are the following:
  - 0 for standard input (`stdin`)
  - 1 for standard output (`stdout`)
  - 2 for standard error output (`stderr`)

# Standard streams

- So the printf() function actually only writes characters to a “text file”, specifically file nr. 1, which is standard output for all Unix processes. The kernel handles all the rest.
- The terminal handles line editing and can pass each “sentence” (finished with the Enter key) to the process. The process just basically request to read a line from the file called stdin.
- Streams can be redirected to/from disk files or other processes (even to other computers - with SSH). More on this in lecture 2.
- In correlation to the known Unix concept of an application doing only one thing but doing it efficiently, redirection makes it possible to solve complex problems with channeling data between simple building blocks on-the-fly.

# Everything is a file

- So far several perspectives have been shown, how the unifying concept of filesystem objects can work in Unix.
- The benefit is that a set of utilitys, APIs can be used on a wide range of resources. Documents, modems, keyboards, inter-process communication, even internal kernel variables and structures, etc... are represented as files.
- The following list of file types are known in Linux, these are all file system objects (FSO): regular files, directories (they are files too), UNIX domain sockets, named pipes, device node files, symlinks. Also have to mention hard links and virtual files.
- More on named pipes in lecture 2, directories, links and virtual files in lecture 5.

# Everything is a file

- A full list of file types with notion of first output field of the ls command, and the related command to create the object.
  - - regular file touch
  - d directory mkdir
  - c character device node mknod
  - b block device node mknod
  - p named pipe mkfifo
  - s socket socket() syscall
  - l symbolic link ln -s
  - - hard link ln
  - - virtual files (/proc, /sys) -

# Terminal, tty

- Terminal from an electronics point of view is an end point, where an external device is connected to the service/network.
- In the early days of Unix mostly mainframe computers where current. Multiple users connected to a single MF using dumb devices (consisting only of a keyboard and a screen or printer and a means of transcieving signals to/from the mainframe). They where called terminals.
- Terminal is synonymous with tty. It's the abreviation of TeleType, which is a sort of electromechanical typewriter with a long “wire”, capable of printing characters remotely corresponding to locally pressed keys. Original device is from 19th century.
- In Linux any character device can be a tty that can perform some ioctl commands and receive/transmit series of bytes, like an RS-232 serial port.



[http://www.harriscomm.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/u/t/uti-sp4425\\_1.jpg](http://www.harriscomm.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/u/t/uti-sp4425_1.jpg)

[https://c1.staticflickr.com/3/2553/3916969051\\_ecc076867e\\_b.jpg](https://c1.staticflickr.com/3/2553/3916969051_ecc076867e_b.jpg)



<https://library.ryerson.ca/asc/files/2013/01/img026.jpg>



[https://upload.wikimedia.org/wikipedia/commons/thumb/7/71/IBM\\_3277\\_Display.jpg/1024px-IBM\\_3277\\_Display.jpg](https://upload.wikimedia.org/wikipedia/commons/thumb/7/71/IBM_3277_Display.jpg/1024px-IBM_3277_Display.jpg)

# Terminal

- The terminal is where the user logs in, presses keys, and reads answers. Terminal handles printable I/O and control characters (like the arrow keys) one by one. No interpretation of commands is done by the terminal.
- A terminal usually has a buffer, contents can be scrolled back. (Shift + PgUp, Shift + PgDn on Linux)
- A terminal can have line editing features but not necessarily.
- Terminals are provided by the cooperation of several kernel layers, some of them closely bound to the hardware itself.
- A terminal can only support characters and key presses, not graphics.

# Shell

- A process usually communicates its standard in- and output through a terminal.
- The shell is a userspace process that also runs in a terminal. The shell is responsible for interpreting commands, providing an environment for execution of other processes and much more. Command history, TAB completion and such features are provided by the shell, not the terminal.
- The shell that executes text applications relates to the terminal like a window manager that executes graphical applications relates to the graphics canvas provided by the video card driver.
- The most common shell used today is bash. More on this in lecture 2.

# Console, vty

- A console is like a primary terminal connected directly to the system. For e.g. A BIOS based computer's console consist of the standard 101-key keyboard and the standard 80x25 character VGA text output screen. (This is just a text buffer, automatically rendered to characters on the screen by the VGA card itself.)
- /dev/console is a link to one of the terminal devices.
- A computer equipped with UEFI firmware has a standard graphical surface as output, called the uefifb. BIOS boxes can still use vesafb with fbcon (vtcon). This is a graphical surface but still a terminal, not a desktop.
- A Linux PC installs with 6 terminals by default. These are called virtual terminals and can be cycled with Ctrl+Alt+F1 to F6. Only one of them can display on the physical console at once (Be it vesa or text don't matter). A userspace process called getty is involved in cycling.

# Terminals

- List current user session with the **who** or **w** command.
- Also consider the **who -a** switch.

## Shell command line

```
# w
 10:22:47 up 2 days, 22:56, 2 users, load average:
0,04, 0,06, 0,05
USER     TTY      LOGIN@    IDLE    JCPU   PCPU WHAT
root     pts/0    10:21    0.00s  0.29s  0.02s w
root     pts/1    10:21    1:08   0.25s  0.25s -bash
```

- The file concept also applies to terminals. You can write to other users terminals, just like writing to text files.

## Shell command line

```
# echo -e 'Hello\n' > /dev/pts/1
```

- This will produce a line of text on pts/1 terminal.

# Terminals

- To enable a mouse cursor in text console, type the following:

Shell command line

```
# systemctl start gpm
```

- To select text press the left mouse button and drag the mouse.
- To paste text in the same or another console, press the middle button.
- The right button is used to extend the selection.
- You can also use **ctrl + ins** and **shift + ins** to use the clipboard (copy/paste).

# Pty

- A pty (pseudo terminal) acts just like a normal terminal to the process that runs inside it, but it is not connected to a piece of hardware. Rather it is connected to a software layer provided by another process, like **screen**.
- One example would be an **X11** terminal emulator window or a **Gnome Terminal**, which is a user space application providing a graphical window with a scrollbar, selectable font and clipboard functions. If a shell is started in this pseudo terminal, it will not be relevant to it, since a pty acts just like a tty.
- An other example would be an **ssh** session, when managing a remote server through a shell. When logging in to a remote **SSH** server, it would create a pseudo terminal for a new shell process. Then all terminal I/O will be transferred to/from the **ssh** daemon through a network tunnel from/to the local **ssh** process.

# How to get help?

- Try executing one of these commands

- man
  - info
  - help

- Search in man pages for specific keywords

Shell command line

```
# man -k keyword
```

- Man pages are also available on Google. Try: “man ls”
- Search for exact error message on Google. Try something like this: “PTY allocation request failed on channel 0”

# How to get help?

- Read text files in /usr/share/doc directory
- For tutorials look at <http://www.howtoforge.com>
- Search the RedHat documentation at  
<https://access.redhat.com/documentation/en/red-hat-enterprise-linux/>
- Use the redhat-support-tool command

# Recommended reading

Lecture 1

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH124 Chapter 3 (getting help)
- RH124 Chapter 1 (command line)
- RH124 Chapter 2 (filesystem layout, file manipulation)
- RH124 Chapter 4 (redirect, cat, vim)
- RH134 Chapter 2 (regexp, grep)
- RH134 Chapter 3 (vim)

Use your `rhlearn.gilmore.ca` login to access the learning materials.

# Thank you for your attention!

Lecture 1, PM-TRTNB219, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB219

Zsolt Schäffer, PTE-MIK

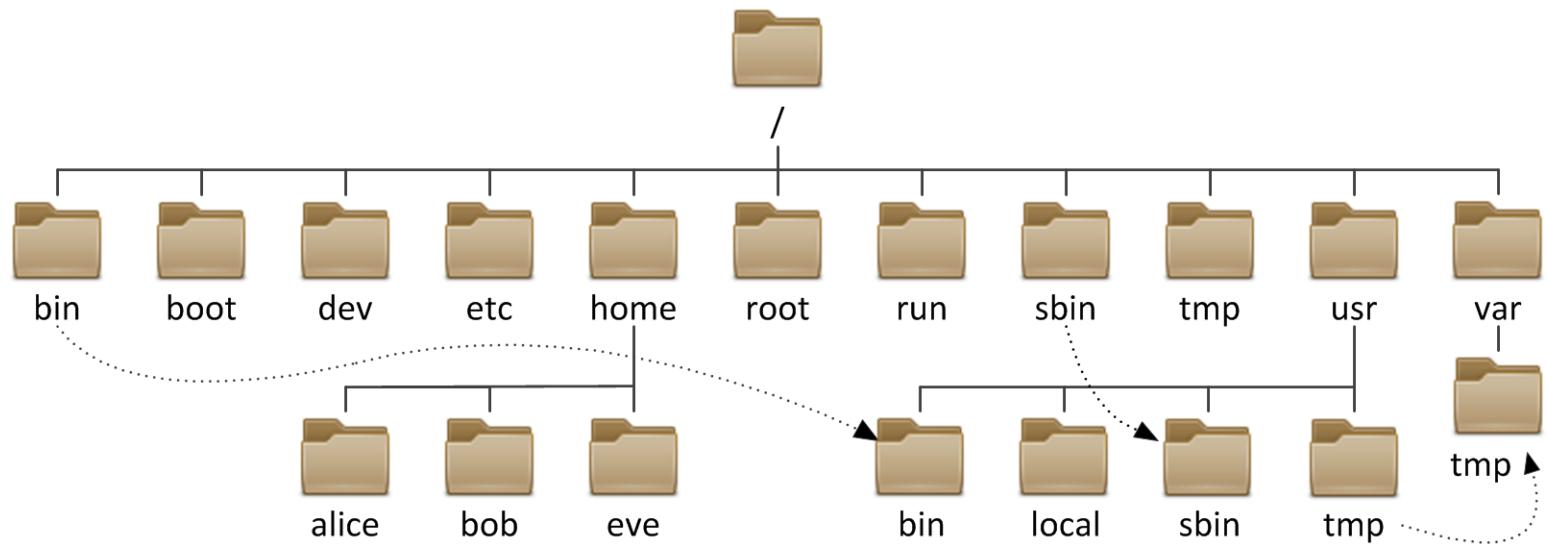
# Managing files

Lecture 2

Chapter 1

# Filesystem hierarchy

- Linux systems have only one filesystem structure, with one single root point, denoted with a slash (/) called root directory.
- Additional filesystems may be attached to a (sub)directory of the structure. This called mounting.
- The standard filesystem structure is shown on the figure. The arrows represent symlinks. More on filesystems in lecture 5.



# Directory structure

- For an explanation of each directory on the figure refer to **RH124 CH 2.1**. As a supplement to the RH learning material some further directories are described here.
  - /proc contains information about all the running processes. It is a virtual filesystem, its contents is determined by the running state of the kernel, it is not stored on disk.
  - /sys provides information about devices, buses, drivers. Also provides an API, most of the files are writable in this directory. Changing files in this directory affects how the kernel or certain kernel modules behave or operate.
  - /dev nowadays isn't stored on disk. It is a tmpfs managed by the udev daemon. It contains the device files for hardware items that are present on the system.
  - /lib contains shared libraries (.so files) and kernel modules (.ko files), along with some accounting information related to libraries.
  - /opt is where optional software is installed. Custom installers outside of the built-in package manager are encouraged to write into this directory instead of /bin or /sbin
  - /mnt is the default directory for mounting additional (e.g. removable or network) drives in several distros. RHEL uses /run/media for this purpose.

# Navigating paths

- A file or directory is explicitly specified by giving its full name and path (location inside the directory structure).
- Filesystem object names are case sensitive in Linux.
- If the location specifier starts with a / (slash), it is an absolute path. E.g. /home/jane/Documents/cv.pdf
- Otherwise its a relative path, which is meant to be interpreted relative to the current working directory. E.g. Pictures/photo1.jpg if current directory is /home/jane → this equals to the absolute path of /home/jane/Pictures/photo1.jpg
- There is shortcut (substitution) for the currently logged in users home directory: ~ (tilde) For e.g. ~/Documents
- The shortcut to a specific users home directory looks like this: ~adam/some.file

# Navigating paths

- The current working directory can be printed with the **pwd** command.
- The current working directory can be changed with the **cd** command.
- The current working directory is referenced with a **.** (single period), the parent directory with **..** (double period)
- A directory can be listed with the **ls** command. Also take note of the **-l -a -R -Z -h** switches.
- Refer to **RH124 CH2.3.** for further details.
  
- You can use the **file** command to get information about a file's type, content and format. (Very useful!)

# Managing files and directories

## create and remove

- You can create empty files with the **touch** command and empty directories with the **mkdir** command. (note **-p** switch)
- You can remove a file using the **rm** command, to remove a directory with everything inside use **rm -r** command. (note the **-f** switch)
- The **-r** switch usually means recursive. Everything in a directory and its sub-directories and their sub-directories, etc... is considered to be the subject of the operation if this is used. E.g. **ls -r; cp -r; rm -r**
- The **-f** switch means force. Don't ask, just do the operation. Use careful with **rm**!

# Managing files and directories

## copy and move

- You can copy files or directories with the **cp** command.
- You can rename/move files or directories with **mv**.
- Consider the source and destination types and number of arguments described in **RH124 CH2.5**.
- You can expect different behavior depending whether object2 is an existing file, an existing directory or a non-existing object.

### Shell command line

```
# mv object1 object2
# cp object1 object2
# cp -r dir1 object2
# mv object1 object2 object3 dir1
```

# Practice

- Complete lab practice in **RH124 CH 2.6.**
- Now do the same the smart way!

## Shell command line

```
# mkdir Music Videos Pictures
# touch {Music/song{1..6}.mp3,Pictures/snap{1..6}.jpg,Videos/film{1..6}.avi}
# mkdir friends family work
# cp **/*[12].{mp3,jpg,avi} friends
# cp **/*[34].{mp3,jpg,avi} family
# cp **/*[56].{mp3,jpg,avi} work
```

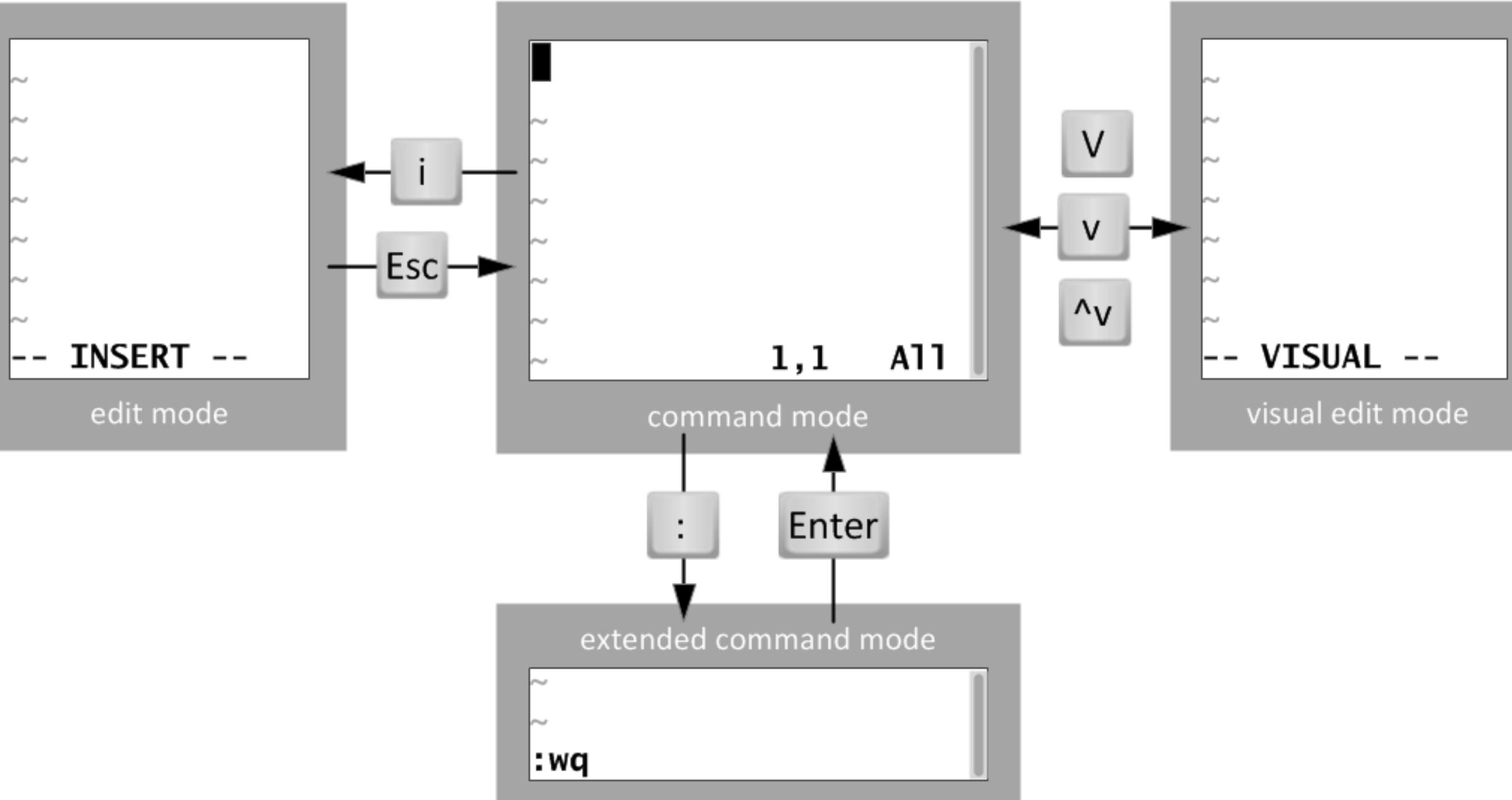
- More on filename globbing in **RH124 CH 2.7.** through **CH 2.9.**

# Viewing files

- The most simple way to show a file's contents is to use the **cat** command. This prints a file to the standard output.
- This command is usually applied when using redirects. (More on redirect later in this lecture)
- When reading on screen you can use the **more** or **less** commands to produce a scroll-able output.
- Remember that less is more :)
- Less produces a sideways and backwards scrollable output. You can use the arrow keys.
- More can only step forward with the output: one line (enter) or one page at a time (space).
- Quitting is possible with the q key.

# Editing files

- The primary text editor present in all Unix systems is vi.
- vi has four operation modes: the command mode the edit mode, the visual edit mode, and the extended command mode.
- In command mode (this is the default) key presses are interpreted as instructions for the editor program. Extended commands start by typing a : (colon).
- In editor mode key presses are directed into file contents (actual typing).
- You can always return to command mode with Esc key.
- Visual mode is started with v or V or ctrl+v and terminates with the same key-press. You can do text selection in visual mode.



# Most important vi commands

- In command mode:
  - press i for insert mode.
  - press a for append mode
  - type dd to delete (cut) whole line
  - type yy to copy whole line
  - type p for paste after cursor, P for pasting before cursor
- Advanced commands:
  - :w to save file
  - :q to quit
  - :q! to quit without saving file
  - :wq to save and quit

# Selecting ranges in vim

- Position the cursor over the starting position.
- Activate char, line or block selection with v, V or **ctrl+v**
- Position the cursor to the ending position.
- Now you can cut with d, copy with y, and paste with p or P.
- You can insert with shift+I, append with shift+A. Input is applied to every selected line.
- Return to command mode by pressing the same key you entered with.
- More on vi in **RH124 CH 4.3-4.4.** and **RH134 CH3.**
- Complete lab practice **RH124 CH 4.7.**

# Finding files

- An easy to use and fast utility to search for files is **locate**. It is speedy because it doesn't search the filesystem itself, rather it looks in a database.
- Mentioned database has to be updated regularly, using the **updatedb** command, to maintain consistency.

## Shell command line

```
# locate *dir2*
/home/user/Download/glibc-2.21/dirent/tst-fopendir2.c
/home/user/Download/glibc-2.21/manual/exapmles/dir2.c
```

- The universal primordial standard tool is **find**.
- Find can be used with several filtering options, not just filename. Filter options include for e.g.: file type (refer to lecture 1, slide 32.), date, permissions, owner, size, etc.

# Finding files

- More on find in **RH124 CH4.1**. Also consider the following examples:
- To find all socket type files under /var directory with it's name containing sql and not owned by adam, type the following.

Shell command line

```
# find /var -type s -name "*sql*" ! -user adam
```

- To find and delete all large and old files owned by bob in external drive (with size larger than 100MB and accessed more then a month ago) type the following.

Shell command line

```
# find /mnt/external -type f -user bob -size +100M -atime +31 -exec rm {} ;
```

# Introduction to BASH shell

Lecture 2

Chapter 2

# Shell

- As discussed earlier in lecture 1 a shell provides an execution environment with variables, starts other processes, interprets user commands, provides advanced command editing features (like history and TAB completion) and much more.
- A shell can also be pre-programmed to execute a series of commands based on decisions. This is called shell scripting.
- Note the difference between interpreted (scripted) and compiled binary programs (ELF vs. text). There is a wide range of scripting languages available in \*nix environments. Note that text files can also be executed in Linux, not just binaries. (specify interpreter with #!)
- There is plethora of shells for Linux. Further on in this course only BASH (Bourne-again shell) will be discussed, which is the most popular one to date.

# Shell basics

- One of the most handy features is the TAB completion. It can complete filenames, commands and with some tweaks even command switches and arguments in order to reduce the time needed for typing.
- Pressing TAB once completes an already typed fragment of a filename or command to the point where it is not ambiguous.
- Pressing TAB twice quickly will display all the possible endings for the currently typed chunk.
- In case output would not fit the screen it is automatically piped through less.
- You can roll back and edit the previously entered commands with the arrow keys. This is called command history. A search in history can be initiated with `ctrl+r`.

# Shell variables

- Bash handles only text variables. Numbers are also stored as a string of characters. Variables don't have to be declared (exceptions!), they can be used simply by giving them a value. E.g. CARBRAND="Toyota"
- Later on you can refer to the value of the variable with a \$ sign. E.g. \$CARBRAND
- Variables are evaluated through a mechanism called substitution. The shell, when parsing any input will look for \$ signs and know it has to do something before executing the command. Basically it will replace one string with another, hence called substitution. Works like the macro pre-processor in a C compiler.

Shell command line

```
# echo "I drive a $CARBRAND"  
I drive a Toyota
```

# Shell variables

- The \* character discussed in file globbing is actually also a substitution. Same as ~ for home directories. The \* (star) character is substituted with a list of all files, then the command is executed with a long list of arguments. E.g. `mv /home/david/old/* /mnt/archive`
- You can specify multi word strings or strings with special characters by enclosing them in " (double quotes) E.g. `FANCY_VAR="There are many *s on the sky"`. This makes bash disregard any substitution other then \$VARIABLE likes.
- You can make bash disregard even variable substitution when enclosing something in ' (single quotes). E.g. `echo 'I have got 10 $ in my pocket.'`
- You can escape single characters with \ (backslash). E.g. `"I've got 10 \$ in my pocket"`

# Shell variables

- Substitution works inside any string (except when using single quotes). E.g. COUNT=3 VAR="There are \$COUNT \*s on the sky"; echo \$VAR
- The scope (lifespan) of variables can be one single line, one whole script, one user session or inheritable to all sub processes (exporting). Practice with the following:

Shell command line

```
# NAME=ALICE echo "Hello $NAME"
Hello
# NAME=ALICE bash -c echo "Hello $NAME"
Hello Alice
# NAME=Alice
# echo "Hello $NAME"
Hello Alice
# export NAME [then check in subshell]
```

# Shell variables

- Substitution even works for executing whole commands and taking their output. This is done with a subshell. E.g. `DAYOFWEEK=$(date '+%A');` `echo "It's $DAYOFWEEK today"`. Enclosing a command in ` (backtick) is equivalent to `$()`
- You can print all the variables with the **env** command. You can use the **unset** command to delete a variable.
- Besides string variables and indexed arrays Bash 4 also has associative arrays. Refer to **man declare** for more.
- There are some special environment variables directly affecting the shell. E.g.: `$TERM`, `$PS1`, `$PATH`, `$LANG`

Shell command line

```
# export PS1="\u@\h \w> "
adam@lab12 /etc/mail>
```

# Interactive shells

- There are two types of shells. The interactive shell is what the user is interfacing with, when typing commands. This is also called a login shell. This is what we have been using so far.
- The non-interactive or non-login shell is also a userspace process, it is started when executing a shell script. It reads the script, interpreting it line by line and performs accordingly. It doesn't interact with a user directly, it doesn't have a prompt.
- When a bash process is starting up it automatically executes the `/etc/bashrc` and `~/.bashrc` scripts.
- A login shell in addition to the former files also executes `/etc/profile` and `~/.bash_profile` scripts.
- This is where the environment variables such as `$PATH` are initialized to a sane value.

# Shell scripting basics

- Use your newly acquired text editor skills to create the following file:

```
/home/student/script1.sh
```

```
#!/bin/bash
SAVE_IFS=$IFS
IFS=" "
echo "I got $# arguments. I'll print them back one
at a time, until I run out or reach \"stop\"";echo
for argument in $*;do
    echo $argument
    if [ $argument = "stop" ]; then
        echo "I reached stop"; break
    fi
done
IFS=$SAVE_IFS
echo; echo "Goodbye"
```

- Now make it executable with **chmod a+x script1.sh**

# Shell scripting basics

- Some special variables:
  - \$IFS – internal field separator for processing lists
  - \$? – the return value of the last command (numeric)
  - \$# – the number of arguments
  - \$0 – the zero-th argument (what was the script called?)
  - \$1 \$2 \$3 ... \$9 – the first, second,... argument
  - \$\* – all the arguments except \$0
- Bash scripts can have functions. Function can also have arguments.

```
get_list_backups() {  
    evaluate $1...  
}
```

# Shell scripting basics

## test condition

- `if test "$V1" = "$V2"`
- `if [ "$V1" = "$V2" ]`  
Both are equal, `test` and `[` are userspace programs returning either zero or non-zero depending on expression that follows. Note that spaces are required.
- Some test examples:
  - `if [ -f path_to_file ]` – test if parameter is a file
  - `if [ -d path_to_file ]` – test if parameter is a dir
  - `if [ -e path_to_file ]` – test if parameter is existing
  - `if [ -z $VAR ]` – test if VAR is defined and not empty
  - `if [ $VAR1 -eq $VAR2 ]` – test if numbers are equal
  - `if [ $VAR1 = $VAR2 ]` – test if strings are the same

# Shell scripting basics

## test condition

- Also consider other test like -z -n -gt -ne -le -ge ... There are many.
- Conditions can be combined with logical operators
  - `if [ -f "$file" ] || [ -d "$file" ]`
  - `if [ "$file" = "aa" ] && [ -d "$file" ]`
- Consider a substitution if \$file variable is not set, there will be an invalid set of arguments.
  - `if [ "$file" = "/home/some/file" ]`
- Use this instead:
  - `if [ X"$file" = "X/home/some/file" ]`

# Shell scripting basics

## If construct and while construct

```
if [ condition ]
then
    operation1
else
    operation2
fi

while [ condition ]
do
    commands
done
```

# Shell scripting basics

## If construct and while construct

```
if [ condition ]; then
    operation1
else
    operation2
fi
```

```
while [ condition ]; do
    commands
done
```

# Shell scripting basics

## Case construct

```
case $VAR in
  -f)
    commands for option -f
    ;;
  -d)
    commands for option -d
    ;;
  *)
    echo "Unknown parameter" >&2
    exit 1
esac
```

# Shell scripting basics

# For construct

for element in list; do

do something with \$element

done

- The commands do run for each element in the list or array. E.g.:

```
for i in $(ls *.xml); do
```

```
echo $i
```

```
mv $i $i.old
```

done

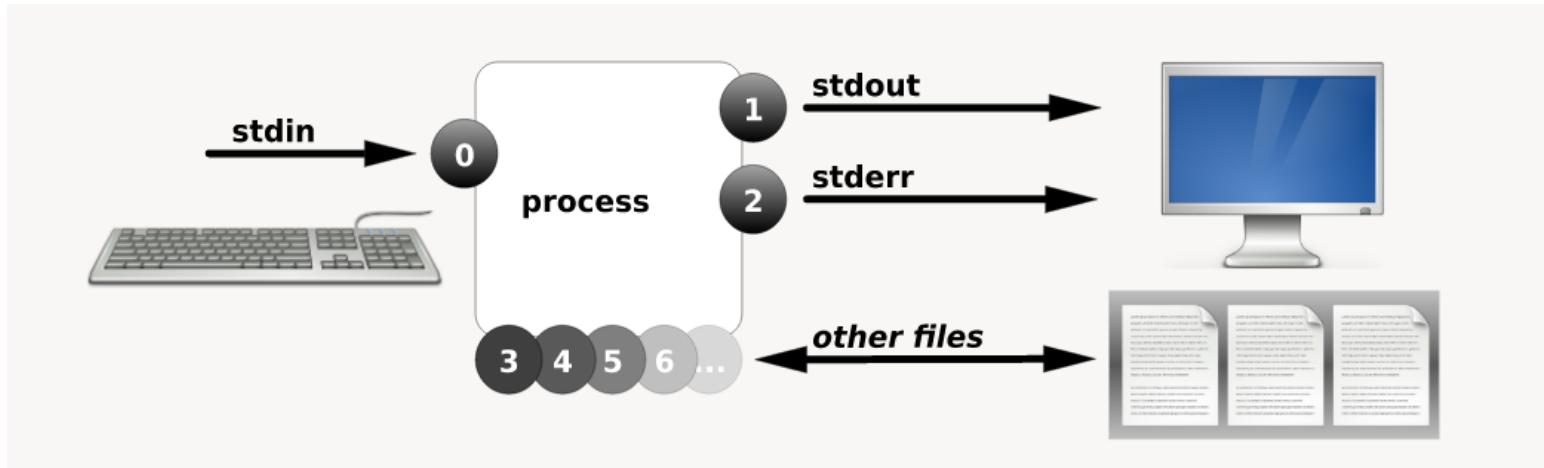
- To have C style counting for cycle use this example

```
for i in $(seq 8); do  
    i (1 2 3 4 5 6 7 8)
```

```
for i in $(seq 10 -2 4); do  
    i (10 8 6 4)
```

# Redirecting standard streams

- Remember lecture 1 slide 28-29 about file descriptors.



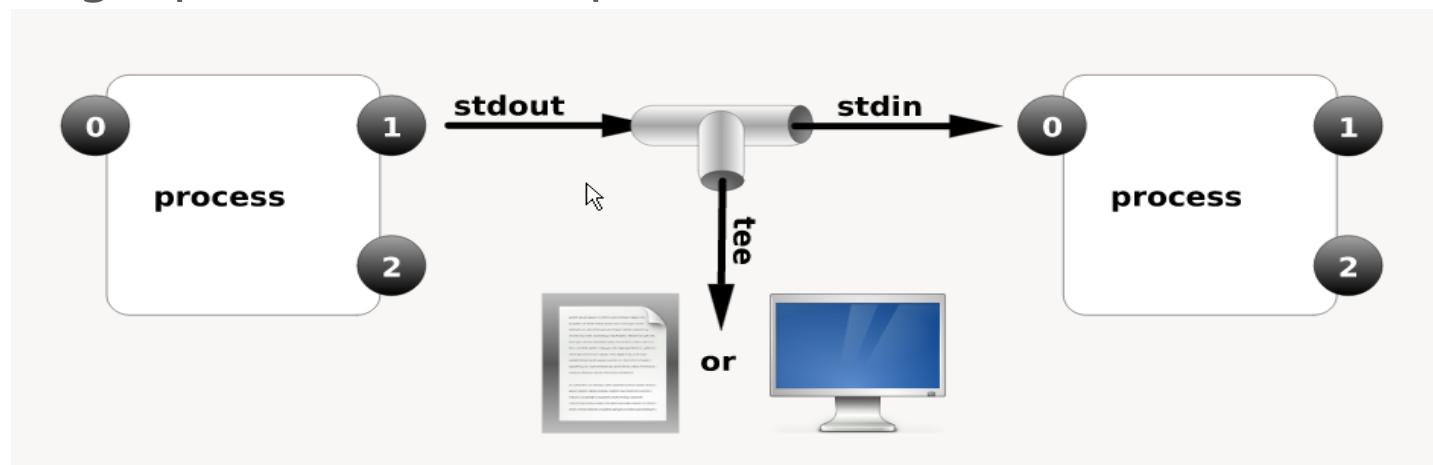
- When starting a process from shell, you can instruct it to override the default standard streams with a disk file, a named pipe or an anonymous pipe.
- This is useful for separating error messages to a log file, or automating input to a process in a script, but the most powerful benefit is the use of pipes. Pipes channel the output of a process to the input of another process. This is fast because no disk I/O or temp files are used.

# Redirecting standard streams

- Examples on how to use redirects. Refer to **RH124 CH4.1.** for more.
  - `some_program >some_file` – redirect the std out stream
  - `some_program 2>some_file` – redirect the std error stream
  - `some_program 1>some_file` – redirect the std out stream
  - `some_program >file 2>&1` – redirect stderr to std out and std out (both streams) to file. Order is important!
  - `>>file` means append to file instead of overwriting from start
  - `date -s < ~/date.sav` – redirect standard input (sets the date to the string specified in date.sav file instead of requesting to type it from keyboard.)
  - `cat <<EOF > ~/some_script.sh` – read multi-line text from input. Input can be terminated by typing EOF. Then save the output in some\_script.sh. This is called a 'here document'. (shell as text editor)

# Pipes

- To connect standard streams of processes the `|` (pipe) symbol has to be used. This creates a temporary pipeline, which is discarded after the emitting process ends. This is called an anonymous pipe.
- You can create a named pipe as a filesystem object with `mkfifo` command. This is persistent, can be deleted with the `rm` command. A pipe can only be opened for write by exactly one process and can only be read by one single process. Once open it is unidirectional.



# Pipes

- Processes can be chained using pipes infinitely.
- A pipe can be forked with the **tee** command, which works like a T junction. It forward its stdin to its stdout but also forwards a copy of the stream to the filename given as argument.
- The Linux concept of having a simple, specific, efficient utility (building block) for every task, combined with pipes is very powerful. Some examples:
  - `ls -t | head -n 10 | tee ~/ten-last-changed-files | mail student@desktop1.example.com`
  - `cut -d: -f7 /etc/passwd | uniq | sort`
  - `dd if=/dev/sda6 | pbzip2 -c -9 | nc hostB -p 2223`  
`netcat -p 2223 -l | pbzip2 -d | dd of=/dev/sda6`

# Common utilities

- If you are not yet familiar with these utilities, please look up their man page, as they are quite useful with pipes.
  - less
  - head
  - tail
  - cut
  - tr
  - uniq
  - sort
  - grep
  - sed

# Regular expressions

- Regular expressions are used to specify patterns that can be programmatically compared/matched against text.
- A utility called **grep** can be used to search files or stdin for pattern matches. Matched lines (or parts) are forwarded to stdout by grep.

## Shell command line

```
# cat /etc/passwd | grep "John Smith"  
jsmith:*:1003:1003:John Smith:/home/jsmith:/bin/sh
```

- Well known command line switches to grep are:
  - -v (invert match)
  - -i (case insensitive mode)
  - -c (count, print only number of matches, not the matched lines themselves)

# Regular expressions

Common pattern match rules and symbols:

- . (dot) - a single character.
- ? - the preceding character matches 0 or 1 times only.
- \* - the preceding character matches 0 or more times.
- + - the preceding character matches 1 or more times.
- {n} - the preceding character matches exactly n times.
- {n,m} - the preceding character matches at least n times and not more than m times.
- [agd] - the character is one of those included within the square brackets.
- [^agd] - the character is not one of those included within the square brackets.

# Regular expressions

Common pattern match rules and symbols (continued):

- [c-f] - the dash within the square brackets operates as a range. In this case it means either the letters c, d, e or f.
- () - allows us to group several characters to behave as one.
- | (pipe symbol) - the logical OR operation.
- ^ - matches the beginning of the line.
- \$ - matches the end of the line.

Refer to **RH134 CH2** on how to use grep and regular expressions.

# Midnight Commander cheat sheet

Lecture 2

Chapter 3

# Useful hotkeys

- F3 – view file
- F4 – start editor
- shift F4 – create new file in editor
- F5 – copy file
- F6 – move file
- shift + F6 – rename file
- F7 – make directory
- F8 – delete
- F9 – pulldown menu
- F10 – quit
- ctrl + o – Hide/show panels

# Useful hotkeys

- `ctrl + s` – Jump to file
- `insert` – select item
- `+` – select items (filter)
- `/` – deselect item (filter)
- `alt + a` – insert `pwd` to command line
- `alt + enter` – insert selected files name to command line
- `ctrl+x, t` – insert selected names to command line.
- `alt + shift + ?` – find file panel
- `ctrl + space` – calculate directory size
- `alt + h` – command history
- `ctrl+x, c` – `chmod` for item
- `ctrl+x, o` – `chown` for item

# Useful hotkeys in mcedit

- F2 – save file
- shift + F2 – save file as
- F3 – submit start of selection/ end of selection
- F4 – replace pattern match panel
- F5 – copy selection to current cursor position
- F6 – move selection to current cursor position
- F7 – search pattern panel
- F8 – delete selection or current line
- F9 – pulldown menu
- F10 – quit
- ctrl + u – undo

# Recommended reading

Lecture 2

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH124 Chapter 2 (filesystem layout, file manipulation)
- RH124 Chapter 4 (redirect, cat, vim)
- RH134 Chapter 2 (regexp, grep)
- RH134 Chapter 3 (vim)

- RH 124 CH5 (local users and groups)
- RH 124 CH6 (file access permissions)
- RH 134 CH6 (ACLs)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 2, PM-TRTNB219, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB219

Zsolt Schäffer, PTE-MIK

# Local users and groups

Lecture 3

Chapter 1

# Users

- All user activity, all the terminals, all the processes and all the files in the system are tied to a specific user.
- Users are accounted and internally tracked by a unique numeric identifier, called **UID**.
- Software output often shows the users name instead of the ID. This is only for convenience, internal structures store the numeric ID. The mapping of UID to username is world readable public information.
- The public part of a user account data is stored in the **/etc/passwd** file.
- The confidential part of user information is stored in **/etc/shadow** (for e.g. password hashes are not public)
- You can get information about the currently logged in user with the **id** and **whoami** commands.

# Users and Groups

- You can ask for the list of other users currently logged in to the system with the **w** or **who** commands.
- You can check the last login time of each user with **lastlog**, and you can print the most recent logins to system the with **last** command.
- You can list the running processes with **ps**. Using the **-u** switch turns on the user column.
- Groups are container objects of users for organizational purposes. All users have exactly one primary group, but additionally can be members of any number of groups.
- The default is to have a UPG (User Private Group), which means the group has only one user as a member and the group has the same name as the user.
- New files created by users will belong to their respective primary groups.

# Groups

- Every filesystem object has exactly one user and one group defined as owner.
- A user can have any number of supplementary groups and groups can have any number of members.
- Groups are also tracked and identified by a numeric value called **GID**.
- Group membership is public information, it is stored in the **/etc/group** file.
- Groups can also have passwords. This is confidential, therefore separated to the **/etc/gshadow** file.
- Refer to RH124 CH5.1. for further information.

# UID conventions

- UID of 0 (zero) is always assigned to the administrative superuser called 'root'.
- UIDs 1-999 are assigned to system users. These are unprivileged users, usually assigned to daemon processes in order to limit their access to a specific subset of tasks/files. They usually can't even log in to a shell session in a regular way.
- A subset of system users (1-200) are statically assigned by RedHat, the rest can be assigned dynamically.
- UIDs starting from 1000 are available for regular (human) users.
- A special UID of 65534 is assigned to the most unprivileged user, called 'nobody'. This is often needed to map privileges of network users to remote filesystems.

# Local user accounts

- The source of user and group accounting information is a database. It can be stored and distributed through several protocols and mechanism, even over a network.
- In this chapter only the locally defined accounts will be discussed.
- Data is stored in tables, that can be viewed or edited with an editor. These are simple text files (file names shown earlier in this chapter) that contain data related to one user per line. Fields are separated with a : (colon).

/etc/passwd

```
jsmith:x:1003:1003:John Smith:/home/jsmith:/bin/zsh
jdoe:x:1004:1004:Jane Doe:/home/jdoe:/bin/bash
```

/etc/group

```
students:x:1010:bob,alice,jsmith,jdoe
```

# The passwd file

- User name
- Password field, x means an encrypted password has to be looked up elsewhere → /etc/shadow
- UID (numeric)
- GID of primary group (numeric)
- GECOS (freely usable text field: comments, location, real name, etc.)
- Path to the user's home directory
- Shell (It doesn't have to be an actual shell, note for e.g. /bin/nologin or /bin/false)

/etc/passwd

```
jsmith:x:1003:1003:John Smith:/home/jsmith:/bin/zsh
jdoe:x:1004:1004:Jane Doe:/home/jdoe:/bin/bash
```

# The group file

- Group name
- Group password field, x means to look for encrypted password hashes elsewhere → /etc/gshadow
- GID (numeric)
- Enumeration of all the members. Member user names are separated by a , (colon).

/etc/group

```
students:x:1010:bob,alice,jsmith,jdoe
```

# The shadow file

- User name (as string, not numeric ID)
- Encrypted password: has several sub-fields (next slide)
- Last change date (days since Epoch)
- Minimum days before user can change password
- Maximum days before user must change password
- Nr. of days to warn user before obligatory pw. change
- Account disable timeout (in days from pw expiry)
- Days since account has been disabled
- Reserved field

/etc/shadow

```
jsmith:$1$gCjLa2/Z$6Pu0EK0AzfCjxjv2hoLOB/:10063:0:  
99999:7:::
```

# The shadow file's password subfields

- The password field has several subfields, separated by \$ (dollar) sign: \$hash\_algo\$salt(optional)\$pwhash
- Hash algo examples: \$1\$ for MD5, \$5\$ for SHA256, \$6\$ for SHA512
- Salt is a random sequence, hash is function of both salt and password string. → Note users with same password
- An ! (exclamation mark) in the password field indicates the account's password is locked. (SSH keypair login is still possible, password login is not possible)
- An \* (asterisk) in the password field indicates the account has been locked. (No login possible)
- An empty password field indicates the user can log in without entering a password. (Guest users)

# The gshadow file

- Group name (as string, non numeric)
- Encrypted password
  - ! means password group login (newgrp) is disabled
  - If set to a hash, users can log in the group with a valid password.
- List of group administrators (comma separated)  
A group admin can change the group password and can add/remove members . (gpasswd)
- List of members (comma separated)  
Members can log into the group without typing a password even if password is set. (newgrp)

/etc/gshadow

students:!!!:instructor:bob,alice

# Managing users

- You can manage before-mentioned text files manually. Or you can use the following command line tools. This conveniently also creates a home directory and sets its permissions, besides adding a new user entry.
- The **useradd [switches] username** command can create a user, note the following switches and default values
  - -c to set comment (Gecos) field (empty)
  - -s to specify shell (/bin/sh)
  - -d to specify home directory (/home/[username])
  - -g to specify primary group (UPG)
  - -G to specify list of supplementary groups (none)
  - -u to specify UID (add one to the last used UID)

Shell command line

```
# useradd -c "John Smith" -u 1050 -G video jsmith
```

# Managing users

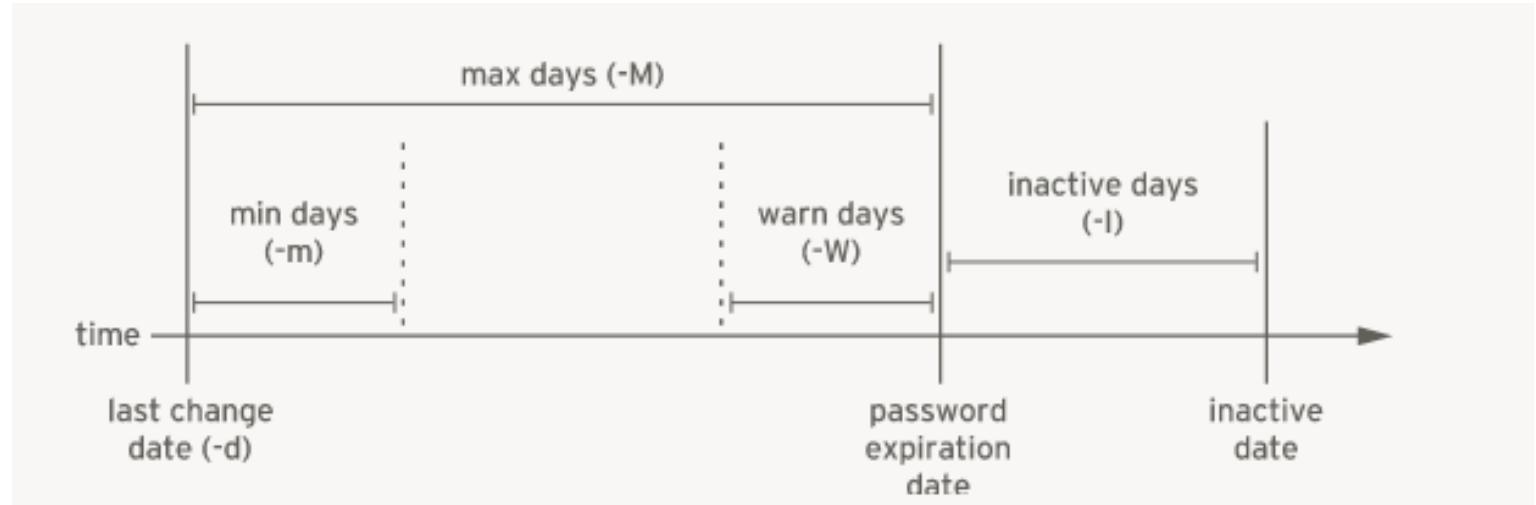
- You can modify an existing user with **usermod [switches] username** using similar switches. Also note the -L and -U switches to lock and unlock the password.
- The **userdel username** command removes the user. Using the -r switch also removes the users home directory from the filesystem.
- Note the seemingly conflicted UIDs in RH124 CH 5.5. when deleting and adding users.
- You can change your own password with **passwd** command. Only root can change passwords for other users. Root can do it like this: **passwd alice**
- Complete lab practice under RH124 CH 5.6.

# Managing groups

- You can add a new group with **groupadd** command.
- You can modify a group with **groupmod**.
- You can remove a group with **groupdel**.
- You can modify group membership with the **usermod** command as described earlier. Note the **-g -G** and **-a** command switches.
- Refer to RH124 CH 5.7. for further
- Complete lab practice under RH124 CH 5.8.

# Managing password aging

- You can adjust the timing fields of the /etc/shadow file with **chage**.



- Use chage -E to set absolute account expiry.
- You can also use the usermod -e command to set expiry.
- Also note the -l, -d options to chage.
- Refer to RH124 CH5.9.-5.10. for further information.
- Complete practice RH124 CH5.11.

# File access permissions

Lecture 3

Chapter 2

# Permission and owner information

- As stated earlier in this lecture files have exactly one owner and one group owner. Processes also have owners and group owners.
- A file can only be accessed by a process if it is permitted by comparing their ownership information and evaluating file permissions.
- In \*nix systems permissions are organized in 3 sets for each filesystem object. One set is for the user owner (u), one set for the group owner (g) and one set for all the other users (o).
- Each set contains three binary values, each determining one of whether or not read, write and execute operation is permitted on the file.
- There are some more special bits for each file that are not part of the three sets. (sticky, setgid, setuid)

# Permissions explained

- Permission sets can be printed by adding the `-l` switch to the `ls` command.

## Shell command line

```
# ls -l /var/lib
drwxr-xr-x.  2 root      root  4096 2014 ápr  14 bluetooth
drwxr-xr-x.  2 chrony    chrony4096 2014 szept 10 chrony
drwxr-xr-x.  3 root      root  4096 2013 dec   12 color
drwxr-xr-x.  3 colord    colord4096 2014 nov   5 colord
drwx-----.  2 apache    apache4096 2014 dec   17 dav
```

- Read and write permissions are self-explanatory for files. Execute permission allows for text scripts or ELF binary files to be started (a process created from them)
- Keep in mind, that directories are simple lists only. They list filename-to-inode mappings. (Further in Lecture 5)
- Read permission of directories allows for reading the list itself. (listing)

# Permissions explained

- Write permission of a directory allows for renaming, adding or deleting files within the directory. (Editing the list). Write only works with execute bit also set.
- Execute permission of a directory allows for entering it with **cd**, and accessing its contained objects.
- Having execute without read for directory makes it impossible to list the objects inside. On the other hand you can still read/access such objects, if you know them to be there and know their name exactly.
- Having read without execute makes listing possible, but objects can not be accessed inside such directories. E.g. **ls** works, but with no details are shown (no access to inodes). Also no files can be read or modified.
- Refer to RH124 CH6.3 for further details.

dir permissions	Octal	del rename create files	dir list	read file contents	write file contents	cd dir	cd subdir	subdir list	access subdir files
---	0								
-W-	2								
R--	4		only file names (*)						
RW-	6		only file names (*)						
--X	1			X	X	X	X	X	X
-WX	3	X		X	X	X	X	X	X
R-X	5		X	X	X	X	X	X	X
RWX	7	X	X	X	X	X	X	X	X

# Octal (numerical) representation

- The octal representation of permissions is also commonly used. Values for special bits are: setuid – 4, setgid – 2, sticky – 1

	u	g	o
access	r	w	x
binary	4	2	1
enabled	1	1	1
	<hr/>	<hr/>	<hr/>
result	4	2	1
	<hr/>	<hr/>	<hr/>
total	7	5	4

# Special permissions

- The **sticky bit** can only be effective with the executable bit set. For files it means to keep the binary in swap for performance tuning. For directories that are world writable (others have w bit) it restricts removing/renaming objects to the owners of said object or the owner of the sticky directory itself. (E.g. /tmp)
- The **setuid/setgid** bits only make sense in conjunction with the executable bit set. A process created from such a file will not have the owner of the user session which executed it, rather it will have the owner and group of the filesystem object instead. Applying the setgid bit to directories forces the group of directory and the setgid bit itself to be inherited to newly created objects inside the directory. Setuid can work similar for the owner but on some systems it may have no effect.
- Note the **ls -l** representation (s,S,t,T)

# Modifying owner and permissions

- You can use the **chmod** command to change permissions of a file. Note the different representations accepted by chmod. E.g. chmod 0755 file, chmod o=rx file, chmod a+w file, chmod -222 file
- The owner can be changed with **chown**. Consider the following forms: chown student file, chown student:class file, chown :class file
- Only the owner can change the file mode. Only root can change the owner of a file (no give-away possible). Group owner can be set by users who are members of both the original and newly set group.
- You can set the default permissions for new objects with the **umask** command. Specify a mask that is subtracted from 777 → the result will be applied to all newly created objects. (also see bashrc)
- Complete practice RH124 CH6.6. and 6.7.

# Teamwork example

- If you had two groups of people, let's say mechanical engineers and electrical engineers and wanted them to have separate directories which are shared among members but not across groups, you could follow this:
- Add each engineer to either the electrical or mechanical group. Create one directory owned by the group for each of the groups. Set the setgid bit for the directories. Set mode for all files and sub-directories to 770. This way all **new** files will be read/writable to the respective group. A specific user can be member of both groups, but created files will only be accessible for one of the groups, the users primary group.
- Now if you wanted to share something between the two groups, you would have to create an additional directory with an additional group, which has all the engineers as members. Than have all users use newgrp accordingly.

# ACLs

- As the number of combinations of possible group memberships grow, this can become quite complex.
- Also it wouldn't be easy to specify negative rights with the standard permission bits. For e.g.: every electrical engineer is allowed except for bob.
- That's where ACLs are more relevant. Access control lists are policies made up of several rules. Each rule can target a specific user or a specific group.
- Each file can have an ACL consisting of any number of rules.
- If ACL is present on a file it will have 3 mandatory elements which can not be deleted, only modified. One for the owner, for the group owner and one for others. These three are called base ACLs, they are copied from standard permissions upon ACL initialization.

# ACL mask

- The base ACLs for owner and others are 1:1 equivalents to standard permissions, chmod actually changes these two base ACLs as well, and changing these ACL entries is represented by file mode. (works vice-versa)
- What ACLs do, is that they override the standard 3-bit group permission mechanism with a set of named rules targeting users and/or groups, including the group owner itself.
- The **ACL mask** only affects the mentioned part of entries, specifically named user and named group entries plus the group owners entry. It does not affect the owning user part and the 'others' part of ACLs.
- The mask is applied as a bit-wise **AND** operand to the subset of rules mentioned before, resulting in the effective ACLs.

# ACL entries

- A + sign in ls output shows that ACLs are set for a file. In this case the 3-bit group permissions would make no sense, since there is a list of rules instead → the **middle triplet** shows the ACL mask, and chmod-ing the middle triplet changes the mask **instead of group permissions**.

## Shell command line

```
# ls -l /home/student/Documents
drwxr-x---+ 2 student school 4096 2017 dec 12 thesis.txt
```

- A file has exactly three base ACLs. Each file can have one or zero mask entry. A mask entry is mandatory if any non-base ACLs are set.
- Besides that a file can have any number of named entries. A named entry can specify a name string or a numeric identifier as well.
- A directory can also have default ACLs. These are automatically inherited to newly created sub-items.

# ACL evaluation

- The evaluation order is the following:
  - If the accessing user owns the file → the user ACL (triplet) applies. (just like with standard permissions)
  - If the the accessing user has a named entry → it will be applied.
  - If the accessing users group own the file → the group ACL is applied.
  - If the accessing group has a named entry → it will be applied.
  - If not any of the above rules match → the 'other' ACL will be applied. (just like with standard permissions)
- For further on ACLs, read RH134 CH6.1.
- For setting and checking ACLs refer to RH134 CH6.3. and the following three slides.

# Reading ACLs

## Shell command line

```
# getfacl some_directory
# file: some_directory
# owner: student
# group: controller
# flags: -s-
user::rwx
user:james:---
user:1005:rwx      #effective:rwx-
group::rwx         #effective:rwx-
group:sodor:r--   #effective:rwx-
group:2210:rwx     #effective:rwx-
mask::rwx
other::---
default:user::rwx
default:user:james:---
default:group::rwx
default:group:sodor:r-x
default:mask::rwx
default:other::---
```

# Setting ACLs

- An ACL can be set/replaced with the **setfacl -s** command. Use the **-m** switch to modify an ACL entry, use **-x** to remove an entry. Use the **-d** switch together with the former two to modify/remove a default entry.
- The format starts with **u:** **g:** **o:** or **m:** for user, group, other, and mask entries respectively.
- The next element is an object specifier (user/group name/id). **Mask** and **other** entries have blank specifiers.
- An empty user or group specifier corresponds to the user/group who owns the file.
- The last element is the permission set itself. e.g **rx**

## Shell command line

```
# setfacl -m u::rwx,g:sodor:rX,o::- filename
# setfacl -x -d u:james filename
```

# Setting ACLs

- You can use the -R switch to apply ACLs to directories and sub-entries recursively. A permission set containing a X means to apply x permission only to directories, not files.
- You can remove all default ACL entries of a directory with `setfacl` using the -k switch. The -b switch removes ACLs completely.
- Modifying the ACLs with `setfacl` can cause the mask to be recalculated. To inhibit mask change use the -n switch to `setfacl`.
- ACLs somewhat overlap with Linux permissions, it is recommended that you don't use `chmod` anymore on files which have ACLs. It is recommended to use `setfacl` instead.
- Complete practices RH134 CH6.4. and 6.5.

# PAM, nsswitch

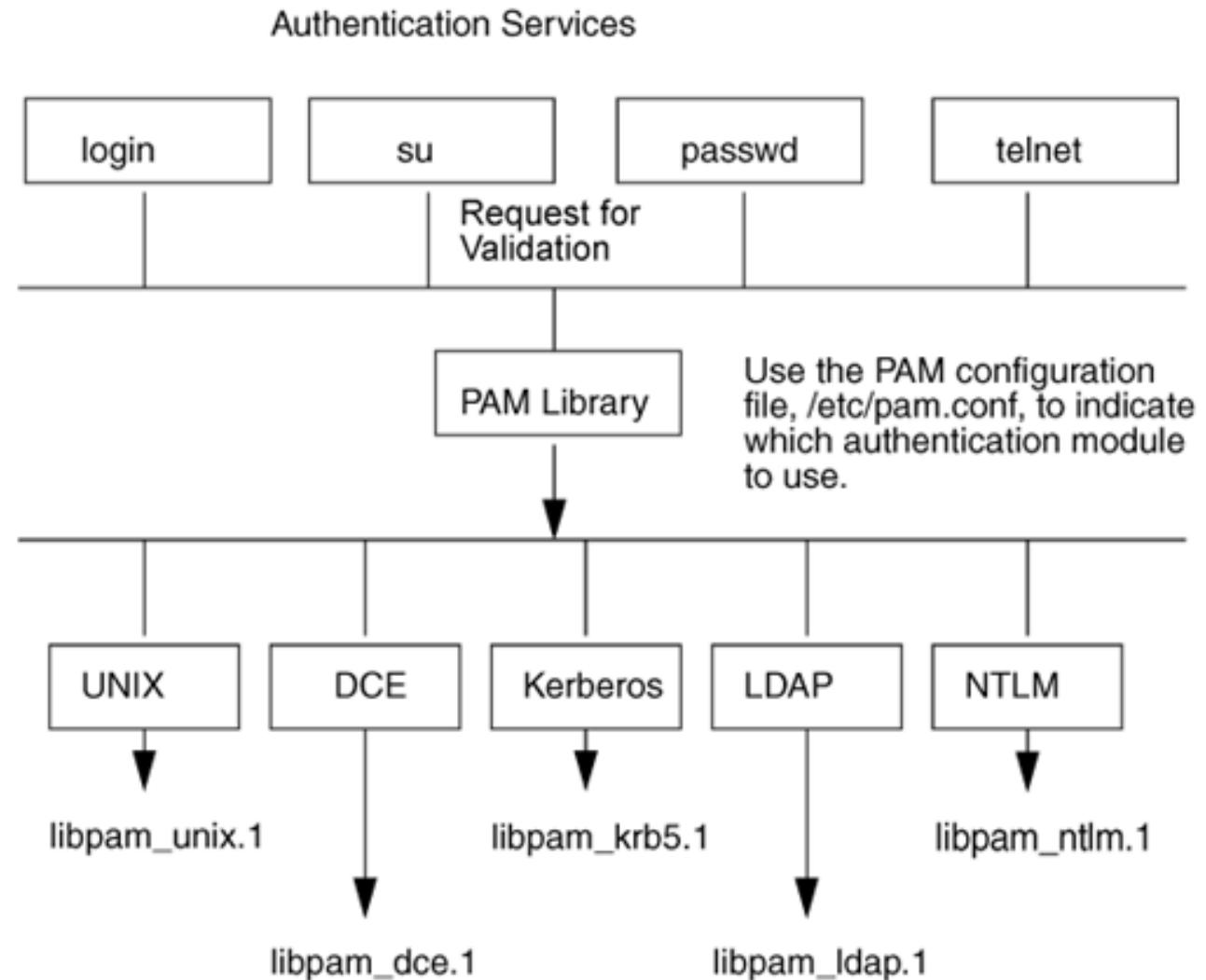
Lecture 3

Chapter 3

# PAM

- Pluggable Authentication Modules apply an abstraction layer to authentication mechanism of Unix.
- Applications send a request for authentication to PAM.
- PAM then performs the authentication with several steps and checks, in a configurable way. This involves using modules.
- When all the necessary modules are done, a response is returned to the application.
- In this way any changes to the authentication scheme induces a reconfiguration of PAM but not the reconfiguration/recompilation of applications themselves.

# PAM diagram



# PAM module examples

- pam\_time: decides whether a user's request for authentication is allowed at a given time from a given terminal. (`/etc/security/time.conf`)
- pam\_securetty: only allows root to log in from a subset of terminals, which are considered secure. The set is listed in `/etc/securetty` file.
- pam\_unix: checks if entered password corresponds to hash stored in `/etc/shadow`.
- pam\_nologin: denies login if `/etc/nologin` file exists.
- pam\_allow: unconditional allow, always returns true.
- pam\_deny: unconditional deny, always returns false.
- pam\_warn: write authentication request and details in system logs.
- pam\_pwdb, pam\_limits, pam\_env, etc...

# PAM modules

- There are four activity types of PAM modules:
  - **authentication** modules verify user identity, typically by asking for a key, password, secret, fingerprint, biometric characteristic, something only a valid user would know/posses.
  - **session** modules perform actions after authentication (session start) and on logout/loss of connection (session end). E.g. print motd, mount home directory, etc...
  - **account** modules check if authentication target (user) is valid under the given conditions. (e.g. time of day, terminal, IP address, etc.)
  - **password** modules take part in updating authentication credentials (e.g. enforcing strong password on pw change.)

# PAM control flag

- There are four control flags possible for modules:
  - **requisite**: If such a module fails, PAM will immediately return failure without any further module calls.
  - **required**: If such a module fails, PAM will return failure, but will continue to call remaining modules in the stack. (For e.g. log the issue, print something, etc.)
  - **sufficient**: If such a module succeeds, PAM will return pass, and stop calling any further module functions.
  - **optional**: such module's return value is ignored. Usually it stands for modules that should perform an action either way. (E.g. log the authentication attempt, regardless whether it fails or passes.)

# PAM examples

- Prevent non-root users from shutting down the system:

```
/etc/pam.d/halt
```

```
auth      sufficient  pam_rootok.so
auth      required    pam_deny.so
```

- Enforce strong passwords

```
/etc/pam.d/system-auth
```

```
[...]
password  requisite  pam_passwdqc.so  min=12,10
           ,10,8,6  retry=3
[...]
```

- Allow only members of 'wheel' group to use su

```
/etc/pam.d/su
```

```
auth      sufficient  pam_rootok.so
auth      required    pam_wheel  use_uid
auth      include    system-auth
```

# PAM examples

- rlogin PAM example

/etc/pam.d/rlogin

```
auth      requisite /lib/security/pam_securetty.so
auth      requisite /lib/security/pam_nologin.so
auth      sufficient /lib/security/pam_rhosts\
           _auth.so
auth      required /lib/security/pam_unix.so
account  required /lib/security/pam_unix.so
account  required /lib/security/pam_time.so
password required /lib/security/pam_cracklib.so \
           retry=1 type=UNIX minlen=10 ocredit=2 dccredit=2
password required /lib/security/pam_unix.so \
           use_authok shadow md5
session   required /lib/security/pam_unix.so
session   optional /lib/security/pam_motd.so \
           motd=/etc/motd
```

# NIS, nsswitch

- On large networks, in case of large number of nodes, it could be troublesome for user/password/hostname/etc.. databases to be maintained, kept synchronized on all hosts.
- System settings, like user accounts can come from different sources. E.g. the local /etc/passwd file or DNS (Domain Name Service) or NIS (Network Information Services) or directory services like freeIPA, LDAP, AD, etc...
- A network source might be more convenient in case of large node count.
- Look at /etc/nsswitch.conf for examples

# NIS, nsswitch

- nsswitch.conf

/etc/nsswitch.conf

```
passwd:      compat
group:       compat
shadow:      compat

hosts:       files dns
networks:    files

protocols:   db files
services:    db files
ethers:      db files
rpc:         db files

netgroup:    nis
```

# Recommended reading

Lecture 3

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH 124 CH5 (local users and groups)
- RH 124 CH6 (file access permissions)
- RH 134 CH6 (ACLs)

- RH 134 CH7 (SELinux)
- RH 124 CH7 (manage linux processes)
- RH 134 CH5 (nice)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 3, PM-TRTNB219, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB219

Zsolt Schäffer, PTE-MIK

# Linux processes, jobs

Lecture 4

Chapter 1

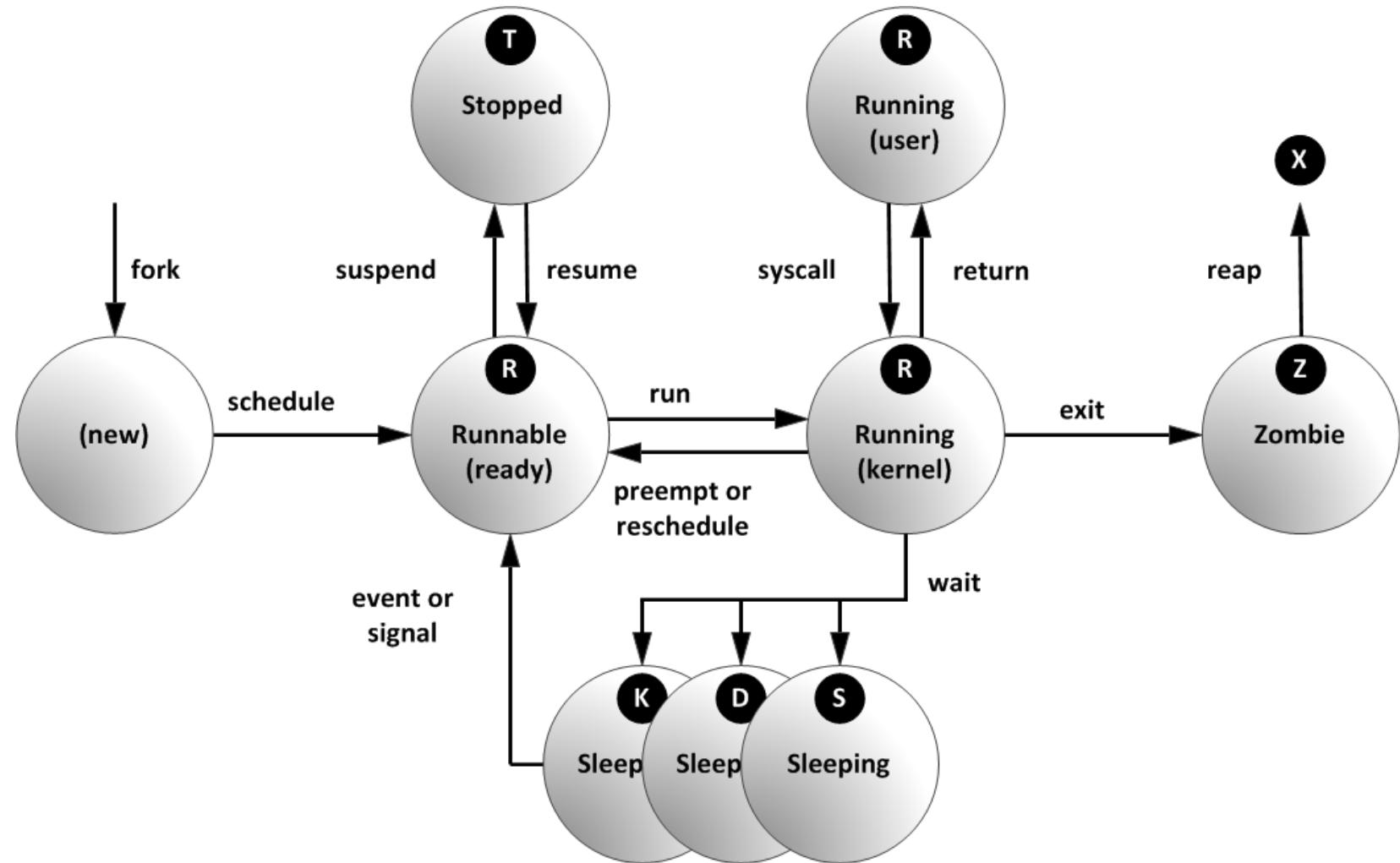
# Processes

- Processes are running instances of an executable program. (Remember Lecture 1)
- Every process has a virtual, isolated address space configured in the MMU. It also has at least one thread of executable code.
- Threads are independently scheduled, just like processes, but threads of the same process share the same virtual address space.
- Processes also have some metadata and accounting information. For e.g. PID, PPID, security properties (SELinux context, capabilities), ownership (UID, EUID, GID, EGID), a current scheduling context, environment variables (refer to shell in Lecture 2), a list of allocated resources (mmaps, fds, open ports in the TCP/IP stack), etc. (Try `ls -la /proc/[PID]/fds`)

# Scheduler

- Processes only get a series of limited time slices to execute code on the CPU. Each CPU runs only 1 process at a time. The time slices are assigned/distributed by the scheduler, which is a part of the kernel binary.
- The scheduler is run periodically by a hardware timer interrupt at regular intervals. The scheduler decides what process to put next in the CPU for active execution. This involves saving the instruction pointer, the stack pointer and other register states of the 'leaving' process, and loading the same registers for the newly scheduled one.
- There are different scheduling algorithms and different considerations to select the next process in each algorithm. For e.g.: how long has the process been starving, how 'nice' the process is, what state does the process currently have (waiting for disk data, waiting for a network packet.), etc.

# Process states



# Process states

- S,D,K states: When a process is waiting for user input, a disk read or a network packet, it would be inefficient for it to regularly check for the required condition (polling). Instead the scheduler itself is notified, that it should not give a time slice to the process in question, until a certain event occurs. (E.g. a packet arrives to the network port, which the process is listening on). In this case the process is in the **sleeping state**. Look up the `select()` syscall and synchronous I/O for more.
- T state: A process can also be manually set aside from the pool of schedulable tasks, by sending a stop signal. This is reversible with a resume signal. A debugger also 'pauses' processes if a breakpoint or certain condition is reached. This is called the **stopped state**.
- R state: All the processes that are not sleeping/stopped are eligible candidates for immediate activation by the scheduler. This is called the **Runnable** or **ready** state.

# Process states

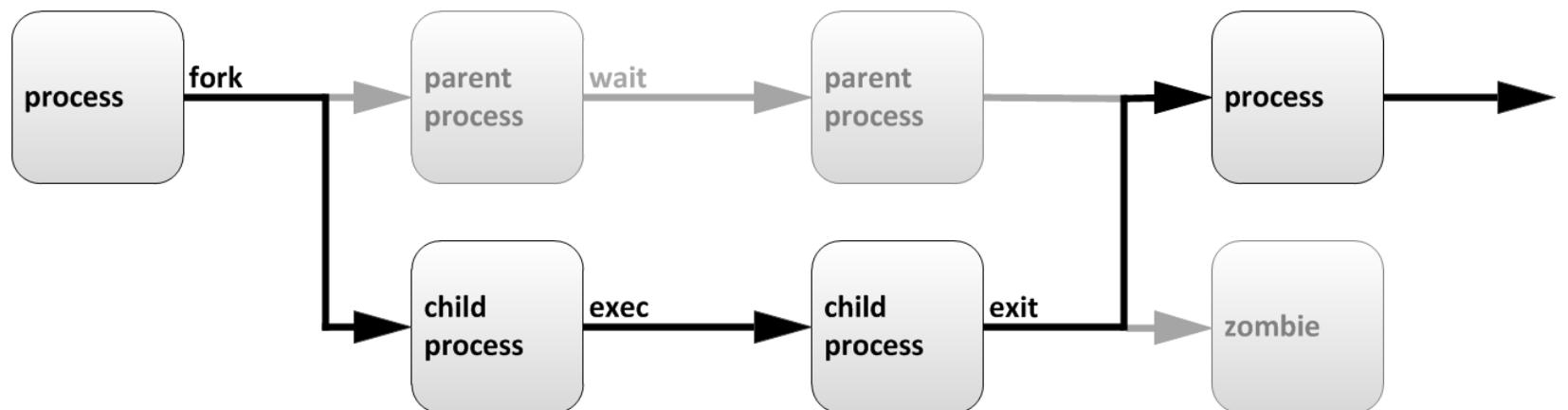
- A new process is also put in the pool of ready processes.
- The scheduler selects one process per CPU from the runnable pool and puts it in the **running** state. A running process is the one actively using the CPU core. The program flow can take excursions when executing syscalls, which are implemented in kernel space. The user space process is not progressing until the syscall function returns.
- When the time slice is over, the running process is preempted and put back to the runnable pool. Or if it requested an I/O operation, it is put in the waiting state.
- The selection process (the scheduling algorithm) usually runs 100 to 1000 times per second, depending on scheduler algorithm, the current dynamic of the load and some compile time kernel parameters.
- Refer to RH124 CH 7.1. for more details

# Process creation

- In Linux a process can only create an other process by replicating his own self. There are multiple ways for this, like vfork (obsolete), fork, clone, etc.
- The fork() syscall creates an exact replica of the process. A new PID is assigned and PPID adjusted. Otherwise everything is an exact copy, including owner, security context, fds, environment, even memory page layout. Today's optimized kernels don't copy the whole mem space of processes, instead the pages are initially shared and upon modification a dedicated copy is made on-the-fly to an available chunk of memory (COW).
- Note, that at the point of returning from fork() there are two exactly matching processes, with same binary code and same execution state: both in about to return from a system call. The call returns 0 inside the child process and returns the PID of the child in the parent process.

# Process creation

- The exec() syscall replaces the current executable image with something loaded from a binary file. This is often called directly after fork(). Control is never returned to the original program from exec.
- The clone() syscall can create a child object with a more refined control over what is going to be shared, and what is going to be copied. Clone can create threads inside the same process space and also create full proc. replicas like fork(), it depends on arguments passed to clone().



# Process creation

- When a child process finishes or terminates irregularly, the parent process is signaled and is expected to read the return value of the child.
- A process will be in **zombie state**, when it is terminated but not queried yet by the parent (this is called reaping, usually short term). A zombie does not hold resources but still reserves a PID.
- Zombie processes can't be killed, since they are already dead. They can be cleaned up by killing their parent.
- The zombie children of a killed process fall under the parenthood of init (PID 1), which regularly reaps them.
- Their accumulation can be dangerous, because there is only a finite number of PIDs available: 32767 in a 32 bit system.

# Process ownership

- It comes from the nature of how processes are created, that the owner of a child process is the same as the owner of the parent (user and group).
- There is syscall for changing the UID to another one at runtime, named setuid(). This can only be done by root-owned processes. It is common, that daemons -after being started by init as root- give up their privileges for security reasons and fall back to a technical user like for e.g. apache. (note RUID, EUID, RGID, EGID)
- The other exception is when starting a process from a binary with the setuid flag. In this case the EUID is set to the owner of the file instead of the executor. (Lecture 2)
- Take the example of running passwd as regular user to change the hash in the /etc/shadow file. This file is not read/writable by regular users, but /bin/passwd binary has setuid flag, and is owned by root. (note RUID's role)

# Listing processes

- You can use the **ps** command to list processes. The **-e** switch enables listing of all processes, otherwise own processes are printed, only the ones with a controlling terminal. Switch **-f** turns on additional columns like user, command, arguments and PPID. **-H** turns on process tree view.
- Note the BSD style equivalent is **ps auxf**
- Use the **top** command to display a repeatedly updated, sorted list of processes. Note the **f** command key for field selection, **H** for displaying threads, **1** for per-CPU load, **u** for user filter, **k** for kill, **r** for renice, and **q** for quit.
- Use **pstree** to see parent-child relationship formatted as a tree with ASCII graphics.
- For more on ps refer to RH124 CH.7.1 For more on top refer to RH124 CH7.7.

# Load average

- The load average triplet is the load numbers calculated exponential moving average for the past 1, 5 and 15 minutes. Try the **top** and **uptime** commands.
- The load number show the total number of processes currently in the wait queue for being scheduled to the 'run' state plus the number of processes waiting for I/O.
- The load average can only be interpreted in conjunction with the number of logical CPU cores (hyperthreads).

## Shell command line

```
# cat /proc/uptime
1573985.02 154800441
# cat /proc/loadavg
0.00 0.01 0.05 1/103 10130
# grep "model name" /proc/cpuinfo | wc -l
4
```

# Signals

- Signals are software delivered interrupts to a process.
- A signal source can be an external event (like I/O request), a hardware originated critical event (SIGSEGV, SIGFPE), or an explicit user command. (e.g. a key combination in terminal or the kill command)
- Signals are meant for the scheduler, and always target a specific process. Some of the signals can be handled or overridden by the targeted process. In this case a handler function (implemented in the user space application itself) is triggered. Otherwise the scheduler itself takes direct action to the process (e.g. unschedules, resumes, or forcefully terminates the process).
- Some known signals: SIGINT (ctrl+c), SIGTERM, SIGTSTP (ctrl+z), SIGCONT, SIGKILL, SIGHUP
- Refer to RH124 CH 7.5 for more details.

# Sending signals

- You can use the **kill [-signal] PID** command to send a specific signal to a process defined by its ID number. If the signal parameter is omitted the default SIGTERM is sent. Use the **-l** switch to list the available signals. Signal numbers are architecture dependent, but names are standardized.
- You can use the **killall [-signal] pattern** command to kill processes matching 'pattern' by name. Use **-u** to filter for user name.
- The **pkill** command can also send signals to multiple processes by filter criteria. To forcefully log out a user: **pkill -SIGKILL -u bob** This kills all his processes. To terminate a single problematic process: send SIGTERM to the bash process of the controlling terminal. Bash will survive, but its children won't.
- Complete practice RH124 CH 7.6.

# Jobs

- Processes and schedulers are operating system concepts. On the other hand jobs are shell related entities. Every pipeline of commands entered in the shell is a job. The processes in a pipe chain are the members of the same job.
- A shell can handle multiple jobs, but only one job can be in foreground. The fg. job is the only one that can get characters and control signals from the terminal input.
- Background jobs can be either be in executing state or can be suspended and resumed later. To start a job right in the background add an **&** (ampersand) to the end of the command. To suspend and send an already executing fg. command to background press **ctrl+z**.

Shell command line

```
# example_command | sort | mail -s "Sort output" &
```

# Jobs

- Jobs sent to background with `ctrl+z` are also immediately paused (suspended).
- To list jobs, use the **jobs** command.
- To bring a job back to foreground use **fg %jobnum**. This will also make it resume.
- To resume a suspended job while still keeping it in the background use **bg %jobnum**.
- Use **ctrl+c** to ask a process for terminating itself cleanly.
- Refer to RH124 chapter 7.3. for more.
- Complete practice RH124 CH 7.4.

# Process priority

- Processes managed by the SCHED\_NORMAL scheduler policy have a numeric representation of their priority.
- This is called niceness. A process can have a nice value of any integer ranging from -20 to +19.
- A nice process politely lets others take the CPU first. The higher the nice value the lower the priority.
- Regular users can only increase niceness and can only assign 0 or positive initial nice values. Root can also define negative values and decrease the niceness of a running process.
- To start a process with a specific nice value use the following example.

Shell command line

```
# nice -n 15 bitcoin_miner &
```

# Process priority

- To print the ninceness of processes:

Shell command line

```
# ps axo pid,comm,nice -sort=-nice
  PID COMMAND          NI
    15 netns          -20
    16 perf          -20
    17 writeback      -20
[...]
 10624 sshd            0
 10626 bash            0
 10653 ps              0
    18 ksmd            5
  554 alsactl         19
```

- To re-nice an already running process use the following command: **renice -n <LEVEL> <PID>**.
- To change niceness in top, press r then type the PID, then the new nice level.

# Practice

- For listing, sorting, signaling, and re-niceing processes with the **top** utility refer to RH124 CH 7.7 and RH134 CH 5.3.
- Complete RH124 CH 7.8. and 7.9.
- Complete RH134 CH 5.4. and 5.5.
- Also note other useful tools like: iftop, iotop, iostat

# **su, sudo and setuid()**

Lecture 4

Chapter 2

# Switching users with su

- It is a security based recommendation even for administrators to use the system as a regular unprivileged user and gain administrative privileges only temporarily, when a specific command requires it.
- To switch user account without logging out, use the **su** command. The password of the to-be-acquired user will be prompted for. Ending the elevated session with **exit** will return to the original user session. (It has not been logged out, just been passive during the su session)
- Privilege escalation is achieved to following way: the su binary has the setuid bit set and owned by root. It will start as a root owned process. It will then check for password validity and call setuid() syscall to set its own EUID according to the requested username, than spawn a shell process with the relevant user. (Note the importance of trusted binaries owned by root with setuid bit.)

# Switching users with su

- Take the the following example: **su apache -c 'cat /var/www/html/index.html' -s /bin/bash**
- If a username is given as parameter, it switches to that one, otherwise if unspecified, root user is assumed.
- The su binary can be instructed to override the users shell with the -s switch.
- It can spawn a login shell with -l (remember Lecture2).
- You can instruct su to run a single command as another user, then end the session immediately. This is achieved with the -c switch. The command should be enclosed in quotes if it contains any spaces or special characters.
- To simply switch to root with root environment (login shell) use this: **su -**
- **Target users password is always required**, except when already root, and want to switch to a regular user.

# Temporary elevation with sudo

- The **sudo** command works in a similar way, but has a fine grained control over who can change to what user and what subset of commands he/she can execute with the elevated/changed user credentials.
- Similar to su you can use -s switch to specify shell, but use -i to start a login shell and -u to specify a user. If username is omitted root is implied.
- Without -i the sudo command is useful for executing only one single command in elevated mode.

## Shell command line

```
# sudo usermod -aG wheel elvis
```

- A ruleset of who can do what is specified in the file /etc/sudoers file. Sudo will ask for the current users password as the user has prove who he/she is. The elevated command will spawn if rules allow. **Target users password is never required when using sudo.**

# /etc/sudoers file format

- The ruleset is specified in this file. One rule in each line. The /etc/sudoers.d drop-in directory is also parsed for files with lines of the same format.

## /etc/sudoers

```
root      ALL= (ALL) ALL
%chem     CHEM= SHUTDOWN, MOUNT
harvey    ALL = NOPASSWD: SHUTDOWN
<user 1> <host 1>=<operator 1> <tag 1> <command 1>
```

- All lists can be specified as a comma separated list. Aliases are also possible to use. (E.g SHUTDOWN is an alias for /sbin/shutdown, /sbin/halt, /sbin/poweroff) ALL is a universal alias for everything/everyone.
- User list is a list of users who are allowed to run the command specified by the rest of the line (% for groups).
- Host list is a list of hosts where the given rule applies. (Mass distribution of rules, rules not relevant to the current host are ignored.)

# /etc/sudoers file format

- Operator list contains the possible users to be 'impersonated' by the sudo command. (Refer to -u switch) This section can be omitted, then ALL is assumed.
- Tag list is a : (colon) separated list of tags, like NOPASSWD:NOEXEC: Can be omitted if no tags are needed.
- Command list is a list of allowed commands for the given user to run.
- Users can be given restricted rights to do limited task as root or other users, while not giving them full access to modify the system.
- Use visudo command to open up sudoers in the default editor. (\$EDITOR)

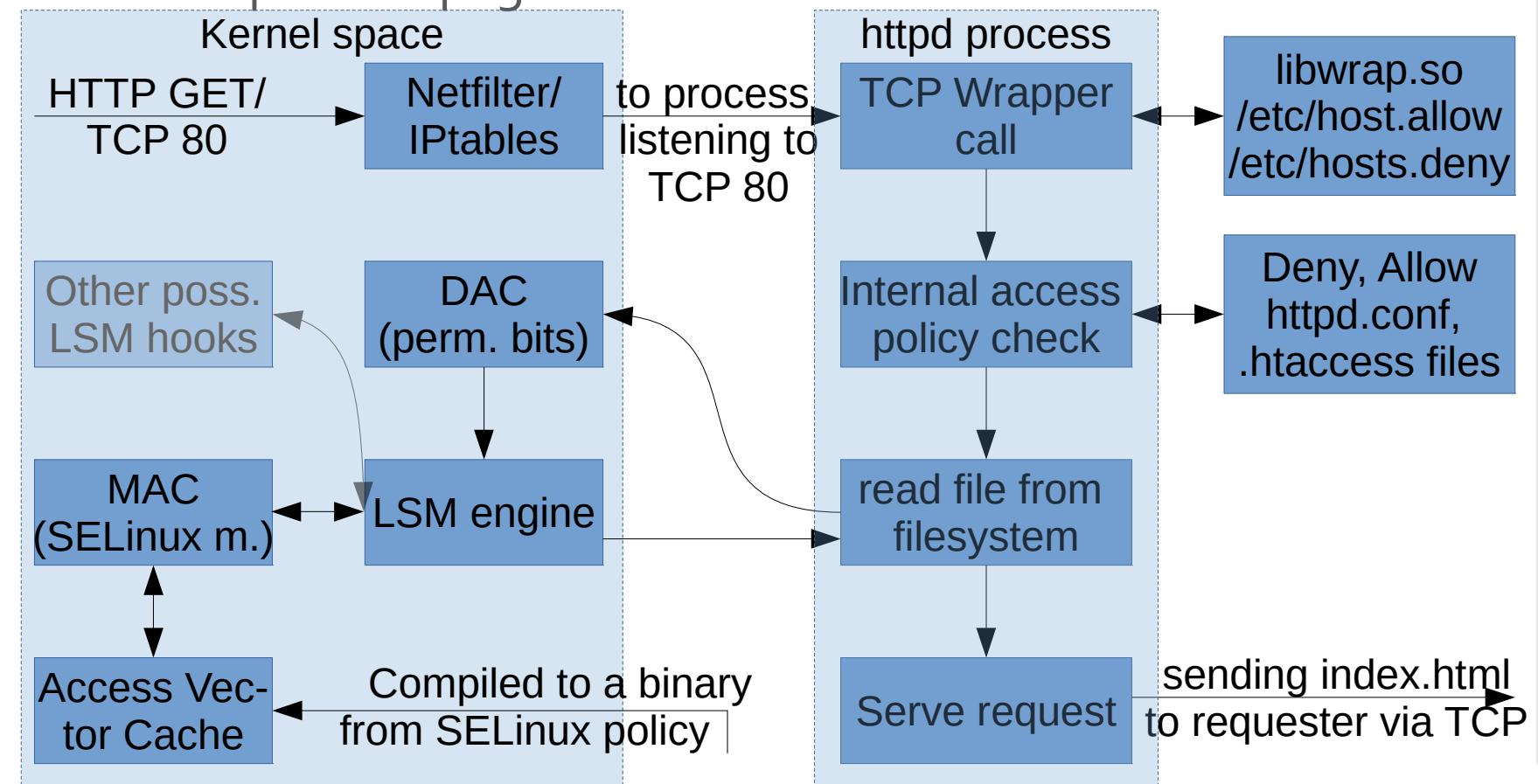
# SELinux

Lecture 4

Chapter 3

# General flow of security measures for network resources

- In Linux several checks are performed before a resource (e.g. file) is served to a requester. Consider this diagram depicting the flow of events when a user wants to access a simple web page over the network.



# SELinux

- All actions taken on the system which invokes Linux kernel calls are passed through LSM (Linux Security Modules) framework to decide whether a call is to be allowed or not.
- Security Enhanced Linux is an implementation of the US DoD style Mandatory Access Control model for Linux.
- SELinux has LSM hooks, therefore it takes part in the LSM decision process.
- It's based on several earlier projects of US NSA.
- The whole set of rules is called the SELinux policy. It can be quite complex, therefore it is compiled to a binary object (Access Vector Cache) for speed considerations.
- A given call's context is compared in the kernel against the AVC to rule either grant or denial.

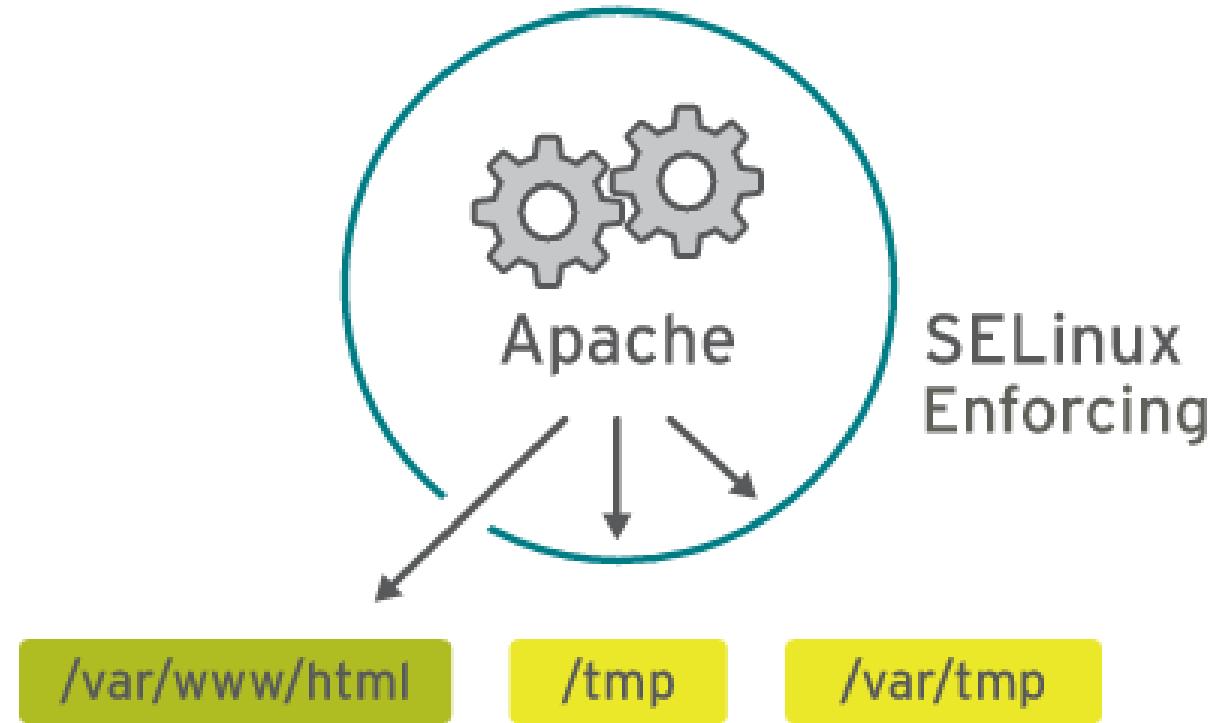
# SELinux context

- Every network port, filesystem object and process has an SELinux context.
- The mapping of context to FS objects is called labeling.
- The SELinux context involves 4 SELinux attributes:
  - **user**: SELinux user is independent of the Unix user model, though a mapping can be done. Most users and files belong to the same SELinux user.
  - **role**: a user is authorized by the policy for a defined set of roles. A role is authorized for specific set of domains. This is an intermediary layer, seldom used.
  - **domain (type)**: the most involved attribute in the targeted policy. (More on this later)
  - **level (category)**: a level of severity with an optional category specifier. Involved in MLS/MCS.
- Refer to RH134 CH 7.1 for more.

# SELinux policies

- There are several ways to structure the policy (ruleset). The most common ones are listed here. You can install a specific policy with the package manager. Throughout this course you'll **not** need to modify the policy itself.
  - **targeted policy:** this is the default in RH. It is most involved with types. An example rule translated to plain English should express something like this: allow httpd\_t type processes to access httpd\_log\_t type files for open and append operations.
  - **mls policy:** users and processes are subjects, labeled with a clearance level. Files and passive components are objects labeled with classification. (level and categ. attributes.) The ruleset defines the allowed data flows (e.g. write only, read only, both) between subj. and obj. of specific levels.
  - **minimal:** a reduced/simplified version of targeted.

# SELinux isolation concept



# SELinux modes

- There are two enable modes for SELinux and also a disabled setting. You can switch between modes at run-time, but can only enable or disable across reboots.
  - **enforcing mode:** all request are evaluated against the policy and result is forced in effect. Logging of AVC denials also happens.
  - **permissive mode:** all request are evaluated against the policy, but only logging happens in effect. Actual denial never occurs. This mode is only recommended for troubleshooting.
  - **disabled:** the subsystem is completely disabled, the SELinux kernel modules are not loaded at all.
- You can switch modes with the **setenforce** command and check with **getenforce**. The default mode and policy is set in the **/etc/selinux/config** file.
- Refer to RH134 CH7.3 and 7.4. for more.

# SELinux modes

- A reboot is required if you set to disable or choose to enable SELinux. The newly booted system will assume the settings in this file. The `selinux=` and `enforcing=` kernel boot time arguments override this file.

`/etc/selinux/config`

```
# This file controls the state of SELinux on the
# system.
# SELINUX= can take one of these three values:
#       enforcing
#       permissive
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

# SELinux booleans

- Some of the rules also evaluate binary state variables.
- Take the following example. Some systems admins need to enable the possibility of running cgi scripts under the web server because it's needed by some web applications, others for e.g. only allow static content for enhanced security.
- When a system administrator decides on this question, he/she may be tempted to modify the policy to allow/disallow this feature. With SE booleans the choice can be brought in effect without modifying the policy itself. Only the state of a variable has to be switched.
- The corresponding rule in the policy for this example would say: allow execute operation on `httpd_script_content_t` files for `httpd_t` type processes **if** the `httpd_enable_cgi` boolean is set to true.
- Refer to RH134 CH 7.7. for more, and complete CH 7.8.

# Get SELinux related information

- SE booleans can be used to fine tune the policy to make selective adjustments.
- You can list all the SE booleans with the **getsebool -a** command. You can set them with **setsebool**. You can use the -P switch for setting a value as permanent. (across reboots). Also note semanage boolean -l and -C.
- You can use the **sesearch -A | grep [keyword]** command to search the policy for expressions, types, rules. Use the -C -b switch in addition to restrict your search to SELinux booleans and print conditions.
- Try the **system-config-selinux** graphical tool for managing SELinux.
- Use the -Z switch in general to get SELinux context information. E.g.: **ls -Z**, **ps -Z**, **id -Z**
- You can use **sesearch -T** to look for transitional rules.

# SELinux useful tips

- Transitional rules define what attribute can be changed to what. E.g.: a root shell of `unconfined_u:unconfined_r:unconfined_t` type can transition to almost any context but not the other way around: a confined type cannot transition back to higher privileges.
- Use the **runcon** command to run a command with specific SELinux context. This command, like others also falls under the rules of transition. Use of runcon is only recommended for troubleshooting/debugging purpose.
- Use the **man -k selinux\_keyword** to search for man pages containing the keyword. E.g. a type, a boolean, ...
- Create a file in the root directory called **/.autorelabel** to automatically restore the default context of all files in the filesystem at boot time.

# Managing SELinux context

- Typically the location of a newly created file's context is determined through a way of applying defaults, which are described in a database. When moving existing objects to different locations the original context is preserved. This may be unintended in the new location.
- To explicitly set a file's context use the **chcon** command. Use the **-t** switch to only alter the type attribute. Use the **restorecon** command to reset a files context to the database defined, location dependent, default value. The **-R** switch stands for recursive, **-v** stands for verbose.

## Shell command line

```
# chcon -t httpd_sys_content_t /var/www/html/app1
# restorecon -Rv /var/www/html
restorecon reset /var/www/html/file1 context
unconfined_u:object_r:user_tmp_t:s0
-> system_u:object_r:httpd_sys_content_t:s0
```

- Refer to RH134 CH 7.5. for more.

# Managing SELinux context defaults

- To change the database of default file-contexts use the **semanage fcontext** command. Also note **semanage port** for network ports. Use the -l switch to list the current database. Use -d to delete a line, use -a to add a line. Note that the order of evaluation is the same as the order of entry (overlapping rules for subdirs).

## Shell command line

```
# semanage fcontext -l
...
/var/www(/.*)? all files sys_u:obj_r:httpd_sys_content_t:s0
...
# semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
# restorecon -RFvv /srv
```

- The database is stored under `/etc/selinux/targeted/contexts/files` directory within both text source files and binaries. If you edit the text file directly, it has to be recompiled. (`sefcontext_compile`)
- Complete practice RH134 CH 7.6. (required for 7.10.)

# Troubleshoot SELinux

- The most common reason for AVC denials is an incorrectly set file context. (E.g. file moved in a new location) → use **restorecon**. This has the most narrow impact on system security, it only affects the given file.
- In some cases a too restrictive policy can cause legitimate access intents to fail. → Look a related boolean with **getsebool -a** or **semanage boolean -l** and enable it with **setsebool**. This has a wider impact on the system.
- It can happen on very rare occasions that the policy installed by the package manager has bugs. → Add an allow rule with careful consideration. This can have a very wide impact on system security.
- Use **sealert -l [uuid]** command to get more details on a specific AVC denial. Its ID will be in the system log.
- Complete practice RH134 CH.7.10.-7.11 . (Do 7.6. first!)

# Recommended reading

Lecture 4

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH 134 CH7 (SELinux)
  - RH 124 CH7 (Manage linux processes)
  - RH 134 CH5 (nice)
- 
- RH 124 CH14 (Linux filesystems)
  - RH 134 CH9 (Partitions and filesystems)
  - RH 134 CH10 (LVM)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 4, PM-TRTNB219, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB219

Zsolt Schäffer, PTE-MIK

# Filesystems and swap

Lecture 5

Chapter 1

# Block devices

- Almost all data storage devices are block devices. An exception would be a tape backup drive, which is a character device, and does not have a fixed block size.
- Block devices can transfer data to be read or written one block at a time. The smallest addressable unit is a block. This is a hardware feature, not an OS limit.
- Blocks are identified by a numeric address, called LBA (Logical Block Address). The CHS (cylinder,head,sector) addressing is obsolete. In case of hard discs the blocks are also called sectors for the geometrical origin of the term.
- Typical block size for hard disks, USB flash drives, SSDs, memory cards is 512 bytes. Optical disks (CD, DVD, BD) have a block size of 2048 bytes. Hard disks manufacturers transition to the block size of 4096 bytes for their new/bigger drives.

# Block devices

- Block devices are usually named with a 3 letter and 1 number combination.
- The first two letters refer to the controller type: **hd** for IDE hard disks, **sd** for SATA and SCSI disk (note that USB attached storage also uses the SCSI protocol), **vd** for disk controllers in a para-virtual computing environment. Optical drive's device names start with **sr**.
- The third letter identifies the disk on the given controller in an alphabetical sequence. For example **/dev/sda** denotes the disk plugged in the first SATA port on the controller, **sdb** denotes the second one, and so on. Optical disc drives are identified by a number: **/dev/sro**, **/dev/sr1**, etc...
- The a last character is a number, that refers to the partition number on the disc (if it is partitioned). e.g. **sda3**

# Filesystems

- A block device itself does only have sequentially numbered compartment units (blocks). No structuring is possible. Block devices are often referred to as raw storage.
- A filesystem makes it possible to organize, manage, search and access data (files) in a comprehensible manner. This usually involves a hierarchical structure called the directory tree.
- A local filesystem is created on top of a raw block device. Clustering and network FSs are not considered in this lesson (ocfs, nfs, smbfs).
- A filesystem consist of a set of fixed conventions regarding storage format, allocation algorithm, addressing, organizing raw units, etc. Many filesystem formats are open and standardized, like for. e.g.: UDF, ISO9660, FAT<sup>?</sup> and most of \*nix FSs.

# Filesystems

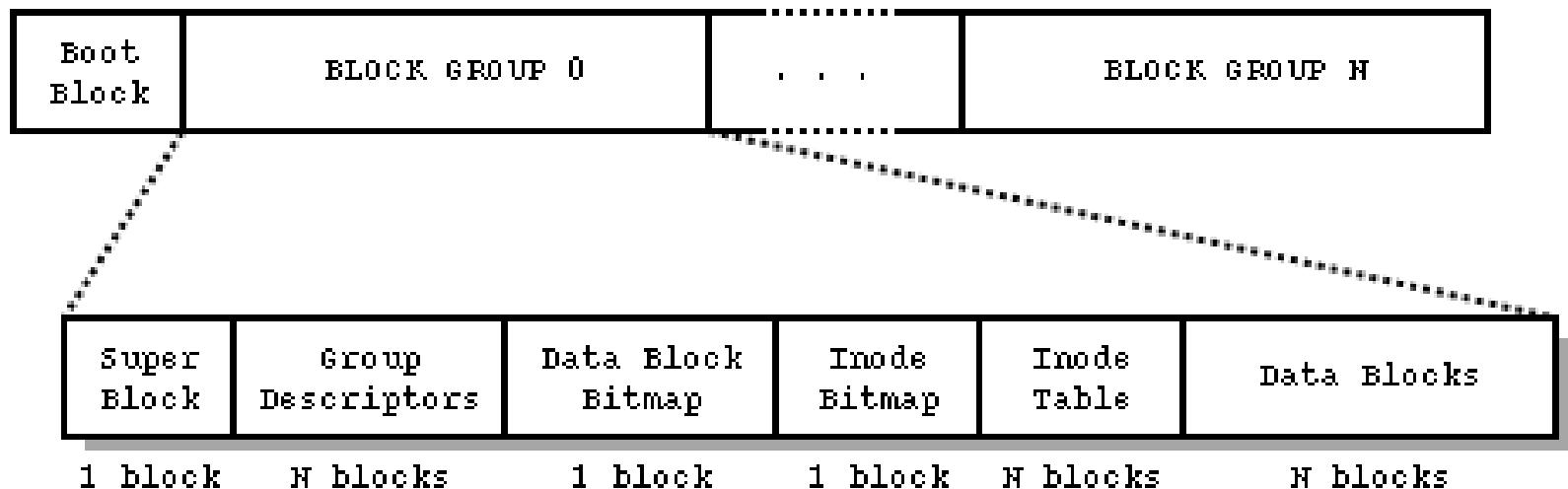
- Linux can support a large set of filesystems, including proprietary Microsoft (ntfs) and Apple (hfs+) FSs.
- There are a lot of filesystems native to Unix like OSes, e.g.: ext2/3/4, xfs, zfs, btrfs, reiserfs, jfs, f2fs, ...
- Each FS has a different set of features and a different way of implementing them. Only the extended filesystem and its versions (2/3/4) will be discussed in detail during this lesson.
- Virtual filesystems (/sys and /proc) are not actually stored on disk blocks. They represent internal kernel structures, states, conditions, settings. The values and texts contained in these files are either stored in memory or are calculated/formatted output from internal kernel variables. Some virtual files are read-only. Some of them are writable and influence the kernel's behavior directly.

# Filesystem features

- Some important filesystem features are:
  - Online shrink (jfs, ntfs\*)
  - Online grow capability (almost all, incl. ntfs)
  - Sparse file support (almost all, incl ntfs, udf)
  - Journaling (ext3/4, jfs, ntfs, many others)
  - Snapshotting (zfs, btrfs, ntfs\*)
  - Subvolumes (zfs, btrfs)
  - Multi device support (zfs, btrfs)
  - On-the-fly checksumming (zfs, btrfs)
  - Data compression (zfs, btrfs)
  - Quota (almost all)
  - Trim/Discard support (ext4, btrfs, ntfs, ...)
- UDF supports both Unix permissions and Windows file attribute bits.

# The ext2 filesystem

- The filesystem's smallest addressable unit is the FS block, which is not the same as the raw device block.
- The FS block size determines the maximum FS size and maximum size of a single file. For regular sized FSs (from a few GiBs to a few hundred GiBs) the block size is usually 4k bytes.
- The blocks are further organized into block groups.

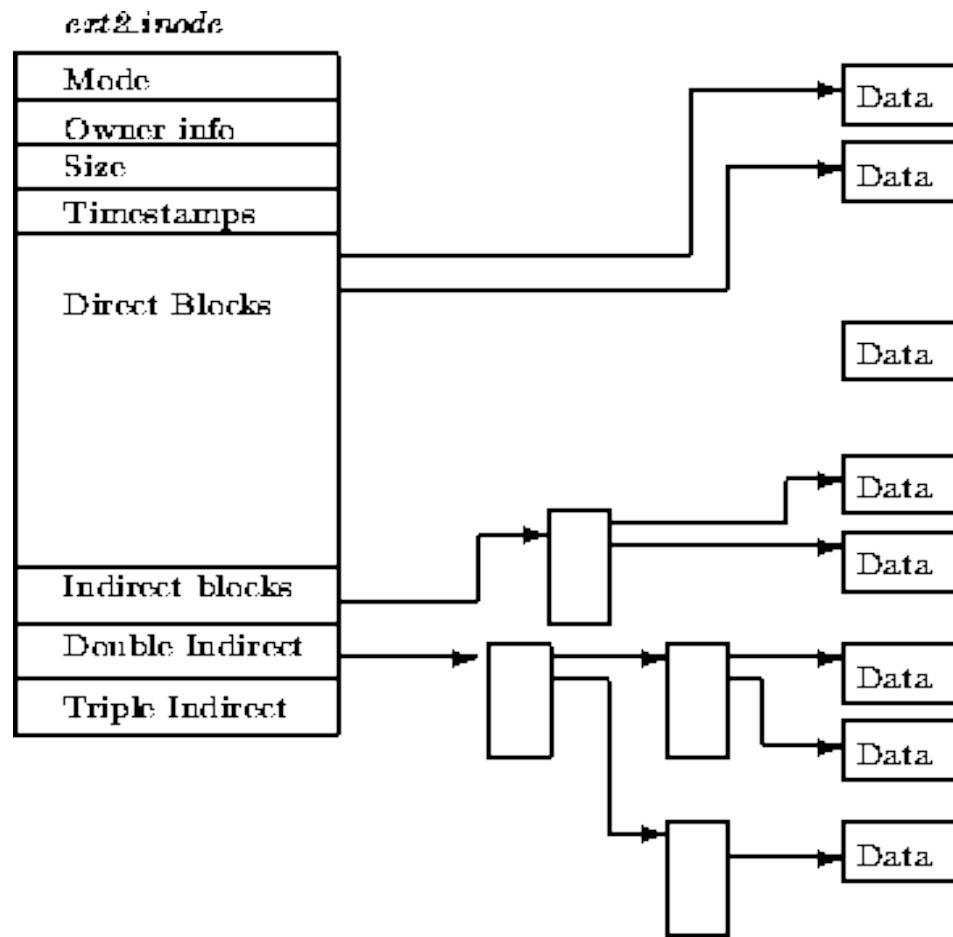


# The ext2 filesystem

- The superblock contains information about the entirety of the FS (block size, first data block, no. of free blocks, etc.). There intentionally are multiple instances of the superblock scattered around the FS.
- The group descriptor describes the location and length of all the block groups in the FS (the location of the bitmaps and inode tables can be calculated from this).
- The bitmaps contain a one bit per block representation of all blocks: indicating whether free or allocated.
- The inode tables contain a predefined number of inode blocks. The index node describes all the file properties and metadata except for the file's name. (file size, owner, perms, atime, mtime, ctime, link count, flags, ...) It also holds the list pointers to data blocks associated.
- Data blocks hold the actual contents of the files.

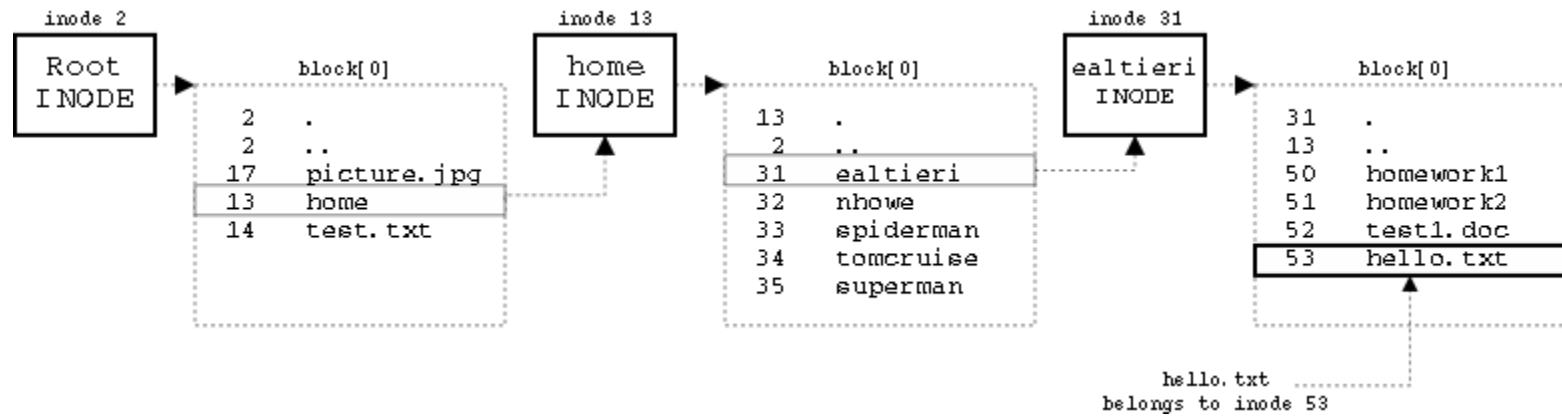
# The ext2 filesystem

- In case the list of data blocks can't fit in a single inode, one to three levels of indirection blocks are inserted.



# The ext2 filesystem

- Directory inodes contain a list of lines (records), one for each entity in the directory. A record is a mapping between filename and numerical inode identifier.
- Each record contains 5 fields: inode number, record length, filename length, file type (pipe, symlink, socket, chardev, blockdev, directory), and the filename itself.
- Remember lecture 3! The directory permissions (rwx) now all make sense. Manipulating a directory is the same as modifying or accessing this list of records.



[http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2\\_locate.png](http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2_locate.png)

# Links

- Hard links are just a name for the case, when the same inode number is pointed multiple times with different names and/or from different directory nodes.
- Each reference lives independently, therefore they are stable and not exposed to renaming/removal of the other references. The inode has a link counter to track the number of references. If it hits 0, the blocks related to the file can be unassigned (freed) and reused later.
- The scope of hard links is confined within each individual filesystem.
- Symbolic links are textual path pointers to any other type of filesystem object.
- Moving a referenced file away from its original path turns the symlink pointing to it into an 'orphan' link.
- Symlinks can have a scope outside (across) individual FSs.

# Evolution of ext FS

- The extended FS (ext) was designed by Rémy Card, implemented in 1992. Could handle storage up to 2GB.
- The ext2 was developed one year later by the same person. Remedied fragmentation issues, and could support FS sizes of 2TiB to 32 TiB and file sizes of 16GiB to 2TiB (depending on block size).
- The ext3 was introduced in 2001 by Stephen Tweedie. The most important addition was the journaling feature. The format is directly convertible from ext2.
- The ext4 FS was introduced in 2008. It has discard support for SSDs. Has increased filesize limit of 16 TiB and FS size limit of 1 EiB. The most important feature additions were not related to storage format, they were algorithmic, like e.g. delayed allocation, journal checksum, etc. → more performant and reliable. Also note that existing ext3 can be mounted directly as ext4.

# Mounting

- Linux works with only one filesystem tree. The root filesystem has an emphasized role and is activated during the boot process. The starting point of the root filesystem is the root directory, its path is denoted with a single / (slash). By default the FS tree is shared among processes (it is the same for all of them), but it is possible for each process to have a different view of the tree (Look up mount namespaces).
- Any additional filesystem tree can be merged in the existing tree structure starting from a freely selectable point (directory). This is called mounting. The reverse operation is unmounting, which is only possible if no files are being held by processes in the given subtree.
- Use the **mount** command to mount a storage device that contains a FS to an existing directory in the tree. Use **umount** to remove the subtree corresponding to the FS.

# Mounting

## Shell command line

```
# mount -t vfat -o ro /dev/sdc /mnt/usbflash
# umount /dev/sdc
# umount /mnt/usbflash
# mount -B /home/teamleader/work /srv/team
```

- The **-o** switch allows for specifying mount options (not mandatory). Many options are FS specific (like noatime, nospace\_cache, discard, ...) some are common (like **ro**).
- The **-t** switch allows for specifying FS type (format). If omitted all known FSs will be tried until one succeeds.
- When un-mounting either specify the mount point or the block device, but not both.
- You can replicate a part (subtree) to another point in the tree by doing a bind mount. Use the **-o bind** or simply **-B** switch to achieve this.

# Mounting

## Shell command line

```
# mount -o loop dvd.img /media/disc_image
```

- Use the `-o loop` or `-o loop=/dev/loop4` option to automatically assign or manually specify a loop device for image files before mounting. In this case the `mount` command will call the `losetup` utility to set up a loop device for convenience, and then mount the FS of the loopdevice on the given directory.
- In the given example the FS type will be probed for, but you could specify `-t isog660` or `-t udf`.
- You can also have raw hard disk images mounted this way, even in read-write mode. Some image formats require, that you specify an offset.
- You can use the `losetup -d /dev/loopN` to remove the Nth loopback block device after un-mounting it.

# Creating filesystems

- You can create a filesystem with **mkfs**. The **-t** switch selects the filesystem type (e.g. btrfs, ext4, xfs, jfs). Also note the command aliases like **mkfs.btrfs**, **mkfs.ext4**, etc.
- The **-L** switch usually sets the new FS's freely adjustable text label.
- Most other switches are FS type specific. Look for man pages for each **mkfs.[fstype]** command.
- After the switches you have to specify the raw block device to create the headers and initial FS structures on.

## Shell command line

```
# mkfs -t ext4 -L "Bobs portable drive" /dev/sdc
```

- At the time of creation a random identifier is generated (UUID), which will uniquely identify the FS. You can also specify the UUID instead of the block device's name for the mount command.

# Managing filesystems

- You can use **blkid** to print the UUID of all the available FSs. This will also list the IDs of FSs that are not mounted.
- You can use the **df** command to view the free space in all mounted FSs. Note **-h** switch for human readable units.
- You can list all the files being held open by processes with **lsof**. You can narrow down to a subtree (a directory).
- Give the **fuser** command to list processes using a specific file. It's recommended to turn on the **-v** (verbose) switch.

## Shell command line

```
# lsof /run/media/joes_portable_drive
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/loop0       18G   15G   2.5G  86%   /
...
/dev/sda3       167G  157G   9.9G  95%  /host
```

# Managing filesystems

- Use **du** to show the disk usage of a single file or all the elements in a directory. Use the **-h** switch for human readable units. Use **-s** for directories to also print the sum of space usage.
- When running Gnome, removable drives will automatically mounted by default. The mount location is `/run/media/[username]/[FS_label or UUID]`. Where `username` belongs to the user logged in to the desktop session.
- Use **fsck.[fstype]** utility or **e2fsck** for ext2/3/4 filesystems to run a check. Most filesystems can only be repaired/ scanned when offline (not mounted).
- Use **resize2fs** to adjust the size of an ext2/3/4 filesystem structures for a specific FS size. Usually online growing is possible, but shrinking works only offline, if at all.

# Managing filesystems

- You can make a hard link to a file with the following:

Shell command line

```
# ln /some/existing/file /to/new/location/filename
```

- Note the link counter (second column) in **ls -l** output. Use the **-s** switch to create a symbolic link instead of a hard link. Links can be deleted with **rm**, just like any file.
- You can use **sync** to flush all the write caches to disc. The **umount** command will call that automatically for you.
- A filesystem of **tmpfs** type is technically a RAM drive. It's not backed by storage blocks, but by memory pages. There is **mkfs.tmpfs**, rather use **mount -t tmpfs**.
- Use the **tune2fs** command to set FS parameters for **ext2/3/4**, like **fsck** scheduling, label, journaling, ...
- Complete practice RH124 CH 14.6. and 14.9.

# Swap

- Swap space is a virtual extension of the memory addressing space. In case the system's memory requirement is larger than the physical amount installed, the kernel memory management subsystem will involve supplementary memory pages, that are actually stored on the disk instead of physical memory.
- The kernel continuously monitors memory page access and moves the least frequently used pages to the much slower disk storage and tries to keep frequently accessed pages in physical memory. Moving pages in and out of physical memory is called swapping.
- The area used for swapping can either be a regular file or a raw block device with a swap header. Swap space is intentionally not called a type of filesystem, since it doesn't hold files or permanent data at all.
- Recommended size of swap is 1-2x the size of memory.

# Managing swap

- You can create a swap header on top of a block device with **mkswap /dev/block\_device** or inside of an existing file with **mkswap /some/regular/file**.
- Use **dd** to create a file of specific size. Example for 1GB:

Shell command line

```
# dd if=/dev/zero of=swap.file bs=1M count=1024
# mkswap swap.file
# swapon swap.file
```
- You can activate a swap space with **swapon filename**, deactivate with **swapoff filename**.
- You can check the amount of total and free swap space along with physical memory usage by issuing the **free** command.
- You can print the priority of each enabled swap space with **swapon -s**.
- Refer to RH134 CH 9.3. for more on swap.

# /etc/fstab

- The root FS will mount during the boot process. The UUID or the block device containing the root FS is passed to the kernel by the bootloader as an argument. (Further in Lecture 6).
- All the other block devices, that are required to mount automatically at system startup must have a corresponding entry in the /etc/fstab file.
- The file format is the following:

## /etc/fstab

```
# <file system> <dir> <type> <options> <dump> <pass>
/dev/sda1      /      ext4  defaults,noatime 0      1
/dev/sda2      none   swap  defaults        0      0
/dev/sda3      /home  ext4  defaults,noatime 0      2
```

- The dump column is obsolete, the pass column determines the order to run fsck at boot time. 0 means no fsck, root FS has always first pass, all the other FSs have 2 or more in this column.

# Partitioning

Lecture 5

Chapter 2

# Partitions

- Partitioning is a means to logically divide one physical block device to separate contiguous sections.
- Partitioning is a linear mapping of coherent, contiguous ranges of physical blocks (sectors) to logical blocks with an offset addition. Partitions are forbidden to overlap!
- The sector size of the logical device is the same as of the physical one.
- On x86 PCs two partitioning schemes are common: the MBR and the GPT scheme. There are other schemes for different architectures and computers.
- Hard disks have at least one partition defined. In Microsoft OSs removable flash drives and memory cards usually either have no partitions at all (superfloppy) or have only one partition defined, that covers the whole space. Linux comes with no such restrictions.

# Partitions

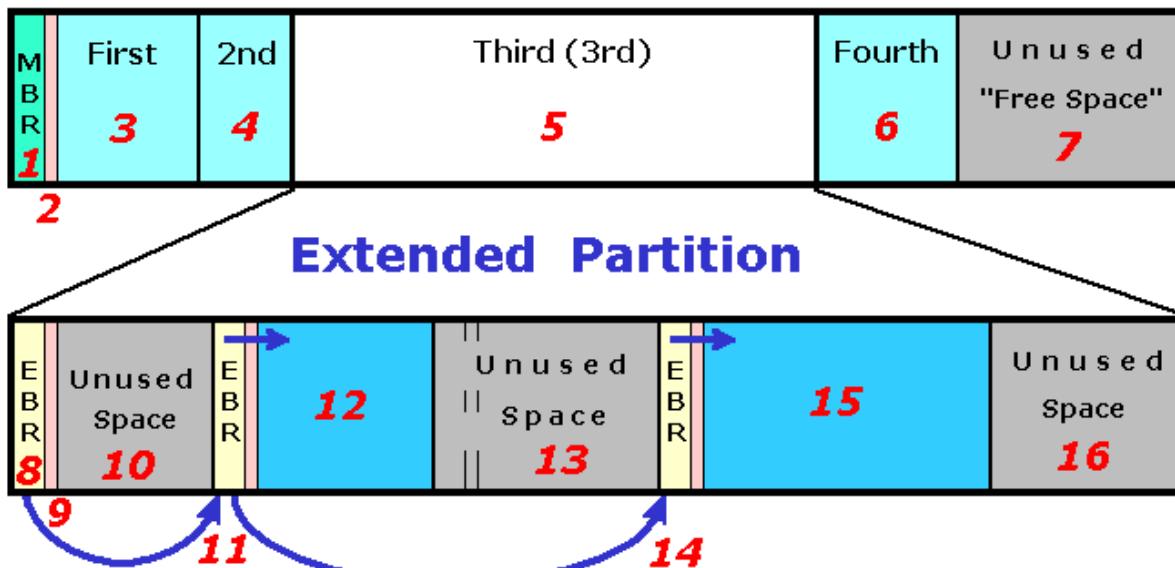
- One could have many reasons to partition a drive.
  - Multiboot: to install more than one OS, that have different native FSs. The filesystems need to be created on isolated block ranges with no overlap.
  - Backup schemes: If some part of the filesystem changes infrequently (e.g. system directories) and some changes often (e.g. user home folders) it might be a good idea make a backup separately. Partitioning facilitates that.
  - Performance: The inner rings of a spinning disk (end of sector range) can be read/written slower than outside rings (start of sector addresses). It might be a good idea to put swap space on the faster part of the disk. It would also be a good idea to put /var (frequently varied data) on a different partition and different disk from swap, since they both can be I/O intensive.
  - System reliability: E.g.: without partitions (one single FS) a user consuming all the free space could prevent the logging daemon from append entries to the log file.

# MBR partitioning scheme

- The MBR scheme is also called legacy or msdos partition table. It is stored in the 0<sup>th</sup> block of the disk, along with a short snippet of executable code (440 B)
- This type of partition table is 64 bytes long, can hold up to 4 entries, 16 bytes each. The LBA of both start and end sectors is represented by a 32 bit unsigned integer. This limits the max. addressable space:  $512 \times 2^{32} = 2 \text{ TiB}$ .
- Each partition has a 1 byte type code. It usually indicates purpose or filesystem type, but in practice FS headers are probed to determine the type. This byte is mostly ignored.
- 4 possible entries (primary parts) can prove to be too few. → There is a special type of partition called extended partition. It takes up 1 place in the table, and serves as a container partition. It can hold an arbitrary number of logical partitions, stored as a chained list.

# MBR partitioning scheme

- There is byte for each partition containing flag bits. The most important flag is the active flag otherwise called the boot flag. Only one partition is allowed to have this flag at a time. The standard Microsoft MBR executable boot code looks for the partition with the active flag set, reads its 0<sup>th</sup> sector (PBR) into memory and continues execution from there.



Copyright(C)2007 by Daniel B. Sedory

<http://thestarman.pcministry.com/asm/mbr/EBRreality.png>

# MBR limitations

- 4 primary entries are too few and many OSs can't boot from logical partitions.
- Some OSs can't boot past the 1024 cylinder boundary due to addressing restrictions of real mode BIOS.
- More than one extended partitions is not allowed.
- Disks larger then 2 TiB can only be used as 2 TiB drive, since there is no way to define an ending address larger than that.
- There is only one instance of the partition table. In case it gets corrupt, the whole disk has to be scanned for filesystem header signatures.
- The chain linked list is prone to corruption. In case one link gets broken, the rest of partitions are lost.

# The GPT partitioning scheme

- The GPT partitioning scheme is tightly bound to the introduction of UEFI firmware replacing the IBM PS/2 originated BIOS firmware.
- GPT disk have a **protective MBR** record covering the whole disk (or a 2 TiB range). The next sector contains the **GPT header**. The disk's GUID identifier and the location and size of the primary table are defined here.
- The **primary GPT table** has a flexible size and therefore can have any number of entries. The default size is 16k bytes, which allows for up to 128 entries.
- An entry contains a 64 bit start LBA address, 64 bit end LBA, a 128 bit type identifier, a 128 bit partition GUID, 64 bits for flags, and a 72 byte UTF-16LE text name.
- The active flag signifies the EFI system partition, which has a significance in the UEFI boot process.

## GPT benefits

## Partition numbering

- The primary header and primary table is repeated at the end of disk as a **secondary** instance for safety.
  - There are no extended or logical partitions in this scheme. Practically any number of primary partitions are possible.
  - The address limit of GPT is more than HUGE.
  - All new OSs support GPT, but BIOS computers can't boot from a GPT disk directly.
- 
- Partitions are numbered from 1 in Linux. The first logical partition (if present) always has the number of 5. For e.g.: `/dev/sdc5` is the device file for the third SCSI disc's first logical partition.

# Managing partitions

- Use **lsblk** to show all block devices in a tree view. Use **partprobe** to force the kernel to re-read the partition table after modification.
- Use the **fdisk** utility to create and manage an MBR table. Use **gdisk** for the GPT scheme.
- Note the command keys: **n** for creating a new partition, **d** for deleting, **p** for printing, **t** for changing type, **w** for writing modifications to disk, and **q** for quitting.
- Also note the menu driven **cfdisk** tool.
- Creating a filesystem on top of a partition works the same way as for any other block device.
- Refer to RH134 CH 9.1. for more.
- Complete practices RH134 CH 9.4. and 9.5.

# Logical Volume Management

Lecture 5

Chapter 3

# LVM

- The device mapper framework is a kernel layer that allows for advanced mapping of block device sectors to high level logical block devices.
- It is the base of many features like for e.g.: block level encryption (dm-crypt), block level hash verification (dm-verity), hybrid SSD-HDD storage (dm-cache), etc...
- The logical volume manager is the most well known target to the device mapper framework.
- LVM allows for defining and resizing raw storage volumes at run-time. It can also provide software RAID functionality of any kind. It also has features like cache pools, thin volumes, block level snapshots for facilitating backups, and more. It is also useful to extend storage volumes as needed with new drives (without the need to copy existing data from the old drives to big new ones).

# LVM

- LVM consists of 4 components (layers).
- The PV (physical volume) can be any block device, typically a partition or a whole disk. By marking block device as a PV they will be involved in LVM. The sectors are organized into contiguous groups by LVM, called extents. The extent size is always an integral power of 2, it defaults to 4 MiB. This will define the management granularity of volumes involved, but not the block size. When creating a PV all its extents are free.
- VG (volume group) is an organizational unit to separate LVM namespaces. VGs can't borrow extents from each other, they are completely isolated. A VG can have any number of PVs, all PVs can only belong only to one VG at a time. The entirety of extents in all the PVs of a VG together add up the available space of the VG.

# LVM

- An LV (logical volume) is carved out of the free extents of one VG. The filesystem is created on top of the LV.
- LVs can be resized on-the-fly if there are enough free extent available in the VG. PVs can be removed from a VG on-the-fly if there are enough free extent remaining. Block level relocation is done automatically by LVM, while the volume is still mounted and operational.
- An additional layer (called pools) is only required by certain types of LVM volumes. You can add extents from a VG to a thin pool to later on carve thin volumes out of it. Or you can add extents to a cache pool to later on carve accelerated block devices out of the pool.
- LVM can also arrange the extent allocation on the PVs to provide redundancy and/or speed boost, if there is an appropriate number of distinct physical drives present in the VG (soft RAID).

# PVs

- You can create a PV with **pvcreate blockdev\_1 blockdev\_2 blockdev\_n** command. Simply list 1 or more block devices after the command, to initialize them as PV.
- You can remove PV metadata from a block device with **pvremove blockdev\_1 blockdev\_n** if you do not want it to be involved in LVM any more.
- You can display PVs with **pvs**. Note the the **-o** option to select columns to display and **-O** for selecting the sort field.
- Use **pvdisplay** to display detailed PV information, including UUID, number of used and available extents.

# VGs

- To create a VG from PVs use **vgcreate -s 2M vg\_name pv\_1 pv\_2 pv\_n** command. The first argument is the VG name, all the remaining arguments must be initialized PVs. All of them will be assigned to the newly created VG. Note the **-s** switch to specify the extent size. If omitted it will default to 4 MiB. It's conventional to name VGs with a **vg\_** prefix.
- Use **vgchange -s** to change the extent size later. Use **vgchange -a n VG\_name** to deactivate a VG at runtime, while **-a y** switch will activate a whole VG at runtime.
- Use **vgs** to list or **vgdisplay** to detail existing volume groups.
- Use **vgreduce vg\_name list\_of\_pvs** to remove one or more PVs from a VG. Use **vgextend vg\_name list\_of\_pvs** to add free PVs to the existing volume group.

# LVs

- To create an LV from a VG consider this example:  
**lvcreate -n lv\_swap -L 8G vg\_chemlab**  
The -n switch specifies the name of the new volume, -L specifies size in binary units, while -l specifies size as number of extents. The last argument is always the VG name to carve the LV out from.
- Use **lvchange -a n lv\_name** to deactivate a single LV. Use -a y to activate, -p r or -p rw to set read-only or read-write permission.
- Use **lvs** to list or **lvdisplay** to detail logical volumes.
- Use **lvresize -L 200G /dev/vg\_name/lv\_name** to enlarge or shrink a logical volume. You can specify an absolute new size a relative size with for. e.g. -512M or +40G. You can also use -l to specify a relative or absolute value in number of extents.

# Managing LVs

- LVM devices are presented as `/dev/dm-[number]` files. There are two symlinks to each of the device nodes: `/dev/mapper/vgname-lvname` and `/dev/vgname/lvname`. You can refer both when creating a filesystem or swap space on top of the LV, they represent the same LV.
- When shrinking, you have to reduce the FS first. In case of ext2/3/4 use **resize2fs** `/dev/vg_some/lv_any [size]` to resize the FS structure. Ext2/3/4 FS can only be shrunk offline, but can grow online. In case of growing you grow the LV first, then give the `resize2fs` command without size argument to fill up to the length of the LV.
- You can use **lsblk** also for listing LVs and child-parent relations.
- Refer to RH134 CH 10.5. for more.
- Complete practice RH134 CH 10.7.

# LVM RAID

- When creating an lv with **lvcreate** you can specify raid personality with --type switch like --type raid0 or --type raid5. Use -i to specify number of stripes -l for stripe size, -m for number of mirrors (no. of disks -1).

## Shell command line

```
# lvcreate -i2 -l64 -L 465G -n stripe_vol  
vg_demo /dev/sdb1 /dev/sdc1
```

- It is possible to omit the list of PVs, they will be assigned automatically to independent physical media.
- The switches --type stripe and --type mirror are not equivalent to --type raid0 and --type raid1. The latter use mdadm algorithms, the former use legacy LVM raid algorithms.

# LVM cache

- Take the following example, where /dev/sde1 is a spinning disk partition and /dev/sdf1 is a fast SSD partition.

## Shell command line

```
# lvcreate -L 40G -n lv VG /dev/sde1
# lvcreate -L 2G -n lv_cache VG /dev/sdf1
# lvcreate -L 12M -n lv_cache_meta VG /dev/sdf1
# lvconvert --type cache-pool --cachemode
writethrough --poolmetadata VG/lv_cache_meta
VG/lv_cache
# lvconvert --type cache --cachepool VG/lv_cache
VG/lv
```

- First three linear volumes are created. Then the cache and cache\_meta volumes are combined to a cache pool, the pool will assume the name of the former cache volume. Lastly the slow lv volume will be converted to type of cache, and will be backed by the cache\_pool. It's SSD accelerated from now on.

# LVM thin volumes

- Thin volume's extents are dynamically allocated/unallocated from/to the pool as they are written/discard.
- It is possible to over-provision storage for virtualized services. This is useful for e.g. to cloud service providers.

## Shell command line

```
# lvcreate -L 100G --type thin vg001/mythinpool
# lvcreate -V 100G -T vg001/mythinpool -n thinvol1
# lvcreate -V 100G -T vg001/mythinpool -n thinvol2

# lvcreate -i 2 -I 64 -L100G -T vg00/pool -V 1T -n
thin_lv
```

- First a pool is created, than thin volumes are allocated using extents from the thin pool. The last command example shows how to create a thin pool and a volume inside it in one step. It also demonstrates that striping is also possible for thin volumes.

## LVM thin volumes

## LVM snapshots

- You can use **fstrim** to force the FS to discard unused blocks. This is useful for SSDs, and also for thin volumes. Extents known not to be needed anymore can be un-allocated, and put back to the available pool.
- The pool actually consists of a simple linear storage and a metadata volume, just like with cache. The command demonstrated creates both in one step.
- Note that thin volumes can become fragmented on disk.
- You can create a block level snapshot of any volume. E.g.

Shell command line

```
# lvcreate -L500M --type snapshot -n dbbackup  
/dev/ops/databases
```

- If you specify the size for a snapshot of a thin volume, it will be created as regular snapshot, otherwise thin volumes have thin snapshots.

# LVM snapshots

- Regular snapshots have a size limit. When modifying the origin volume, the modified extents are copied to the snapshot volume. So both the original and current states are preserved somewhere. The size limit restricts the amount of preservable extents.
- You can discard a snapshot by deleting the snap volume. You can roll back a volume to the saved state by merging it into the origin volume.

## Shell command line

```
# lvconvert --merge /dev/ops/backupdb
```

- You can create several snapshots of the same volume, but leveling this has a serious write performance impact, since all the snapshots have to be updated on writing.
- Extents of thin snaps are simply COW-ed to an available extent, only metadata versioning happens, which is fast.

# Recommended reading

Lecture 5

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH 124 CH14 (Linux filesystems)
  - RH 134 CH9 (Partitions and file systems)
  - RH 134 CH10 (LVM)
- 
- RH 134 CH16 (Boot issues)
  - RH 124 CH8 (Controlling services)
  - RH 124 CH13 (RPM, yum, package management)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 5, PM-TRTNB219, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# Package management

Lecture 6

Chapter 1

# Version management

- Files in the Linux filesystem layout are grouped by function. E.g. all documentation information goes to /usr/share/doc folder, all config files to /etc, all libraries to /lib and /lib64. All binaries go to /bin and /sbin. These folders are also included in the \$PATH variable.
- It would be easier to group files by program if there weren't any shared components, like .so libraries. So both solutions have their drawbacks.
- Simple extraction of installable software to the filesystem is hard to maintain. It would be difficult to track what to remove or update in case of a new version.
- A package manager consists of a database and a set of tools for tracking and managing software, updating versions and config files. Using a package manager suit is a possible solution for the above problems.

# Distributions

- A distribution is an assorted set of packages (usually within an installation media.) Distributions contain a specific version set of software composed together into a whole usable Linux system. Distribution is the 'unit' of releasing and handing out a specific Linux compilation to customers.
- There are a lot of Linux flavors, most of them have multiple release versions. They are all considered separate distributions, and usually have a unique nickname, like for e.g. Debian Wheezy or Fedora Heisenbug.
- Distributions are usually only supported in a limited time window, then they will get obsoleted/abandoned. An exception would be for e.g. the Arch Linux distro, which has only one 'continuous' release.

# Package managers, formats

- The image file of the installation media for most Linux distributions are downloadable free of charge.
- The package manager is specific to a given flavor of a Linux distributions. There are two major families of Linux systems, the most important difference between them is the package manager and package format used.
- Debian family (e.g. Ubuntu) uses the deb package format and the dpkg package manager, usually with the apt front-end.
- Red Hat family (e.g. Fedora, Novell, SUSE, Oracle Linux) uses the rpm format with the rpm package manager, usually with the yum or dnf as front-end.

# Additional package managers, formats

- The Gentoo distribution uses an exotic package mgmt. system, called emerge. All packages are compiled from source on the users computer, to absolutely make sure no unintended code is executed on the computer. No binaries are copied to the computer.
- Linux distros for embedded devices (e.g. OpenWRT) have the opkg/ipkg format and manager.
- Arch Linux's package manager is called pacman. This distro is exotic in the following way: There are no isolated distribution versions with separate support periods. There is only one streamline distribution, all packages are continuously updated. This is called a rolling release.
- Package formats can be converted, but this by itself does not resolve dependency problems, and library compatibility issues across different Linux distributions.

# Package formats

- A package is not executable by itself, it needs the management application.
- One package manager handles all the packages on the system.
- Packages have a standardized format.
- There is one consistent database that holds all the installation information, like for. e.g. which file in the FS belongs to which version of which package.
- The package manager is a distinctive part of a Linux distribution.
- A package consists of a payload (installable files, compressed), executable scripts (they run on install/ update/remove operations) and metadata (package info, version, description, signature, etc.).

# RPM packages

- RPM pkgs work based on file dependencies. DEB pkg. mgmt. considers package dependencies. Two different packages can't have the same file installed in RPM.
- RPM packages can be digitally signed to prevent unauthorized/maliciously modified packages from being installed.
- RPM package database is handled by the Berkley DB engine.
- RPM naming convention:  
`<name>-<source version>-<release>-<arch>.rpm`  
Where version signifies the source code the package was compiled from, release is an incremental tracking number of the package maintainer (e.g. apply patches, recompile) and arch is either *noarch* or signifies a specific CPU architecture, the package content binaries are compiled for, for. e.g.: `x86_64`

# Repositories

- A repository is an aggregated, grouped set of packages available for installation.
- Repositories are the main source of packages for the package manager.
- A repository is usually accessed through `http://` or from a filesystem directory (mount point).
- All packages inside the same repository are bound to be compatible and form a clean and consistent dependency tree (Look up DLL hell). All binaries inside a repo are compiled against the same set of libraries. Only the library versions, which are ABI compatible (function call signatures), are to be found in a repository. There are no conflicts in version requirements inside the boundaries of a repo. All the linked libraries themselves are part of the repository (maybe except for 3<sup>rd</sup> party repos).

# Repositories

- The RH family of distributions have two repositories enabled by default:
  - One release repository, which contains the package versions of the date of the distribution release. This is the same as the set of packages contained in the installation media. This repo does not change through the life cycle of the distro.
  - One updates repository, which will only contain the most current versions of packages. This changes frequently during the distro's life cycle.
- Third party repositories can be added to the package manager, but be aware of any possible dependency issues before doing so. There are several semi-official, well maintained and compatible 3<sup>rd</sup> party repositories for the RH family of Linuxes: RPMFusion, EPEL, Livna

# Repositories

- It is common to move free-of-charge, but non-open source packages to a different repo for legal reasons.
- It is also common that software vendors put their programs, plugins and libraries to a repository to make a certain product available for customers to install. For e.g. there is repository for Adobe products, one for Oracle's VirtualBox, one for Google's Chrome/Earth/ Picasa,...
- You can enable a 3<sup>rd</sup> party repository by downloading its descriptor file from the maintainer, and setting the enabled flag.
- Red Hat repository access requires that you have a valid active subscription for your registered installation instance. Look up **subscription-manager** in **RH124 CH 13.1.** for more details.

# Repositories

- The repository descriptors are to be found under **/etc/yum.repos.d** directory. Here follows an example:

```
/etc/yum.repos.d/fedora.repo
```

```
[fedora] (< This is the repository's name)  
name=Fedora $releasever - $basearch  
metalink=https://mirrors.fedoraproject.org/... [cut]  
enabled=1  
metadata_expire=7d  
gpgcheck=1  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-... [cut]  
skip_if_unavailable=False
```

- You can list all repos by issuing the **yum repolist all** command. You can enable/disable a repo permanently by **yum-config-manager --enable/--disable [reponame]** command, or by editing the enabled line manually.
- Temporary setting can be done with **yum's --enablerepo =pattern** or **--disablerepo=pattern** switches.
- Complete practice **RH124 CH 13.8**.

# yum command

- Yum is an easy to use front-end to the rpm manager. You can use it to install/remove/update/search packages.
- By issuing **yum list** or **yum list installed** you can list all the available/installed packages.
- You can search for patterns in both package name and description text with **yum search all 'keyword'** command. To search for file (remember: a Linux command is a file in /bin or /sbin) use **yum provides \*filename** or **yum provides /exact/path/to/file/name**
- Issuing **yum update *packagename*** will update a package to the latest version that has been made available in the repo. Omitting *packagename* will update all packages.
- You can use the **-y** switch to skip the confirmation on doing a yum operation.

# yum command

- Use **yum install *packagename*** to install a package from the set of enabled repos. Use **yum localinstall *some-file.rpm*** to install from an rpm from a file in the file-system. Use **yum remove *pattern*** to erase one or more packages. This will also remove packages that depend on the removed package(s).
- Issuing **yum info *package*** will show the detailed description of the package.
- Some software suits are organized into groups, for e.g. "Development tools". You can do group operations by using the group sub-command: **yum group list**, **yum group info *grpname***, **yum group install *grpname***, etc..
- Yum has a history feature: **yum history [*nr*]** will print details of a transaction defined by *nr*. Omitting *nr* will list all recent transactions sorted by tr. number. You can roll back a transaction with **yum history undo *nr*** command.

# rpm command

- You may use the **rpm** command for low level interaction with the package manager. The yum front-end actually executes several rpm commands behind the scenes.
- Only the query commands will be discussed in this course. You can use the **-q** switch to rpm to select query mode.
- The **rpm -q -a** command will list all installed packages, **-q -f *filename*** will find what package holds *filename*. Use **-q -i** to display package info, **-q -l** to list files in a package, **-q -c** to list config files only, **-q -d** to list documentation files only. The **-q --changelog** switch will print the version history and comments of a package, **-q --scripts** will show the executable scripts in the package.
- Use **rpm2cpio *file.rpm* | cpio -idv** to extract all files from an rpm to the current directory.
- Complete practice **RH124 CH 13.10.** and **CH 13.11.**

# RPM config file management

- In case you did not modify a package's default config file in the /etc directory, every remove or update operation is obvious: it will remove the unmodified config with the package.
- In case you installed a package that comes with config file(s), and modified the file, then the following renaming convention is used:
  - Any update that would overwrite the config will result in a filename ending with .rpmnew. Your edited config will remain in place.
  - When removing such a package, your modified config will be saved as *something.rpmsave*.
  - When installing a package that already has a relevant .rpmsave file, the .rpmsave file will be automatically moved back in place.

# Boot process

Lecture 6

Chapter 2

# Linux boot process

- The startup process of a Linux OS has three major steps: the loading and starting of the kernel binary (bootloader), the kernel initialization process (initrd FS, hardware init and rootfs mount), the service initialization part (init process, setting network, additional mounts, starting server processes like: sshd, dhcps, httpd, etc.)
- The first part is different for computers equipped with a BIOS firmware and for those with UEFI.
- The third part can differ based on your distribution. Many older, but possibly still running systems use the **init** (System V) scheme for this part, current distros ship with the **systemd** service (and system) manager.

# Bootloader

- The running kernel has access to filesystems, has a memory space and hardware set up, has a scheduler enabled and can handle new executed processes.
- The kernel itself is a binary executable that runs on the CPU. But how is it started in the first place, when there are no filesystems and process spaces available?
- This is job of the bootloader. The most common today is GRUB2 (Grand Unified Bootloader), which can load several operating systems, including Android, Windows, Linux, OSX, and many more.
- The bootloader is considerably different for BIOS equipped and UEFI firmware equipped computers. Both will be discussed in this lecture.

# BIOS bootloader

- In this case the bootloader has to do everything on its own: access filesystems, networks, display adapters, etc... There are no system calls or libraries available.
- BIOS computers have a very limit set of features ready when powered on. Instead of system calls to the OS (which is non-existent yet) GRUB resolves to calling software interrupts for e.g. for reading sectors from the hard disk or to read keypresses from the keyboard.
- BIOS computer can distinguish disks, but not partitions and files. You can select a **drive to boot from**, only the very first sector of this drive will be loaded at startup -> The first stage of grub is located in the MBR (remember lecture 5), BIOS is unaware of partitions at all.
- Grub can also load from the PBR of the active partition (with MS MBR), but this is not the recommended way.

# GRUB stages

- The first stage of GRUB will load a pre-defined set of sectors. The list is encoded in the first stage of grub and is calculated as an absolute disk sector of the next stage's file-position when installing grub. It's forbidden to move this file around in the FS. It's recommended to have a separate boot partition to hold files related to grub (grub.cfg, grub modules, kernel images and initrd files), but they can reside on the rootfs itself.
- This binary code in the list of sectors loaded by stage 1 is also referred to as stage 1.5. By the time stage 1.5 is loaded grub is aware of some filesystem formats. It can load stage 2 from a regular file in a filesystem.
- In stage 2 grub becomes as full featured as possible. It can now load additional modules from files to extend its functionality, e.g. other filesystem drivers, graphic drivers, images, etc...

# GRUB kernel load

- In stage 2 grub also loads its configuration file, **grub.cfg**, possibly presents a graphical menu to choose from operating systems. This is done according to the configuration file. Upon user's menu selection an iso image can be booted from FS or another partition can be chainloaded (e.g. Windows), or Linux can be loaded.
- In case of selecting a Linux entry, the goal is to load the kernel binary from an available filesystem. The kernel is a self extracting compressed executable (vmlinuz), which is placed at a predefined location in the memory.
- Grub will also load the initrd (initial ramdisk) file from FS and also place it to a predefined location in memory. Lastly the kernel command line arguments are copied from the config file to a predefined address in memory.
- The bootloader jumps execution to the first byte of the pre-loaded kernel image.

# Grub.cfg example

/boot/grub2/grub.cfg

```
menuentry 'Fedora (4.9.14-200.fc25.x86_64) 25  
(Workstation Edition)' --class fedora --class gnu-  
linux --class gnu --class os --unrestricted  
{  
    load_video  
    set gfxpayload=keep  
    insmod gzio  
    insmod part_msdos  
    insmod mdraid1x  
    insmod lvm  
    insmod ext2  
    set root='lvmid/re3pvr-[...cut...]'  
    linux16 /vmlinuz-4.9.14-200.fc25.x86_64  
    root=/dev/mapper/fedora_[...cut...] ro  
    rd.lvm.lv=fedora_[...cut...]/root  
    rd.md.uuid=76e0[...cut...]  
    rd.lvm.lv=fedora_[...cut...]/boot  
    rd.md.uuid=60e2[...cut...] rhgb quiet LANG=hu_HU.UTF-8  
    initrd16 /initramfs-4.9.14-200.fc25.x86_64.img  
}
```

# Kernel execution

- This far the x86 PC is still not in 32bit protected mode. Only a limited amount of memory is addressable. The kernel extracts itself by steps of relocations and mode switching to protected mode. This involves initializing the MMU and the memory subsystem.
- More hardware initialization follows, like setting up a frame buffer console, enumerating the PCI bus to look for disk controllers. If all the drivers (controller, disk, fs) required to mount the rootfs are compiled statically into the kernel, it can be mounted by now.
- This is not the case too often. The purpose of the initrd is to serve as a temporary root filesystem (it only exists in RAM). All the tools and additional drivers to mount the actual rootfs are inside it. (For e.g. network drivers, userspace tools to ask for encryption password, etc...)
- Once done, the memory occupied by initrd is freed.

# UEFI boot

- The UEFI firmware can be considered a miniature operating system in many ways. It has a scheduler, can execute binaries, it provides services via system calls and libraries, can have loadable modules (drivers), has a network stack, can handle GPT partitions and mount filesystems.
- This makes things a lot easier, a separate boot loader is not even strictly necessary. Though for maintaining existing features and concepts by grub, an EFI version of GRUB exists. (for e.g. EFI does not support initrd)
- The bootloader in this case is a simple **.efi executable file** started by the firmware from the EFI system partition (Lecture 5), like `/EFI/Boot/Fedora/grubx64.efi`. There are no stages to grub, but still a `grub.cfg` and possibly some modules are read from a filesystem.

# UEFI boot

- The whole 64 bit or 32 address space is available in UEFI (depending on architecture). When the kernel is loaded into memory, it initializes a bit differently:
  - Extraction is simpler, doesn't need multiple relocations, the whole space is accessible at once.
  - There is a point when the kernel abandons all UEFI services and asks the firmware to stop its scheduler. Then EFI occupied memory is freed.
- Role of initrd remains the same with UEFI booting.
- There is also a way for loading the kernel directly as an .efi file. In this case the initrd has to be embedded in this file. This is called **EFI boot stub**.
- For BIOS computers it is possible to put the start of the kernel directly in the MBR. This solution is seldom used and it is called the **legacy boot stub**.

# Late boot operations

- When the kernel is done with hardware setup to the point of mounting the root partition it will execute the first userspace process (PID=1) with root permissions, namely `/sbin/init`. It is also common that the init binary is located inside the `initrd`, and it is responsible for mounting the real `rootfs` itself and execute the disk instance of `/sbin/init`. Note that both `systemd` and `SysV` managers go under the filename of `/sbin/init`.
- Init is the only userspace process the kernel will start. All other processes, including services, are forked from init or its children. If `PID1` process crashes, a kernel panic is triggered, and system operation is intentionally halted.
- After initialization is done, the kernel's main execution thread becomes `PID0`. It is scheduled when no other processes are queued and does nothing else but tries to reduce CPU power consumption in this idle state.

# System manager

- The kernels role is reduced to executing system calls when the first process is created. Strictly saying the boot procedure is done, the rest of system startup continues in user space.
- The PID 1 process is responsible for a lot of early (process) startup tasks, like mounting additional FSs from the fstab, configuring network interfaces, bridges, addresses, etc...
- All the daemons (server processes) are started by PID1. The getty processes are also spawned by PID1 on the virtual ttys to present a login terminal to the user.
- There is a daemon called udevd (also started by PID1), which is responsible for enumerating the rest of hardware devices, loading non-essential modules like soundcard drivers. This process will remain resident and handle hardware changes, like USB plug events.

# System V

- The most common system manager was the System V style **init** process up till a few years ago. (RHEL6)
- The legacy early startup config files like fstab, crypttab, inittab belong to this scheme, but they are preserved by systemd for compatibility reasons.
- The server processes are started by shell scripts located in **/etc/init.d** directory. These scripts start and fork the server processes binaries with several arguments, like config file, data directory, and several switches; so that the user doesn't have to know them by heart. Some init.d scripts accept special arguments (like initdb for postgres, reload for httpd), but all of them have to handle the start, stop, restart arguments.
- Services are managed by executing commands in this form: **/etc/init.d/svcname start** (or **restart** or **stop**).

# System V

- The systems predefined states are mapped to one digit runlevels (not all of the are used). Each runlevel has a corresponding directory called `/etc/rcN.d`, where  $N$  is the number of the runlevel.
- Automatic process startup is done by init the following way: Each `rcN.d` has a set of numbered symlinks with names starting like `Sxx` and `Kxx`, where  $xx$  is a two digit number. Each symlink points to a script in `/etc/init.d`, all `K` symlinks are executed with the `stop` argument, all `S` symlinks are executed with the `start` argument, when entering a given runlevel.
- The two digit number defines the order `S` and `K` scripts are run. They are run **sequentially** with no overlap.
- The default runlevel is defined in `/etc/inittab`. You can switch runlevels with `init N`, where  $N$  is the runlevel nr.

# systemd

- The systemd system manager suite is complex set of daemons and config files. It has a builtin system logger, called `systemd-journald`, a network manager subsystem called `systemd-networkd`, several compatibility subsystems, like `systemd-fstab-generator`, it features `dbus`, `SELinux`, `cgroups` integration and many more.
- It was therefore argued by many Linux professionals. It violates the Unix concept of limiting subsystems/daemons to doing one specific task. Systemd eventually won the war against SysV, mostly because it outperforms the legacy system, and it has a very clear API and syntax.
- Systemd has very clear declarative ini style (sections, key=value pairs) unit files for services, sockets, mounts.
- Systemd organizes these units into targets. Targets are the closest equivalent to runlevels.

# systemd units

- A service unit example. (lines with - accept non zero exit)

```
/usr/lib/systemd/system/postfix.service

[Unit]
Description=Postfix Mail Transport Agent
After=syslog.target network.target
Conflicts=sendmail.service exim.service

[Service]
Type=forking
PIDFile=/var/spool/postfix/pid/master.pid
EnvironmentFile=-/etc/sysconfig/network
ExecStartPre=-/usr/libexec/postfix/aliasesdb
ExecStartPre=-/usr/libexec/postfix/chroot-update
ExecStart=/usr/sbin/postfix start
ExecReload=/usr/sbin/postfix reload
ExecStop=/usr/sbin/postfix stop

[Install]
WantedBy=multi-user.target
```

# systemd units

- A mount unit example and target unit example:

```
/etc/systemd/system/mnt-backups.mount
```

```
[Mount]
```

```
What=/dev/disk/by-uuid/86fef3b2-bdc9-47fa-bbb1-  
4e528a89d222  
Where=/mnt/backups  
Type=ext4  
Options=defaults
```

- Mount unit filename always represents mount point.  
Mount units can also have roles in dependencies.

```
/etc/systemd/system/foo.target
```

```
[Unit]
```

```
Description=Foobar boot target  
Requires=multi-user.target  
Wants=foobar.service  
Conflicts=rescue.service rescue.target  
After=multi-user.target  
AllowIsolate=yes
```

# systemd

- Targets, services, mount units have requirement definitions like Requires, Wants, Before, After.
- Evaluating these will define a complex dependency tree, that is built in run-time. Starting of tasks that are not dependent on each other is done in **parallel** at any stage.
- Switching targets is done with the **systemctl isolate *some.target***, where target can be for e.g. graphical.target or multi-user.target.
- The default target can be set by **systemctl set-default *some.target*** command.
- (Re)Starting and stopping units is done with **systemctl start/stop/restart *unit.name***, for e.g systemctl restart sshd.service.
- Setting a unit for autostart is done by **systemctl enable/disable *unit.name***.

# systemd

- Use **systemctl status** to query all services or **systemctl status *unit.name*** to query a single unit. This will show whether the unit is active or not, enabled or not. It will also print the last few lines of log entries relevant to the unit. Use the **-n** switch to specify number of log lines, use **-l** to prevent long lines from going out of the screen.
- Systemd units are stored in **/usr/lib/systemd/system**. This folder is managed by the package manager. User defined units go to **/etc/systemd/system** folder. If you want to modify a unit, place a unit file with the same name to the **/etc** folder of systemd instead, this will override the **/usr** version of the file with the same name.
- Use **systemctl list-units** or **list-unit-files** to list names or filenames. Use **--type=service** to filter for service units, use **systemctl --failed** to filter to failed units.
- Complete practice **RH124 CH 8.2.** and **8.4.**

# systemd cgroups

- There is a kernel feature called control groups for the purpose of grouping processes and accounting/isolating the group's resource usage: for e.g CPU share, memory usage and/or swap limit.
- Cgroups are represented as systemd slice units. They are organized in a multi-tier hierarchical tree structure.
- Use **systemd-cgls** to print the control group tree.
- User slices belong to process trees started by users, system slices contain services started by systemd, and machine slices contain resources allocated virtual computers.
- Issue **systemd-cgtop** for a top-like utility that sorts systemd slices based on resource usage.

# Troubleshoot boot issues

Lecture 6

Chapter 3

# GRUB issues

- The grub bootloader can be reinstalled if necessary. Use an install media to boot to a Linux instance. Mount the rootfs to `/mnt/sysimage`, if you have a separate boot partition mount it to `/mnt/sysimage/boot` directory. Now **grub2-install --root-directory=/mnt/sysimage /dev/sda** This command will install grub to the MBR of `/dev/sda` and place modules and required files under `/mnt/sysimage/boot`.
- To create a new grub.cfg file use **grub2-mkconfig -o /boot/grub/grub.cfg**. This will autodetect all operating systems (non-Linux's too) and all kernel versions available in the system and make menu entries for all of them. Do not edit grub.cfg manually, since the above command is run by the package manager on kernel updates, and will be overwritten. Modify `/etc/default/grub` file instead for adding command line argument to the kernel.
- Complete practice **RH134 CH 13.8**.

# initrd issues

- You can recreate the initrd image for the current running kernel with **dracut -f**.
- To generate initrd for another kernel version specify the whole filename and version: **dracut -f /boot/initramfs-2.6.32-358.el6.x86\_64.img 2.6.32-358.el6.x86\_64**
- You can enable/add more features and modules, like sshd to your initrd image. Edit files under `/etc/dracut.conf.d/` directory to change settings.

# Kernel cmdline

- You can use the **e** key in grub to edit a menu entry at runtime. After editing press **ctrl+x** to execute the entry.
- This modification has a one time effect, grub will never write it to disk. Edit grub.cfg or rather **/etc/default/grub** instead for permanent changes.
- This way you can modify the kernel cmdline to influence the boot process. The kernel, just like any other binary, accepts several arguments.
- Many kernel modules and userspace processes (including systemd) parse this line too. It is available under the **/proc/cmdline** virtual text file for every process.
- Look up <https://www.kernel.org/doc/html/v4.10/admin-guide/kernel-parameters.html> for more.

# Kernel cmdline

- You can override the path of the executable started as PID=1 by adding the `init=/some/binary/file` argument.
- You can specify a systemd target on the command line. This will override the default target, that is set by `systemctl set-default` command. Use **`systemd.unit=rescue.target`** argument to boot into a very basic system. Specify **`systemd.unit=emergency.target`** to stop the boot process immediately after the mount of the rootfs. At this state the root filesystem will only be mounted in ro mode. Switching to rw is done at a later target.
- Add the **`rd.debug`** and **`rd.shell`** to drop to a shell in case of initrd errors. Add **`rd.break=[cmdline|pre-udev|pre-trigger|initqueue|pre-mount|mount|pre-pivot|cleanup]`** to drop to a shell in specific points of the initrd part of the boot process.
- Complete practice **RH134 CH 13.4.** (reset root password)

# Systemd issues

- Issue **systemctl enable debug-shell.service**. This will enable a root shell early on the **ctrl+alt+F9** virtual terminal, long before normal vtys would be available. This is a serious security hole, only use temporarily for debugging.
- Issue **systemctl list-jobs** in the root debug shell to see stuck systemd jobs. Issue **systemctl --failed** to list failed systemd units, then look up their status with **systemctl status -l -n 80 *unitname***.
- A common reason for systemd to get stuck is a broken fstab, that prevents the system from mounting FSs. Since almost all units depend on **local-fs.target**, this will hold up the whole startup → Use the **systemd.unit=emergency.target** kernel cmdline to drop to a **sulogin** shell and repair the fstab file or do an fsck on the filesystems.
- Complete practice **RH134 CH 13.6**.

# Recommended reading

Lecture 6

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH 124 CH13 (RPM, yum, package management)
- RH 134 CH13 (Boot issues)
- RH 124 CH8 (Controlling services)

- RH 124 CH11 (Manage networking)
- RH 134 CH14 (Firewalld)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 6, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# Network settings

Lecture 7

Chapter 1

# Interfaces and networks

- There are several ways to automate networking setup at system startup.
- Legacy Sys V init systems have a network script (`/etc/init.d/network`) that sets up network interfaces at boot, based on files in `/etc/sysconfig/network-scripts` in Red Hat family, and files in `/etc/network/interfaces` in Debian family of Linux.
- Both families have the Network Manager daemon available that works according to configuration files in `/etc/NetworkManager` directory.
- The systemd system manager has a builtin network manager called `systemd-networkd`, which works based on config files in the `/etc/systemd/network` directory.
- Only one of the above network manager daemons/scripts should be enabled at a time in a system.

# General network config files

- The `/etc/hosts` file serves as a basic name resolution service, it has name  $\leftrightarrow$  IP address pairs listed in text format. This overrides any other source of name resolution.
- The `/etc/hostname` file contains the hostname of the system. Use **hostnamectl status** and **hostnamectl set-hostname station.domain.tld** form to manage it.
- The `/etc/resolv.conf` contains the DNS resolver settings:

`/etc/resolv.conf`

```
search example.com company.net
domain test.org
#either use search or domain, not both
nameserver 192.168.0.100
nameserver 8.8.8.8
```

- The `/etc/protocols`, `/etc/services` files contain numeric  $\leftrightarrow$  alphabetical pairs of protocol and service identifiers.

# Network config files

- The `/etc/sysconfig/network-scripts` directory contains the ifcfg, ifup and ifdown scripts. On a legacy RH system, they are the direct source of network settings.
- If the NM daemon is set up and enabled, it will create ifcfg files programatically.
- The NM daemon as well as systemd-networkd will manage it's own version of resolv.conf dynamically (based on DHCP replies and static connection settings). In this case the `/etc/resolv.conf` file is a symlink to the daemon-created instance (usually found somewhere in the `/run` directory).
- The `systemd-resolved` service has to be enabled separately from `systemd-networkd` for the above functionality to work. `NetworkManager` doesn't have this separated.

# Interface naming

- A computer connects to networks via hardware devices called NICs (Network Interface Cards). A computer can have many interfaces, IFs can have multiple addresses.
- In legacy systems the interfaces are named by their type (physical layer mostly) and numbered in order of device node creation (udevd). E.g. `eth0`, `eth1`, `wlan0`, etc...
- In modern Linux systems the naming convention is the following:
  - The first two letters are assigned based on IF type, like for e.g. `en` or `wl`
  - The next letter is assigned based on IF connection, e.g. `p` for PCI, `o` for on-board, `s` for hot-plug interfaces
  - The last character is numeric, it represent port or index number.

# Systemd networking

- You can choose an arbitrary name for the configuration files, but they should end with .network. Put them in /etc/systemd/network directory. Look up <https://www.freedesktop.org/software/systemd/man/systemd.network.html> for more details.

```
/etc/systemd/network/25-wireless.network
```

```
[Match]
Name=wlp2s0
[Network]
DHCP=ipv4
[DHCP]
RouteMetric=20
```

```
/etc/systemd/network/50-wired.network
```

```
[Match]
Name=enp1s0
[Network]
Address=10.1.10.9/24
Gateway=10.1.10.1
```

# Querying networks and interfaces

- The legacy **ifconfig -a** command can be used to query all interfaces, you can also use **ifconfig *if-name* *addr*** command to set an address for a given interface. The ifconfig utility has limitations, it is considered obsolete.
- Use the **ip** utility to manage ip settings. Type **ip address show enp1** to see the address of the enp1 interface. Omitting IF name will result in printing all interface addresses.
- Issue **ip route show** to see the routing information.
- You can query IF statistics with **ip -s link show *if-name***.
- The **ss** command may be used to query listening and established connections of the processes running in the system. Note -t for TCP only, -u for UDP only, -l for listening sockets only, -p for printing assigned PID, -n for skipping host name and protocol name resolution.

# Trouble- shooting networking

- Use **ping** to check connectivity. You can either specify an address or a hostname after ping. If the hostname is not resolved by ping, then your resolv.conf might not be in order. The -c switch can set a counter for ping.
- Use **host**, **nslookup** or **dig** for checking name resolution.

## Shell command line

```
# nslookup -type=any google.com ns1.company.net
Server: 192.168.19.2
Address: 192.168.19.2#53
Non-authoritative answer:
Name: google.com
Address: 173.194.35.7
Name: google.com
Address: 173.194.35.8

google.com  nameserver = ns1.google.com. [...]
```

- Use **traceroute** or **tracepath** for checking routing path of a packet. (relies on ICMP, might be forbidden)

# Network Manager concepts

- The NetworkManager daemon (NM) if enabled, is a persistent process, that automatically configures interfaces at boot, and handles all the interface events, like hot-plugging and interface or the loss/gain of a link/carrier on interfaces.
- NM can be configured using a GUI, a TUI, and a CLI interface. It also has a Gnome notifier application for the graphical desktop. The command line interface will be discussed in detail in this lecture.
- NM can handle several objects, like interfaces, connections, radios, and general network settings.
- The most important objects for us are the device objects, which represent Linux network interfaces, and connection objects, which represent sets of settings applicable to interfaces.

# Network Manager concepts

- A device can have many related connections but only one connection can actively use a device at a time.
- A device can be used in multiple networks, like for e.g. connecting the same computer's interface to a lab network for once that requires static config, then to a home network for once which provides DHCP services.
- Each connection can be saved by NM, and instead of changing each individual network setting manually, you can instruct NM to switch connections instead.
- Look at the output of **nmcli con show** and **nmcli dev status** !

# nmcli

- NM's CLI interface is called **nmcli**. The general form of the command is as follows:  
**nmcli [OPTIONS] OBJECT {COMMAND | help}**  
where object (in our case) will either be device or connection. After that, you should specify a command like add, modify, etc... You can use the help command to get help relevant to the selected object.
- The words in the nmcli command line can be abbreviated to a non-ambiguous level. For e.g **nmcli connection show**, **nmcli con sh**, **nmcli c s** all instruct for the same activity.
- You can also use TAB completion when dealing with nmcli commands.

# nmcli con

- You can list all connections with **nmcli con show** or list only the active ones with **nmcli con show --active**.
- You can request all the details of a connection by naming it in the command: **nmcli con show "static-eth0"**

Shell command line

# nmcli connection show			
NAME	UUID	TYPE	DEVICE
docker0	33181c5b-[...]	bridge	docker0
dravanet	5dcea3ee-[...]	pppoe	enp3s0
enp2s0	41bba397-[...]	802-3-ethernet	enp2s0

- You can list devices and their active connection with **nmcli dev status**. Get details by naming the device to the show subcommand, e.g. **nmcli dev show enp2s0**.
- Refer to **RH124 CH 11.5.** for more.

# nmcli con

- An example for adding connections: **nmcli con add con-name "default" type ethernet ifname eth1** This will set the connection to be a DHCP client since no address options have been specified. Another example: **nmcli con add con-name "demo" type ethernet ifname eth1 autoconnect no ip4 172.25.0.10/24 gw 172.25.0.254** This will create a connection named demo for the eth1 interface with a staticIPv4 address.
- Issue **nmcli con add help** for all available property strings.
- You can switch between connections by issuing **nmcli con up "demo"** or **nmcli con up "default"**.
- You can administratively disable an interface with **nmcli disconnect dev eth1**. This will also terminate its current active connection.

# nmcli con

- You can bring a connection down by **nmcli con down "demo"**. This will not disable the interface, meaning that in the event of a link change, the autoconnect connection for the device will be activated.
- You can modify an existing connection with **nmcli con mod *con-name* *property value*** like commands. E.g.: **nmcli con mod "static" connection.autoconnect yes**
- Some properties can have multiple values, like for e.g. `ipv4.dns`, since it is valid to have multiple resolvers configured in a network. You can assign and overwrite previous values the same way, e.g. **nmcli con mod "static" ipv4.dns 8.8.8.8** But in case of multi-value properties you can also prefix the setting with `+` or `-` to add/remove an entry, without modifying the other values of the same property. E.g. .... **+ipv4.dns 8.8.8.8**
- To delete a connection use **nmcli con del "demo"**.

# Manual configuration of NM

- Complete practice **RH124 CH 11.6.** (nmcli)
- NM stores the connection properties in the `/etc/sysconfig/network-scripts/ifcfg*` files. This is directly compatible with RH network configuration schemes.
- You can also modify these files directly with an editor like `vi`. In this case you should notify NM of the changes. To achieve this, issue **nmcli con reload**. To apply the newly set values, issue **nmcli con down "con-name"** and **nmcli con up "con-name"** afterwards.
- Complete practice **RH124 CH 11.8.** (manual config)
- All settings done to NM or the `ifcfg*` files will be permanent, meaning they will be preserved across reboots. To temporarily set network settings (until reboot) or to work without NM → look at the next slide.

# Manual configuration

- To manually set the ip address of an interface, issue a command like this: **ip addr add 192.168.50.5 dev eth1**  
To remove: **ip addr del 192.168.50.5 dev eth1**
- To enable/disable an interface, use **ip link set eth1 up/down**
- To add a routing table entry, issue **ip route add 10.10.20.0/24 via 192.168.50.100 dev eth0**
- To remove it, use: **ip route del 10.10.20.0/24 via 192.168.50.100 dev eth0**
- To set the default gateway: **ip route add default via 192.168.1.1**
- To manually edit the DNS resolver, modify /etc/resolv.conf with a text editor.
- Complete practice **RH124 CH 11.10.** (manual config)

# Firewall

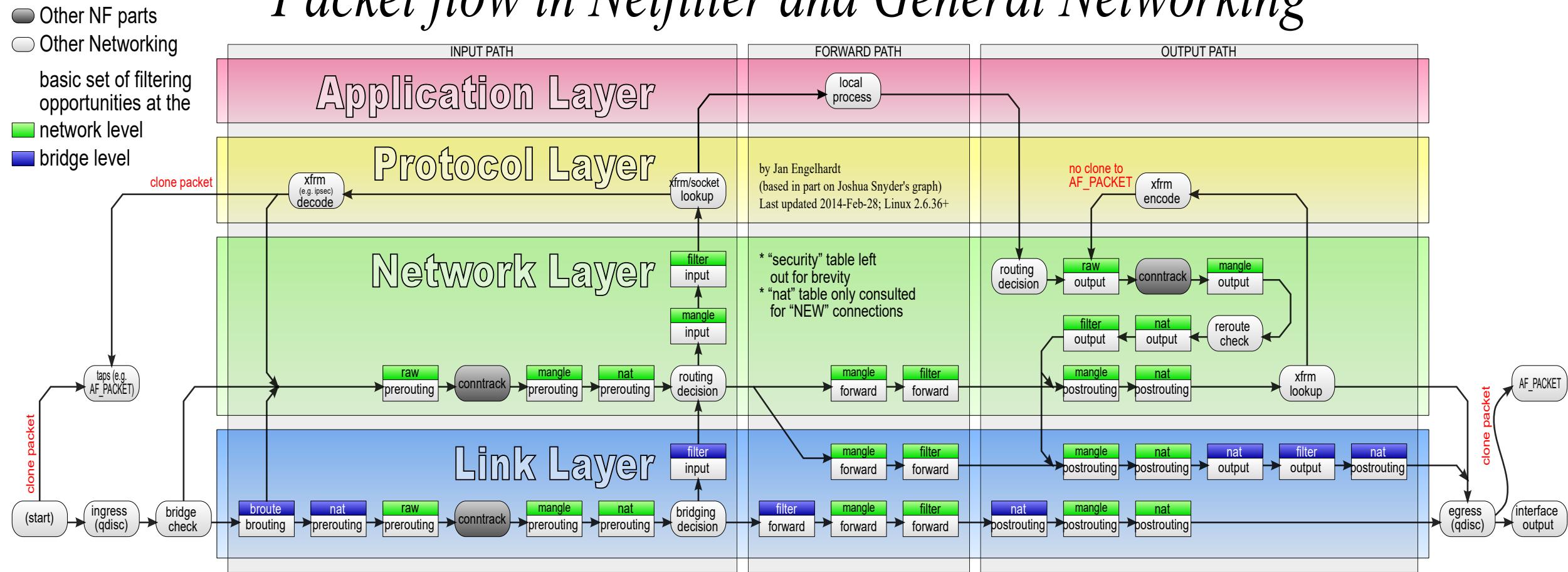
Lecture 7

Chapter 2

# Firewall functions

- In Linux the firewall functions are implemented in kernel space. This subsystem is called netfilter. The standard userspace tool for entering and manipulating the rule set is called iptables, ip6tables and ebttables.
- Firewall rules are applied to all network traffic that passes the kernel. All processes are subject to the ruleset, which is enforced by the kernel itself.
- In the RH family of Linux a user-friendly front-end called firewalld is responsible for loading the rule-set at boot time, and managing it later on. Firewalld resolves to calling several iptables commands behind the scenes.
- The older way of setting netfilter rules automatically was the iptables damon/script-set. You can either enable iptables or firewalld, but not both at the same time.
- Just like NM firewalld also has a GUI, TUI and CLI interface.

# *Packet flow in Netfilter and General Networking*



<https://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Netfilter-packet-flow.svg/800px-Netfilter-packet-flow.svg.png>

# Netfilter

- Netfilter is not only capable of filtering, but it can also manipulate packets and frames.
- Netfilter is a very complex and sophisticated system. Many network device manufacturers put an embedded Linux system in their devices, because it has a well built, proven, feature rich network stack.
- For e.g. the Linux kernel can implement a NAT/PAT router, which is achieved by setting the connection tracking and packet mangling rules accordingly.
- All the frames and packets come from network devices, which also work in kernel space. The whole netfilter job is done inside the kernel, therefore fast and reliable. Only the rule manipulation tools run in user space.
- Only a small part of the netfilter framework's features will be relevant to us.

# iptables tables

- The rule-set of iptables consist of 5 tables: raw, filter, mangle, nat, security.
- Only two of those will be relevant for us: nat, where the PAT/NAT feature is implemented and filter, where the filtering rules apply.
- Mangle resembles nat, it can do specialized packet alterations.
- Security is only used by MAC (SELinux) mechanisms.
- Raw is a way of exempting sorts of traffic from connection tracking.

# iptables chains

- Each table is built up of three major chains: the input, output, and forward chains. The rules in the chains are matched against traffic ACL style. The nat table also has a prerouting and a postrouting chain. Any number of chains can be added to tables manually.
- The INPUT chain controls the packets that are intended to be delivered to one of the addresses of the current system. (Not all received incoming packets are input)
- The FORWARD chain controls packets that are received as incoming data on one of the interfaces but are not meant for this specific system. If routing (IP forwarding) is enabled and configured correctly, such traffic will be forwarded via a relevant interface to another router/destination.
- The OUTPUT chain controls the packets that are assembled/originated by a process of the local system.

# iptables rules

- Each chain contains a set of rules, that are evaluated in a specif order. A rule also specifies a target. Once a packet matches a rule, no further evaluation is done, it's target (fate) is decided.
- A target can be another chain, but most commonly it is a built-in target, like ACCEPT, DROP, REJECT, LOG, RETURN.
- Each chain has a default policy, which simply determines a target. If a packet does not match any rules, it will be destined to the default target.
- An example rule would be like the following:  
**-A INPUT -p tcp --dport ssh -s 10.10.10.10 -j DROP**  
This will destine all packets that come from 10.10.10.10 and have a destination port of 22 to the DROP fate.

# iptables

- To add rules, simply issue **iptables rule**, where rule is like a rule line shown above.
- Using the -D switch instead of -A will erase a rule.
- To clear all tables use **iptables -F**.
- To print all rules of a table: **ipatbles -L -t tablename**, if omitting the -t switch, the FILTER table will be printed.
- The old fashioned iptables daemon simply saves all the active rules on shutdown and loads them at startup. This is done by calling **/sbin/iptables-save** and **/sbin/iptables-restore**. The I/O of these command should be redirected to a file, which holds the rules.
- In order for IP forwarding to work, besides the permissive rules in iptables you also have to enable the global packet forwarding switch in the kernel: **echo 1 > /proc/sys/net/ipv4/ip\_forward**

# iptables PAT router example

## Shell command line

```
# iptables -F; iptables -t nat -F; iptables -t
mangle -F [this will clear the 3 relevant tables]
# iptables -t nat -A POSTROUTING -o ppp0 -j
MASQUERADE [this will turn on packet header
modification according to PAT. All TCP sessions
passing through will be tracked by the kernel.]
# iptables -A INPUT -m state --state ESTABLISHED,
RELATED -j ACCEPT
# iptables -A INPUT -m state --state NEW -i ! ppp0
-j ACCEPT
[only accept incoming packets that are related to an
established TCP session]
# iptables -P INPUT DROP
# iptables -A FORWARD -i ppp0 -o ppp0 -j REJECT
[new sessions initiated from outside will be
dropped, forwarding of foreign packages will be
rejected.]
# iptables -t nat -A PREROUTING -i ppp0 -p tcp
--dport 80 -j DNAT --to 172.31.0.23:80
[a port forward example]
```

# Firewalld zones

- Firewalld divides traffic into zones. Zones are implemented as chains on the iptables level.
- Each zone has a separate list of rules (allowed services). An incoming packet will be classified to a zone, then matched against the zones rules and its fate will be determined.
- An incoming packet is checked for its source address, to determine the applicable zone. If no zone rule matches to the source address, then the zone selection decision will be based upon the interface that received the packet. All interfaces belong to a zone. There is a default zone for new interfaces.
- Zones rank from trusted, work, external, ..., to dmz and drop, in order of strictness. You can even define your own zones.

# firewalld services

- A service is defined by a set of ports, for e.g. the ssh service is equivalent to the single port of 22/tcp. The samba-client service consists of allowing two ports, 137/udp and 138/udp.
- A zone can have any number of enabled services. All ports, that do not belong to an enabled service of the zone, will not be available for communication.
- Use the **firewall-config** GUI utility for managing firewalld.
- Use **firewall-cmd** for command line management.
- You can print all the services with **firewall-cmd --get-services**.
- There is a way for enabling individual ports or ranges without assigning a service name to them.

# firewall-cmd

- Firewalld separates its runtime configuration from the saved configuration that is automatically applied at startup. When modifying a setting it will only be applied to the runtime rule-set, but won't be saved to disk. Adding the **--permanent** switch to firewall-cmd will change the saved configuration, but not apply the rule at runtime.
- In many cases you have to issue a firewall-cmd command both with and without the **--permanent** switch.
- There is the **--timeout=N** switch for applying a change to firewall-cmd. A rule entered with the timeout switch will be automatically reverted after timeout passes. N is interpreted as number of seconds.
- Issuing **firewall-cmd --reload** will drop runtime configuration, reload and apply all settings of the saved state.

# firewall-cmd

- Issue **firewall-cmd --get-zones** for listing the zones, **--get-active-zones** for printing only the zones that have at least one interface in them.
- Use the **--get-default-zone** switch to see the default zone for new interfaces.
- Use **--list-all-zones** to print detailed information about zones (for. e.g. interfaces, enabled services, etc.)
- Use **firewall-cmd --get-services** to list the available services.
- Issue **firewall-cmd --list-all** for printing the whole configuration.
- Use **firewall-cmd --set-default-zone=zone** to set the default zone to *zone*.

# firewall-cmd

- Take the **--add-source=10.1.0.0/16 --zone=trusted** example to assign a network to a zone. Use **--remove-source=10.1.0.0/16 --zone=trusted** to revert such a change.
- Use the following example to add an interface to a zone:  
**firewall-cmd --add-interface=eth0 --zone=trusted**  
Use **--change-interface=eth0 --zone=dmz**, to reassign the interface to a different zone.
- Use **--add-service=mysql --zone=dmz** to enable a service such as mysql to be available in the dmz zone.
- Use **--add-port=17784/tcp --zone=dmz** to allow non-standard service ports to a zone, without creating a service for it.
- Use **--remove-port** and **--remove-service** logically.
- Complete practice **RH134 CH 14.2.** and **14.3.**

# firewall-cmd

- To enable gateway functionality with firewalld:
  - Add your WAN facing interface to a zone, for e.g. **external: firewall-cmd --add-interface=pppo --zone=external --permanent**
  - Enable masquerading on the zone: **firewall-cmd --zone=external --add-masquerade --permanent**
  - Apply the configuration: **firewall-cmd --relaod**
- Take the following example: **firewall-cmd --add-forward-port=port=2822:proto=tcp:toport=22:toaddr=10.0.0.28 --zone=external** This exposes the ssh service of a masqueraded host: It will channel the data bound to the 2822 port of the gateway's external interface to the 22 port of the 10.0.0.28 machine behind the gateway. Instead of a single port you can also specify a port-range of the same size in *port* and *toport* fields.
- Use **--remove-forward-port=[same long line]** to revert.

# TCP Wrapper

Lecture 7

Chapter 3

# TCP wrapper

- TCP wrapper is an implementation of ACLs for network services.
- It filters network access of processes, based on the peer's network address or hostname.
- TCP wrapper is a library, it works somewhat similar to libPAM. Binaries compiled against it can ask the "opinion" of the library, which in turn replies with a binary answer regarding whether the given communication can go on or not.
- As opposed to netfilter, which is enforced to all processes by the kernel, in case of the TCP wrapper library processes volunteer themselves for an ACL check.
- TCP wrapper is considerably easier to configure, than iptables, but almost as powerful in case you only want to limit access to services.

# libwrap.so

- Most deamons from the repository are compiled to be libwrap-aware, libwrap.so is the name of the binary shared object file of TCP wrapper.
- You can check whether a process relies on libwrap.so or not by running ldd on it's binary executable file. This command will list all the dynamically linked libraries the given binary uses.

## Shell command line

```
# which sshd
/usr/sbin/sshd
# ldd /usr/sbin/sshd
    linux-gate.so.1 =>  (0xb776c000)
  => libwrap.so.0 => /lib/libwrap.so.0 (0xb76a1000)
    libpam.so.0 => /lib/libpam.so.0 (0xb766b000)
    libssl3.so => /lib/libssl3.so (0xb6f92000)
    [...cut...]
    libz.so.1 => /lib/libz.so.1 (0xb73dc000)
```

# TCP wrapper hosts files

- There are only two files considered by libwrap: the /etc/host.allow and /etc/hosts.deny text files.
- The .allow file is evaluated first, in case a match is not found, the .deny file is also parsed line by line. Once a match is found no further processing is done, the answer is decided (ACL style). In case no matches are found in either file, the access will be allowed.
- Even if the connection is allowed, it can be logged by libwrap.
- Each line in these files contains exactly one rule. The line format is the following:  
**<daemon list> : <client list> [: <option> : <option> : ...]**
- The last line should be terminated with a newline character.

# hosts files format

- The daemon list is a comma separated list of process names (not service names) that the rule applies to.
- The list may also contain the ALL keyword and the EXCEPT operator. For e.g. ALL EXCEPT vsftpd
- The client list is a comma separated list of IP addresses, IP ranges or hostnames. Take the following examples:
  - .example.com -All hosts inside (ending with) the example.com domain name.
  - 192.168. -All IP addresses in the range of 192.168.0.0 – 192.168.255.255
  - 192.168.0.16/255.255.255.240 -An IP subnet
  - The ALL and EXCEPT keywords can also be used:  
.example.com EXCEPT attacker.example.com

# hosts files format

- There are a few more wildcards to the client list, other than ALL.
- The LOCAL wildcard refers to hostnames without a period, like server7, localhost, etc...
- The PARANOID wildcard causes libwrap to do reverse DNS lookup based on the peers IP address. Then the resulting hostname is looked up as forward DNS query. If the result doesn't match, the access is denied. (spoofing)
- The KNOWN and UNKNOWN wildcard also relies on DNS, it is checked whether the host name and address is known or the user is known via identd.
- A number of options are possible, e.g. executing external commands on a match with the **spawn** option. The deny or allow option can also be used, this overrides the decision, that is normally determined by the filename.

# TCP wrapper hosts examples

## /etc/hosts.allow

```
sshd      : .example.com : spawn /bin/echo "$( /bin /date ) access denied" >> /var/log/sshd.log : deny
mysql    : .company.net EXCEPT boss.company.net
in.ftp    : 172.25.10.12,172.25.10.13
in.telnet : /etc/telnet.hosts
ALL      : 192.168.
```

## /etc/hosts.deny

```
sshd      : 10.10.9.9
ALL EXCEPT vsftpd : 10.16.32.0/255.255.224.0
mysql    : [3ffe:505:2:1::]/64
ALL      : PARANOID
```

# Recommended reading

Lecture 7

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH124 CH11 (Manage networking)
- RH134 CH14 (Firewalld)

- RH124 CH10 (Logs)
- RH134 CH4 (Scheduling tasks)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 7, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# Mid-term overview

Linux basics

Lecture 8

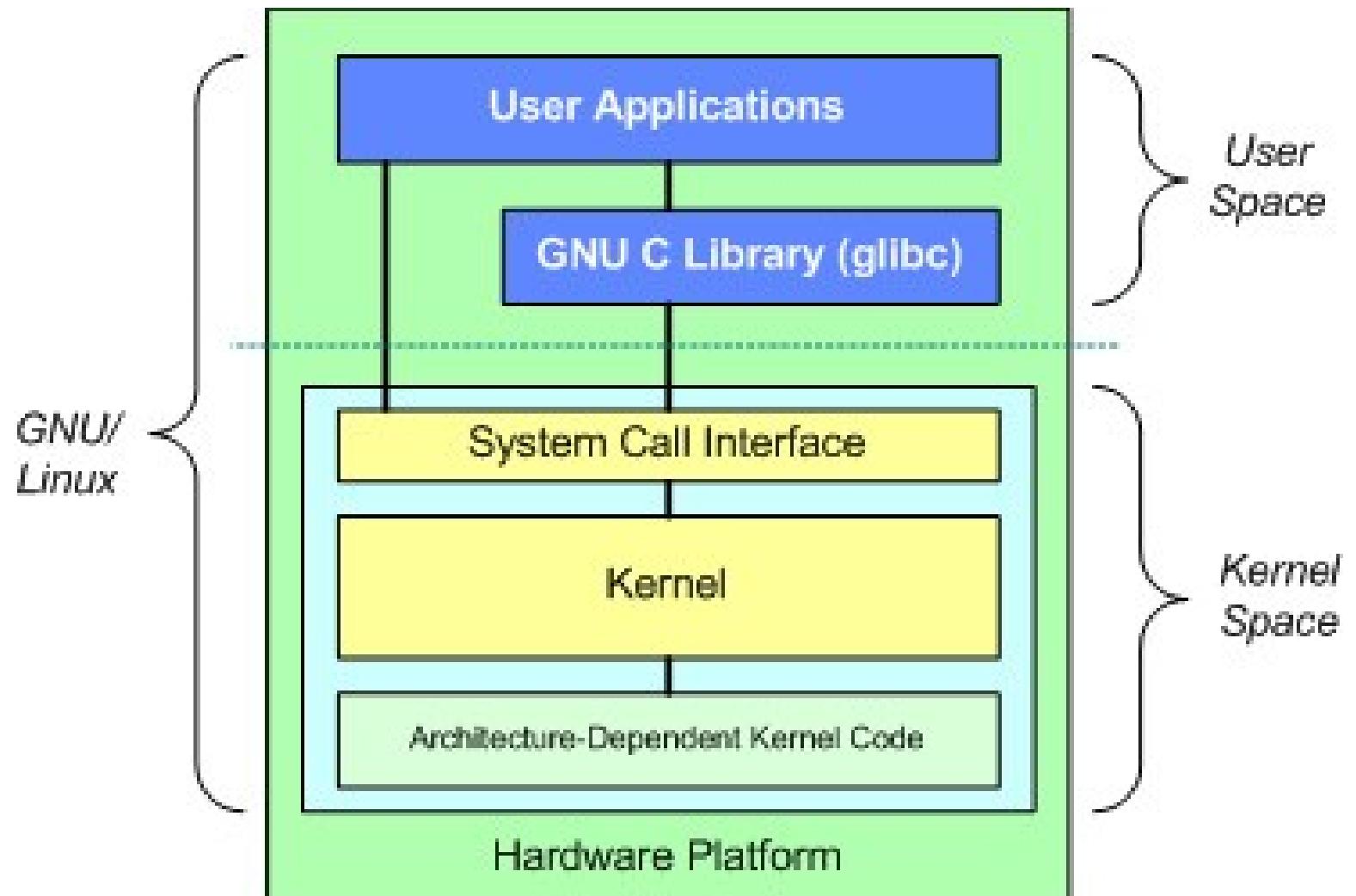
Chapter 1

# Linux basic concepts

## kernel space

- The kernel itself runs in a **privileged** hardware ring (ring0) on the Intel x86 CPUs. This privilege level will allow access for everything in the computer (all addresses and buses).
- User space processes are separated to their own virtual memory space, they are supervised by the kernel and managed by the kernel's **scheduler**.
- When a process requires a privileged operation (accessing an I/O address, reading data from the disk, or create other processes for e.g.), it will ask the kernel to do it. This is called a **system call**. The standardized interface to system calls is provided through the **std. C library** (libc).
- The **device drivers** provide an interface to processes under the /sys virtual filesystem and the device node files under the /dev directory. Device drivers run within the kernel's privileged hardware ring, they are implemented as kernel objects (.ko).

# Fundamental architecture of Linux



<https://www.ibm.com/developerworks/library/l-linux-kernel/figure2.jpg>

# Linux basic concepts

## terminals

- All interactive user sessions work through a **terminal**: a special character device, which moves in-/output bytes to/from the user. The nature of the terminal is not relevant to process running inside it. For e.g. a terminal can be a direct hardware device file provided by the kernel, or can be a pseudo terminal, which is connected to a software layer. (E.g. a graphical terminal window, or a terminal device that is remotely connected to an other computer via a software TCP/IP tunnel.)
- Terminals usually have a **shell** process (command interpreter and run-time environment) running inside them. A shell runs further processes and handles jobs.
- All processes have **3 standard streams**, which are by default connected to the terminal that holds the parent process.

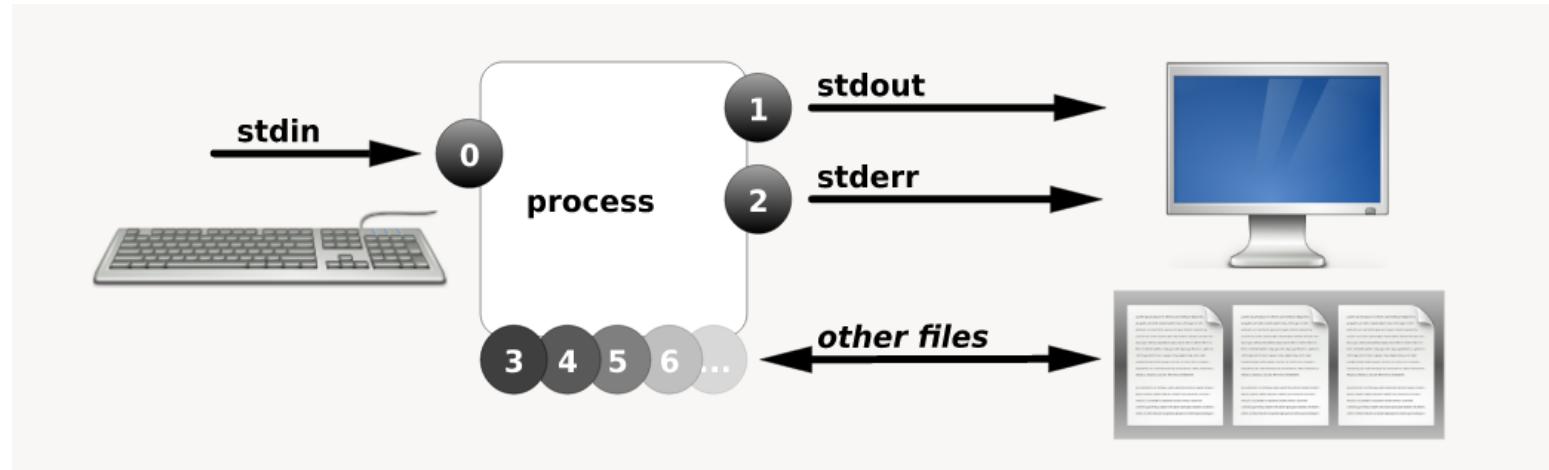
# Linux basic concepts

## standard streams

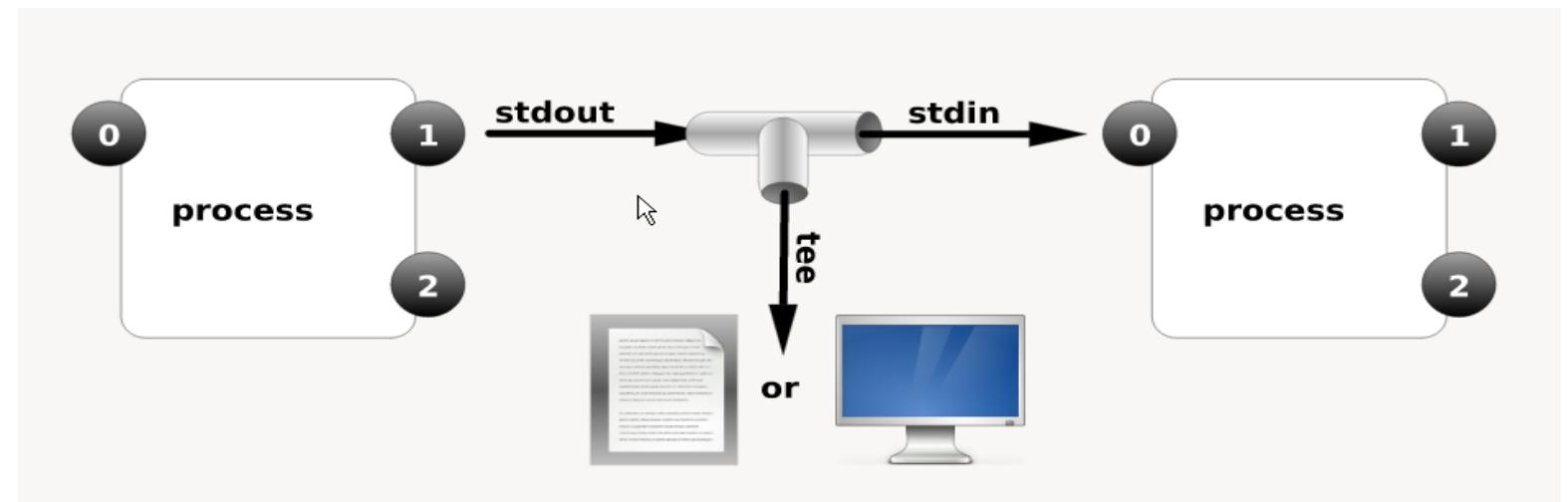
### help

- The standard stream specifier (file descriptor) is usually inherited from the parent process, but can be overridden with redirects and pipes. Standard streams are represented as files to the processes. (fds 0,1 and 2)
- Daemons are a special type of userspace processes. They have their standard streams redirected in a way, that they don't communicate with the user directly through a terminal, they read config files and write logs instead. The usual server processes, which provide services to other processes or other computers on the network are implemented as **daemons** (e.g. httpd, dhcpcd, mysqld, ftpd, ...).
- Use the **man** command to look up the manual pages for a command or a config file. Use **man -k keyword** to search the man pages for a keyword. Also consider **info** and **apropos**.

# Redirecting standard streams



- You can create a named pipes with `mkfifo`.



# Redirecting standard streams

- Examples on how to use redirects. Refer to **RH124 CH4.1.** for more.
  - `some_program >some_file` – redirect the std out stream
  - `some_program 2>some_file` – redirect the std error stream
  - `some_program 1>some_file` – redirect the std out stream
  - `some_program >file 2>&1` – redirect stderr to std out and std out (both streams) to file. Order is important!
  - `>>file` means append to file instead of overwriting from start
  - `date -s < ~/date.sav` – redirect standard input (sets the date to the string specified in date.sav file instead of requesting to type it from keyboard.)
  - `cat <<EOF > ~/some_script.sh` – read multi-line text from input. Input can be terminated by typing EOF. Then save the output in some\_script.sh. This is called a 'here document'. (shell as text editor)

# Pipes

- Processes can be chained using pipes infinitely.
- A pipe can be forked with the **tee** command, which works like a T junction. It forward its stdin to its stdout but also forwards a copy of the stream to the filename given as argument.
- The Linux concept of having a simple, specific, efficient utility (building block) for every task, combined with pipes is very powerful. Some examples:
  - `ls -t | head -n 10 | tee ~/ten-last-changed-files | mail student@desktop1.example.com`
  - `cut -d: -f7 /etc/passwd | uniq | sort`
  - `dd if=/dev/sda6 | pbzip2 -c -9 | nc hostB -p 2223`  
`netcat -p 2223 -l | pbzip2 -d | dd of=/dev/sda6`

# Linux basics

## filesystem objects

- In linux everything is a file. A full list of file types with notions (ls command style), and the related command to create the object follows:
  - - regular file touch
  - d directory mkdir
  - c character device node mknod
  - b block device node mknod
  - p named pipe mkfifo
  - s socket socket() syscall
  - l symbolic link ln -s
  - - hard link ln
  - - virtual files (/proc, /sys) (e.g. mount -t proc)

# Linux basics

## file manipulation

- Use **mkdir** to create an empty directory, use **touch** to create an empty file.
- Use **mv** for moving or renaming a file, use **cp** for copying.
- Use **pwd** to display current working directory, use **cd** to change current working directory.
- Use **ls** to list elements in a specific directory or in the current working directory.
- Use **rm** to remove a filesystem object.
- Use the **file** command to get information about a file's format and/or header.
- Use **ln** to create a hard link or **ln -s** to create a symbolic link to an existing filesystem object.

# Linux basics

## file manipulation

- Complete lab practice in **RH124 CH 2.6**.
- Now do the same the smart way!

### Shell command line

```
# mkdir Music Videos Pictures
# touch {Music/song{1..6}.mp3,Pictures/snap{1..6}.jpg,Videos/film{1..6}.avi}
# mkdir friends family work
# cp **/*[12].{mp3,jpg,avi} friends
# cp **/*[34].{mp3,jpg,avi} family
# cp **/*[56].{mp3,jpg,avi} work
```

- More on filename globbing in **RH124 CH 2.7** through **CH 2.9**.

# Linux basics

## file manipulation

- More on find in **RH124 CH4.1**. Also consider the following examples:
- To find all socket type files under /var directory with it's name containing sql and not owned by adam, type the following.

Shell command line

```
# find /var -type s -name "*sql*" ! -user adam
```

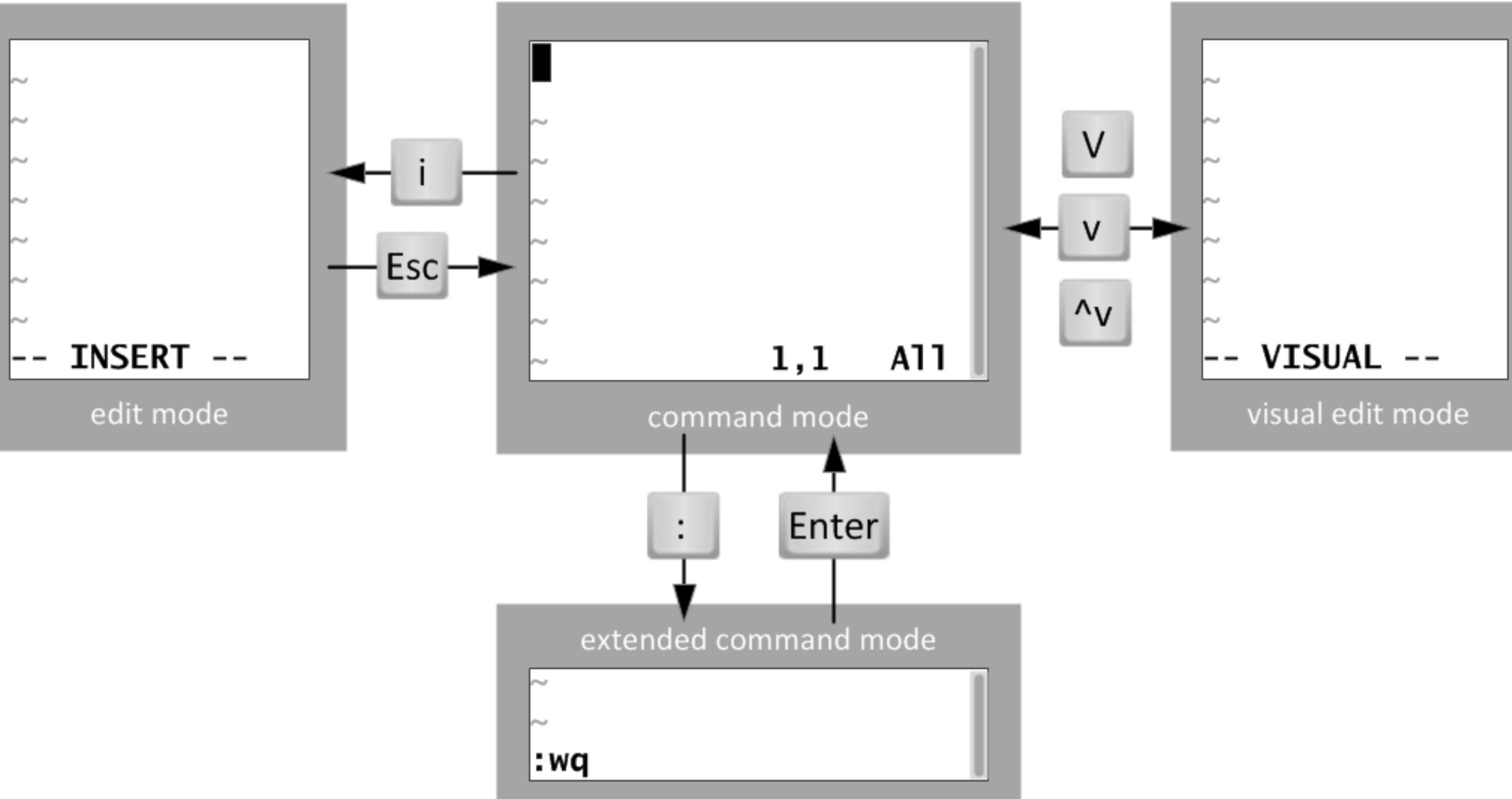
- To find and delete all large and old files owned by bob in external drive (with size larger than 100MB and accessed more then a month ago) type the following.

Shell command line

```
# find /mnt/external -type f -user bob -size +100M -atime +31 -exec rm {} ;
```

# Most important vi commands

- In command mode:
  - press i for insert mode.
  - press a for append mode
  - type dd to delete (cut) whole line
  - type yy to copy whole line
  - type p for paste after cursor, P for pasting before cursor
- Advanced commands:
  - :w to save file
  - :q to quit
  - :q! to quit without saving file
  - :wq to save and quit



# Linux basics

## file manipulation

- Use **find** to locate files in the filesystem directory structure (tree).
- Use **cat**, **less** or **more** to display the contents of files.
- Use **vi** for editing text files.
- Consider Midnight Commander (**mc**) it is a character-graphics dual panel file manager utility.
- Complete lab practice **RH124 CH 4.7.** (visual modes)
- Complete lab practice **RH134 CH 3.4.** (vi basics)
- Complete lab practice **RH134 CH 3.6.** (vi editing)
- Complete lab practice **RH134 CH 3.7.** (vi editing)
- Complete lab practice **RH124 CH 14.6.** (link files)
- Complete lab practice **RH124 CH 14.8.** (find, locate)

# Mid-term overview

Users, processes, permissions, security

Lecture 8

Chapter 2

# Users and groups

- All system activity in Linux is tied to a user and a group. All processes and all files have a user owner and a group owner. (UID, GID, EUID, RUID, EGID, RGID)
- Locally defined user accounts are stored on disk in the **/etc/passwd** file (public part of account information) and the **/etc/shadow** (secret part, e.g. password hashes, password aging, account expiry) file.
- Local group definitions are set in the **/etc/group** and **/etc/gshadow** files.
- Use **id**, **whoami** to print information about the current terminals (logged-in) user.
- Use **who**, **last** and **lastlog**, to print information about running user sessions and latest login events.

# Managing users

- You can manage before-mentioned text files manually. Or you can use the following command line tools. This conveniently also creates a home directory and sets its permissions, besides adding a new user entry.
- The **useradd [switches] username** command can create a user, note the following switches and default values
  - -c to set comment (Gecos) field (empty)
  - -s to specify shell (/bin/sh)
  - -d to specify home directory (/home/[username])
  - -g to specify primary group (UPG)
  - -G to specify list of supplementary groups (none)
  - -u to specify UID (add one to the last used UID)

Shell command line

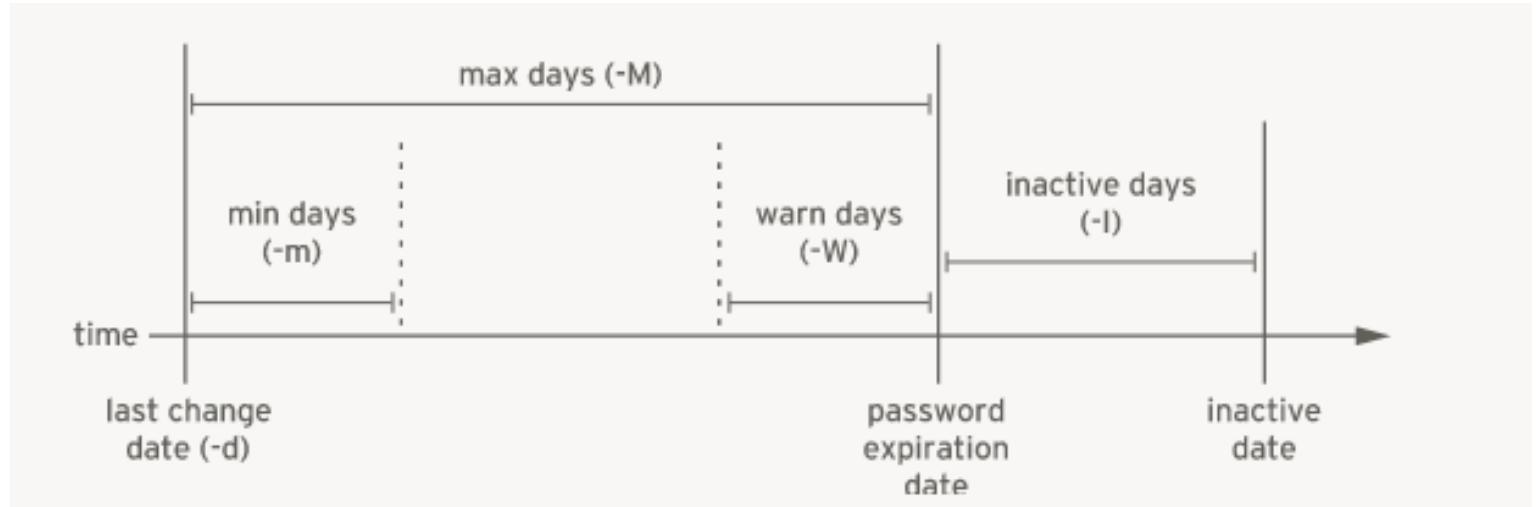
```
# useradd -c "John Smith" -u 1050 -G video jsmith
```

# Managing users

- You can modify an existing user with **usermod [switches] username** using similar switches. Also note the -L and -U switches to lock and unlock the password.
- The **userdel username** command removes the user. Using the -r switch also removes the users home directory from the filesystem.
- Note the seemingly conflicted UIDs in **RH124 CH 5.5.** when deleting and adding users.
- You can change your own password with **passwd** command. Only root can change passwords for other users. Root can do it like this: **passwd alice**

# Managing password aging

- You can adjust the timing fields of the /etc/shadow file with **chage**.



- Use **chage -E** to set absolute account expiry.
- You can also use the **usermod -e** command to set expiry.
- Also note the **-l**, **-d** options to **chage**.
- Refer to **RH124 CH5.9.-5.10.** for further information.

# Managing groups

- You can add a new group with **groupadd** command.
- You can modify a group with **groupmod**.
- You can remove a group with **groupdel**.
- You can modify group membership with the **usermod** command as described earlier. Note the **-g -G** and **-a** command switches.
- Refer to **RH124 CH 5.7.** for further
- Complete practice **RH124 CH 5.6.** (create users)
- Complete practice **RH124 CH 5.8.** (group membership)
- Complete practice **RH124 CH 5.10.** (chage -E)
- Complete practice **RH124 CH5.11.** (login.defs, chage)

# File permissions

- Each file has a user owner and a group owner. A permission triplet (r,w,x) is stored for every file for the user owner, for the group owner, and a triplet for everyone else (other). There is also a special permission triplet (sticky, setuid, setgid).
- Note the significance of x permission on directories (see table on next slide)
- The owner information of a process (source) that accesses a file (target) is compared by the kernel, it will decide which triplet to apply.
- The ACL scheme allows for additional rules besides the 3 basic triplets. Most importantly ACLs allow for named rules, that target individual users and groups. Each rule in ACL specifies a permission triplet to apply to a user or a set of users/group.

dir permissions	Octal	del rename create files	dir list	read file contents	write file contents	cd dir	cd subdir	subdir list	access subdir files
---	0								
-W-	2								
R--	4		only file names (*)						
RW-	6		only file names (*)						
--X	1			X	X	X	X	X	X
-WX	3	X		X	X	X	X	X	X
R-X	5		X	X	X	X	X	X	X
RWX	7	X	X	X	X	X	X	X	X

# Octal (numerical) representation

- The octal representation of permissions is also commonly used. Values for special bits are: setuid – 4, setgid – 2, sticky – 1

	u	g	o
access	r	w	x
binary	4	2	1
enabled	1	1	1
	1	0	1
result	4	2	1
	4	0	1
total	7	5	4

# File permissions

- Use **chmod** to change a files permissions, use **chown** to change a files owner (only root can do that).
- Use the **ls -l** switch to include permissions in the listing.
- The umask is a negative file mode applied to all newly created filesystem objects, use the **umask** command to set the value of the mask.
- Use **getfacl** to print the ACL ruleset for a file. Use **setfacl** to set or modify the ACL list belonging to a file or directory.
- The ACL mask is applied to all named entries and the group owner, before evaluating the triplet.
- Default entries only apply to directories, they will be automatically and recursively inherited to new sub-elements of the directory accordingly.

# Reading ACLs

## Shell command line

```
# getfacl some_directory
# file: some_directory
# owner: student
# group: controller
# flags: -s-
user::rwx
user:james:---
user:1005:rwx      #effective:rwx-
group::rwx         #effective:rwx-
group:sodor:r--   #effective:rwx-
group:2210:rwx    #effective:rwx-
mask::rw-
other::---
default:user::rwx
default:user:james:---
default:group::rwx
default:group:sodor:r-x
default:mask::rwx
default:other::---
```

# Setting ACLs

- An ACL can be set/replaced with the **setfacl -s** command. Use the **-m** switch to modify an ACL entry, use **-x** to remove an entry. Use the **-d** switch together with the former two to modify/remove a default entry.
- The format starts with **u:** **g:** **o:** or **m:** for user, group, other, and mask entries respectively.
- The next element is an object specifier (user/group name/id). **Mask** and **other** entries have blank specifiers.
- An empty user or group specifier corresponds to the user/group who owns the file.
- The last element is the permission set itself. e.g **rx**

Shell command line

```
# setfacl -m u::rwx,g:sodor:rX,o::- filename
# setfacl -x -d u:james filename
```

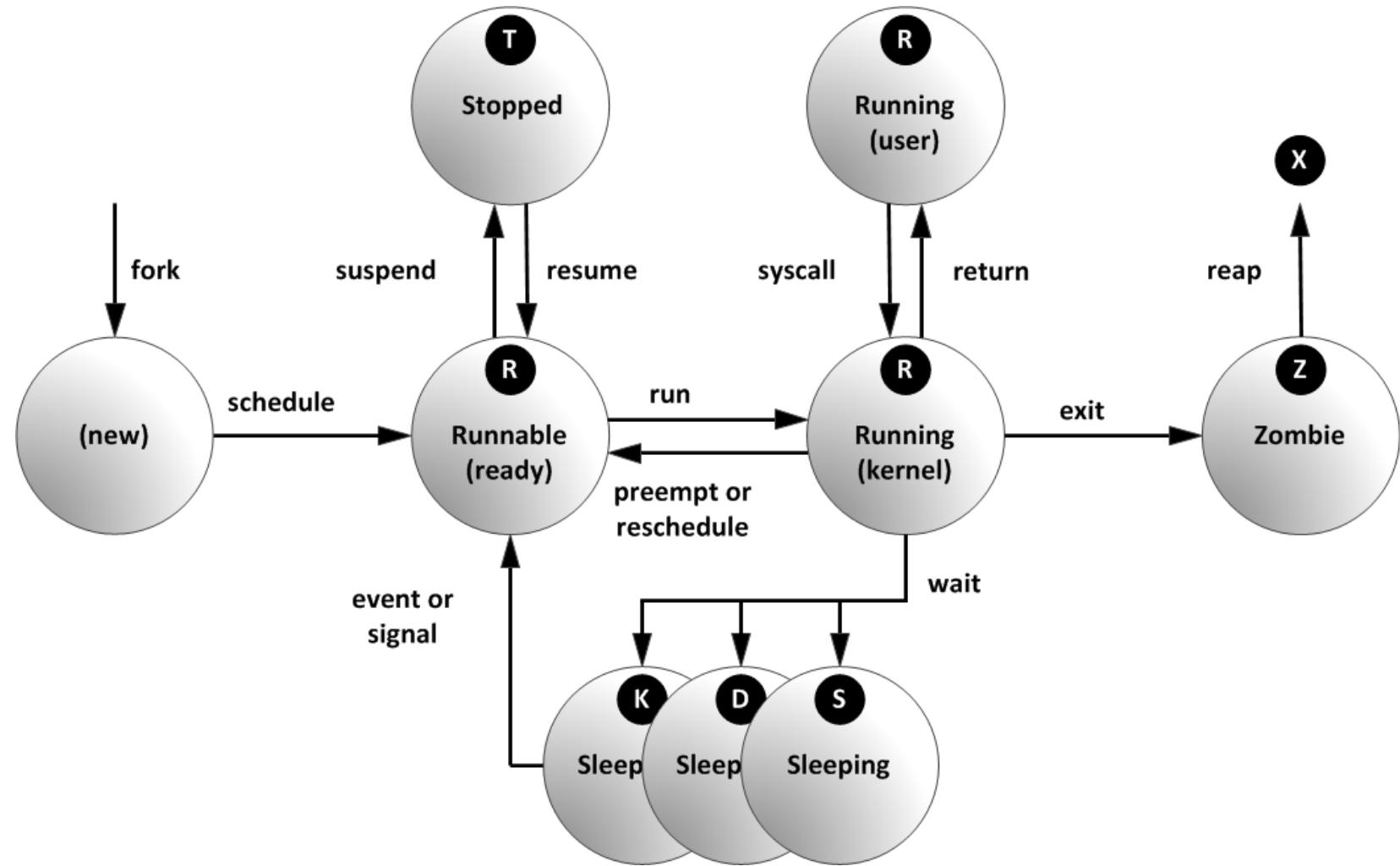
# Setting ACLs

- You can use the -R switch to apply ACLs to directories and sub-entries recursively. A permission set containing a X means to apply x permission only to directories, not files.
- You can remove all default ACL entries of a directory with `setfacl` using the -k switch. The -b switch removes ACLs completely.
- Modifying the ACLs with `setfacl` can cause the mask to be recalculated. To inhibit mask change use the -n switch to `setfacl`.
- Complete practices **RH124 6.6.** (`setgid`, `umask`), **RH124 6.4.** and **6.7.** (`chmod`, `chown`)
- Complete practices **RH134 CH6.4** (ACLs Thomas), **RH134 6.5.** (ACLs Holmes)

# Processes

- The processes in every operating system are handled by the scheduler. In Linux you can influence how the scheduler prioritizes each processes with the **nice** and **renice** commands. Processes can be listed with **ps**, **pstree** commands and be managed with the **top** utility.
- The processes can be targeted with signals by the scheduler. Use **kill**, **killall** and **pkill** commands to send signals. Some signals can be handled by the target process. Some signals originate from hardware events, some can be triggered by the user.
- The system load and uptime can be queried with **top** and files in the procfs (**cat /proc/loadavg**, **cat /proc/uptime**).
- Complete practice **RH124 CH 7.8.** (**top**), **CH 7.9.** (**top**, **kill**) and **RH134 CH 5.4-5.5.** (**nice**).

# Process states



# Processes

- Processes are not just created in Linux, a process can only be created by another process by cloning, then forking itself. This enforces the inheritance of security descriptors and privilege levels of child processes. A process owned by root can decrease its privilege level (set EUID, EGID) voluntarily.
- The only way to escalate privileges is through the **setuid** file permission mechanism. Several binaries have this bit set. Two of them (**su** and **sudo**) serve the purpose of running specific commands or whole user sessions as a different effective user (or group).
- Sudo allows for more refined control, the **source** (current) **user's** password may be required. To use **su**, you need the **target user's** password.
- Complete practice **RH124 CH 5.4.** (**su**)

# Jobs

- Processes and schedulers are operating system concepts. On the other hand jobs are shell related entities. Every pipeline of commands entered in the shell is a job. The processes in a pipe chain are the members of the same job.
- A shell can handle multiple jobs, but only one job can be in foreground. The fg. job is the only one that can get characters and control signals from the terminal input.
- Background jobs can be either be in executing state or can be suspended and resumed later. To start a job right in the background add an **&** (ampersand) to the end of the command. To suspend and send an already executing fg. command to background press **ctrl+z**.

Shell command line

```
# example_command | sort | mail -s "Sort output" &
```

# Jobs

- Jobs sent to background with `ctrl+z` are also immediately paused (suspended).
- To list jobs, use the **jobs** command.
- To bring a job back to foreground use **fg %jobnum**. This will also make it resume.
- To resume a suspended job while still keeping it in the background use **bg %jobnum**.
- Use **ctrl+c** to ask a process to terminate itself cleanly.
- Refer to **RH124 chapter 7.3.** for more.
- Complete practice **RH124 CH 7.4.** (jobs)
- Complete practice **RH124 CH 7.6.** (jobs, pkill)

# SELinux

- SELinux is an implementation for the Mandatory Access Control scheme for Linux based on NSA projects.
- In our case the targeted policy is the most important one. It builds mainly on the "type" element of the SELinux context and has rules like the following: "allow httpd\_t type processes to open httpd\_log type files for append operation." The policy is compiled into a binary (AVC). It is usually not required to modify the policy, it is handled by RH developers and put in the package repository. Though you can refine the effects of the ruleset by managing SELinux booleans with **setsebool** and **getsebool** tools.
- System boot-up defaults for SELinux mode and policy are stored in **/etc/selinux/config**, you can switch between permissive and enforcing mode at runtime with the **setenforce** and **getenforce** commands.

# Get SELinux information

- You can use the **sesearch -A | grep [keyword]** command to search the policy for expressions, types, rules. Use the -C -b switch in addition to restrict your search to SELinux booleans and print conditions.
- Try the **system-config-selinux** graphical tool for managing SELinux.
- Use the **-Z** switch in general to get SELinux context information. E.g.: **ls -Z**, **ps -Z**, **id -Z**
- You can use **sesearch -T** to look for transitional rules.
- Use the **runcon** command to run a command with specific SELinux context. This command, like others also falls under the rules of transition.
- Use the **man -k selinux\_keyword** to search for man pages containing the keyword. E.g. a type, a boolean, ...

# Manage SELinux context

- To explicitly set a file's context use the **chcon** command. Use the **-t** switch to only alter the type attribute. Use the **restorecon** command to reset a file's context to the database defined, location dependent, default value. The **-R** switch stands for recursive, **-v** stands for verbose.

## Shell command line

```
# chcon -t httpd_sys_content_t /var/www/html/app1
# restorecon -Rv /var/www/html
restorecon reset /var/www/html/file1 context
unconfined_u:object_r:user_tmp_t:s0
          -> system_u:object_r:httpd_sys_content_t:s0
# semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
# restorecon -RFvv /srv
```

- To change the database of default file-contexts use the **semanage fcontext** command. Use the **-l** switch to list the current database. Use **-d** to delete a line, use **-a** to add a line. Note that the order of evaluation is the same as the order of entry (overlapping rules for subdirs).

# Troubleshoot SELinux

- The most common reason for AVC denials is an incorrectly set file context. (E.g. file moved in a new location) → use **restorecon**. This has the most narrow impact on system security, it only affects the given file.
- Use **sealert -l [uuid]** command to get more details on a specific AVC denial. Its ID will be in the system log.
- Complete practice **RH134 CH.7.4.** (setenforce)
- Complete practice **RH134 CH.7.6.** (semanage fcontext)
- Complete practice **RH134 CH.7.8.** (SE booleans)
- Complete practice **RH134 CH.7.10-7.11 .** (sealert, restorecon)

# Mid-term overview

Storage

Lecture 8

Chapter 3

# Block devices, partitions, filesystems

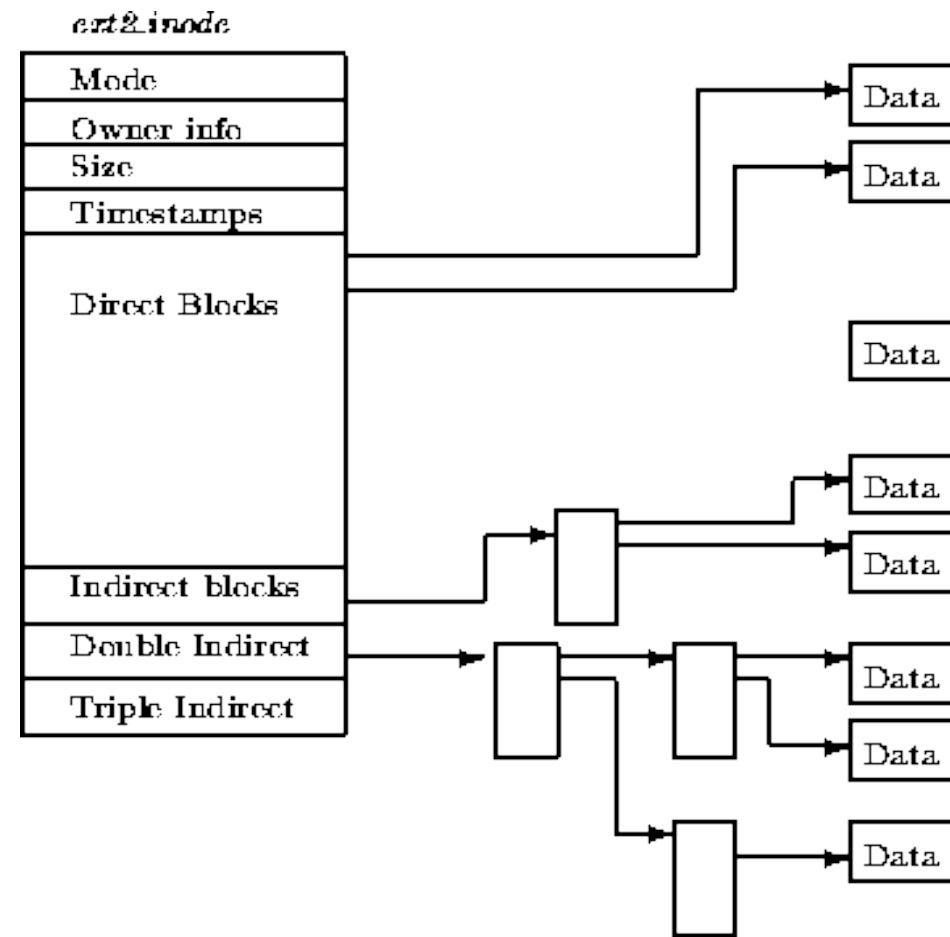
- Block devices are also called raw storage as they only provide a range of hardware addressable storage compartments, no structure or hierarchy is available.
- By partitioning, a block device can be divided into logically separated segments.
- LVM, besides simple logical segmentation, also provides additional features, like space aggregation, live resizing, caching, thin volumes, snapshots, etc... LVM relies on the device-mapper framework, LVs are still raw block devices. Block device can be stacked on each other with dm (for e.g. crypto, cache, other logical block devices.)
- A filesystem standard describes the disk storage format and a set of management algorithms. An FS provides a clear tree structure for organizing data. It may also provide advanced features, like subvolume snapshots.

# Block devices, partitions, filesystems

- LVM has 4 major component layers: **physical volumes**, which add up the the extents of **volume groups**, and **logical volumes**, that are carved out of the free extents of a VG. The 4<sup>th</sup> (optional) layer is for LVM **pools**.
- Filesystems are created on top of block devices (for e.g. partitions, logical volumes or dm-crypt devices).
- **Swap** space can be used to temporarily store infrequently used memory pages on disk. The kernel swaps pages in- and out of the disk to/from memory as needed. Swap is not considered a filesystem.
- The procedure of merging a filesystem into a directory of the existing directory tree is called **mounting**.
- Unix filesystems have **symbolic links** (textual path references, can point outside FS) and **hard links** (numerical reference to structure/element identifiers within a FS).

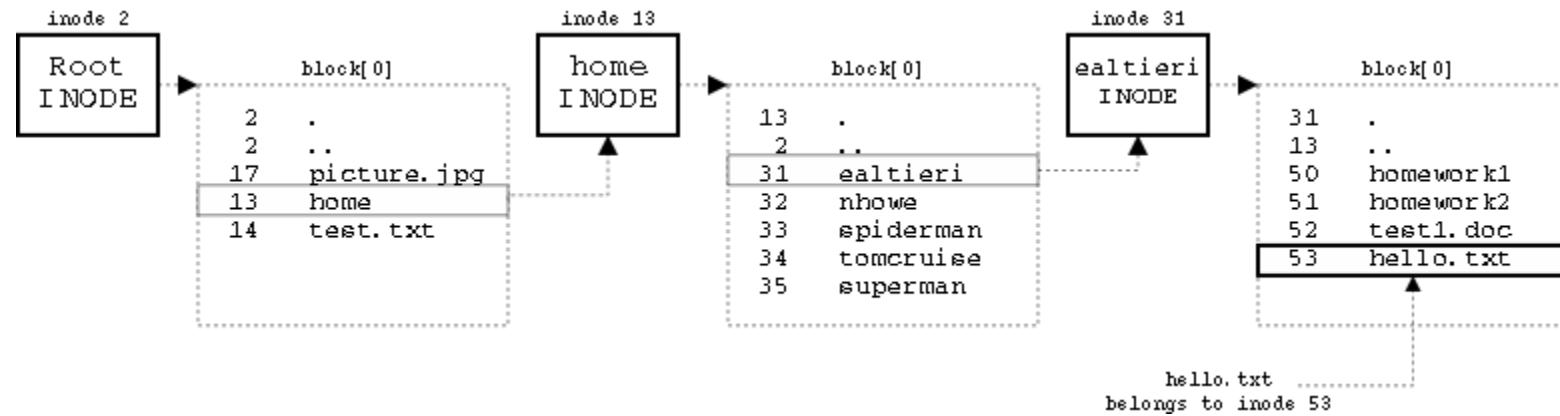
# The ext2 filesystem storage format

- In case the list of data blocks can't fit in a single inode, one to three levels of indirection blocks are inserted.



# The ext2 filesystem storage format

- Directory inodes contain a list of lines (records), one for each entity in the directory. A record is a mapping between filename and numerical inode identifier.
- Each record contains 5 fields: inode number, record length, filename length, file type (pipe, symlink, socket, chardev, blockdev, directory), and the filename itself.
- Remember lecture 3! The directory permissions (rwx) now all make sense. Manipulating a directory is the same as modifying or accessing this list of records.



[http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2\\_locate.png](http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2_locate.png)

# File system management commands

- You can use the **mount** command to list active mounts or to join an additional FS into the directory tree. Use **umount** for the reverse operation. Use **swapon** and **swapoff** to activate/deactivate swap space.
- Use **lsblk** to list all block devices as they are stacked (tree). Use **blkid** for printing UUID and header type of block devices.
- Use **mkfs.[fstype]** for creating filesystems. Use **mkswap** for creating a swap header.
- Use **du** for a file's/dir's space usage, **df** for free space info. Issue **free** for memory and swap usage info.
- Use **lsof** and **fuser** to look up open files and corresponding processes.
- Consult the **/etc/fstab** file for auto-activating block devices at system boot.

# Partitoning and LVM commands

- Use **fdisk /dev/block\_dev\_name** for partitioning disks with the MBR partitioning scheme.
- Use **gdisk** the same way for GPT disks.
- The **cfdisk** utility is a text-menudriven partitioning tool.
- The **pvcreate**, **pvremove**, **pvdisplay**, **pvs** commands can be used to manage the physical volume header on block devices/partitions.
- The **vgcreate**, **vgremove**, **vgdisplay**, **vgs** and **vgchange** commands stand for managing volume groups. Use **vgextend** or **vgreduce** to assign/unassign PVs to VG.
- Use **lvcreate**, **lvremove**, **lvchange**, **lvresize** to manage logical volumes. Use **lvdisplay** or **lvs** to print LV information.
- Each filesystem has its own FS resizer tool, e.g. **resize2fs**.

## Relevant practices

- Complete practice **RH124 CH.14.4.** (mount)
- Complete practice **RH124 CH.14.9.** (du, blkid, ln, find)
- Complete practice **RH134 CH.9.2.** (fdisk, mkfs)
- Complete practice **RH134 CH.9.4.** (swap)
- Complete practice **RH134 CH.9.5.** (gdisk, swap, mount)
- Complete practice **RH134 CH.10.4.** (lvm)
- Complete practice **RH134 CH.10.6.** (lvextend, xfs\_grow)
- Complete practice **RH134 CH.10.7.** (lvextend, xfs\_grow)

# Mid-term overview

Package manager, boot process, systemd

Lecture 8

Chapter 4

# Distributions, repositories, packages

- Linux vendors/families usually have their distinctive package management systems with their own format.
- Package managers serve the purpose of version tracking/managing installed software.
- Packages are organized into package repositories. Repositories are consistent regarding dependencies and compatibility, packages inside a repository are maintained by the same organization.
- The RPM package format also allows for cryptographically signing released software to prevent malware injection.
- In the RH family repositories are stored under the **/etc/yum.repos.d** directory, and they can be enabled/disabled with the **yum-config-manager** command. (**--enable=reponame**, **--disable=reponame**)

# yum command

- Yum is an easy to use front-end to the rpm manager. You can use it to install/remove/update/search packages.
- By issuing **yum list** or **yum list installed** you can list all the available/installed packages.
- You can search for patterns in both package name and description text with **yum search all 'keyword'** command. To search for file (remember: a Linux command is a file in /bin or /sbin) use **yum provides \*filename** or **yum provides /exact/path/to/file/name**
- Issuing **yum update *packagename*** will update a package to the latest version that has been made available in the repo. Omitting *packagename* will update all packages.
- You can use the **-y** switch to skip the confirmation on doing a yum operation.

# yum command

- Use **yum install *packagename*** to install a package from the set of enabled repos. Use **yum localinstall *some-file.rpm*** to install from an rpm from a file in the file-system. Use **yum remove *pattern*** to erase one or more packages. This will also remove packages that depend on the removed package(s).
- Issuing **yum info *package*** will show the detailed description of the package.
- Some software suits are organized into groups, for e.g. "Development tools". You can do group operations by using the group sub-command: **yum group list**, **yum group info *grpname***, **yum group install *grpname***, etc..
- Yum has a history feature: **yum history [*nr*]** will print details of a transaction defined by *nr*. Omitting *nr* will list all recent transactions sorted by tr. number. You can roll back a transaction with **yum history undo *nr*** command.

# rpm command

- You may use the **rpm** command for low level interaction with the package manager. The yum front-end actually executes several rpm commands behind the scenes.
- Only the query commands will be discussed in this course. You can use the **-q** switch to rpm to select query mode.
- The **rpm -q -a** command will list all installed packages, **-q -f *filename*** will find what package holds *filename*. Use **-q -i** to display package info, **-q -l** to list files in a package, **-q -c** to list config files only, **-q -d** to list documentation files only. The **-q --changelog** switch will print the version history and comments of a package, **-q --scripts** will show the executable scripts in the package.
- Use **rpm2cpio *file.rpm* | cpio -idv** to extract all files from an rpm to the current directory.

# GRUB

- The self-extracting kernel binary (**vmlinuz**) and the **initrd** image is loaded and executed by the bootloader. The kernel binary itself has **command line arguments**, which can be edited from the **GRUB boot menu** with the **e** key.
- Boot flow is controlled by the **rd.break**, **rd.debug**, **systemd.unit=some.target** kernel cmdline arguments (among many others).
- GRUB configuration is stored in **/boot/grub/grub.cfg** file. This file is auto-generated with the **grub2-mkconfig** command. The configuration for programmatically generating the **grub.cfg** is stored in **/etc/default/grub** file.
- You can use **grub2-install --root-directory=/root/of/fs /dev/boot\_device** to install a GRUB instance.
- Use **dracut** to generate a new initial RAMdisk image.

# systemd

- The **PID=1** process has a special role in the Linux system startup. It is the only userland process executed by the kernel. By executing several other processes **PID1** (`/sbin/init`) is responsible for setting up filesystem mounts, network connections, starting all the server processes, daemons, login terminals.
- Systemd is the **PID1** process of many modern Linux systems. The **systemd** suite is a complex system manager with a substantial set of utilities and implicit daemons, like a built-in system logger (`systemd-journald`) and a network manager (`systemd-networkd`).
- Systemd has very clear declarative ini style (sections, key=value pairs) **unit files** for services, sockets, mounts. Units are organized into **targets**. Units are executed in **parallel** with respect to their dependency requirements.

# systemd

- Switching targets is done with the **systemctl isolate *some.target***, where target can be for e.g. graphical.target or multi-user.target.
- The default target can be set by **systemctl set-default *some.target*** command.
- (Re)Starting and stopping units is done with **systemctl start/stop/restart *unit.name***, for e.g systemctl restart sshd.service.
- Setting a unit for autostart is done with **systemctl enable/disable *unit.name***.
- Issuing **systemctl mask *unit.name*** will hide a units, it can not be enabled in this state. Issue unmask to revert.
- Use **systemctl status** to query all services or **systemctl status *unit.name*** to query a single unit.

# systemd

- Systemd units are stored in `/usr/lib/systemd/system`. This folder is managed by the package manager. User defined units go to `/etc/systemd/system` folder. If you want to modify a unit, place a unit file with the same name to the `/etc` folder of systemd instead, this will override the `/usr` version of the file with the same name.
- Use **systemctl list-units** or **list-unit-files** to list names or filenames. Use **--type=service** to filter for service units, use **systemctl --failed** to filter to failed units.
- Systemd has integrated kernel cgroups features.
- Use **systemd-cgls** to print the control group tree.
- Issue **systemd-cgtop** for a top-like utility that sorts systemd slices based on resource usage.

## Relevant practices

- Complete practice **RH124 CH.13.6.** (yum install)
- Complete practice **RH124 CH.13.8.** (repo enable, yum)
- Complete practice **RH124 CH.13.10.** (rpm local install)
- Complete practice **RH124 CH.13.11.** (repo enable, yum)
- Complete practice **RH134 CH.13.4.** (lost root password)
- Complete practice **RH134 CH.13.6.** (emerg.targ, fstab)
- Complete practice **RH134 CH.13.8.** (grub2-mkconfig)
- Complete practice **RH124 CH.8.2.** (systemd units)
- Complete practice **RH124 CH.8.4.** (s. start/enable)
- Complete practice **RH124 CH.8.5.** (s. start/enable)

# Mid-term overview

Networking, firewalld

Lecture 8

Chapter 5

# Network Manager

- Network Manager is a resident process that initializes automatic network connections and handles all the network related events (like for e.g. plug of a cable, loss of a WiFi link) while it runs.
- Its global configuration is stored under the **/etc/NetworkManager** directory. Connection configurations are to be found under **/etc/sysconfig/network-scripts/ifcfg-\*** files.
- It has CLI (nmcli), TUI (nmtui), and GUI (control network, nm-connection-editor) interfaces.
- Device objects in NM represent Linux network interface cards. Connection objects represent settings applicable to interfaces. Each interface can have several connections saved on disk. You can dynamically switch between saved connections.

# nmcli command

- Examples for adding connections:  
**nmcli con add con-name "default" type ethernet  
ifname eth1**
- **nmcli con add con-name "demo" type ethernet ifname  
eth1 autoconnect no ip4 172.25.0.10/24 gw  
172.25.0.254**
- You can switch between connections by issuing **nmcli  
con up "demo"** or **nmcli con up "default"**.
- You can administratively disable an interface with **nmcli  
disconnect dev eth1**. This will also terminate its current active connection.
- You can bring a connection down by **nmcli con down  
"demo"**. This will not disable the interface, meaning that in the event of a link change, the autoconnect connection for the device will be activated.

# nmcli command

- You can modify an existing connection with **nmcli con mod *con-name* *property value*** like commands. E.g.: **nmcli con mod "static" connection.autoconnect yes**
- Some properties can have multiple values, like for e.g. `ipv4.dns`, since it is valid to have multiple resolvers configured in a network. You can assign and overwrite previous values the same way, e.g. **nmcli con mod "static" ipv4.dns 8.8.8.8** But in case of multi-value properties you can also prefix the setting with `+` or `-` to add/remove an entry, without modifying the other values of the same property. E.g. `... +ipv4.dns 8.8.8.8`
- To delete a connection use **nmcli con del "demo"**.
- To display connection details issue **nmcli con show "connection-name"**. To list connections simply omit the connections name. To list devices issue: **nmcli dev status**

# Manual IP configuration

- To manually set the ip address of an interface, issue a command like this: **ip addr add 192.168.50.5 dev eth1**  
To remove: **ip addr del 192.168.50.5 dev eth1**
- To enable/disable an interface, use **ip link set eth1 up/down**.
- To add a routing table entry, issue **ip route add 10.10.20.0/24 via 192.168.50.100 dev eth0**
- To remove it, use: **ip route del 10.10.20.0/24 via 192.168.50.100 dev eth0**
- To set the default gateway: **ip route add default via 192.168.1.1**
- To manually edit the DNS resolver, modify **/etc/resolv.conf** with a text editor.
- To change hostname use: **hostnamectl set-hostname**

# Firewalld

- Just like NM firewalld is a resident process with several interfaces (CLI, TUI, GUI). It accepts commands and generates iptables rules at run-time for setting up the kernels Netfilter subsystem accordingly.
- Firewalld divides traffic into zones. Zones are implemented as chains on the iptables level.
- Each zone has a separate list of rules (allowed services). An incoming packet will be classified to a zone, then matched against the zones rules and its fate will be determined.
- An incoming packet is checked for it's source address, to determine the applicable zone. If no zone rule matches to the source address, then the zone selection decision will be based upon the interface that received the packet. All interfaces belong to exactly one zone.

# firewalld services

- A service is defined by a set of ports, for e.g. the ssh service is equivalent to the single port of 22/tcp. The samba-client service consists of allowing two ports, 137/udp and 138/udp.
- A zone can have any number of enabled services. All ports, that do not belong to an enabled service of the zone, will not be available for communication.
- Use the **firewall-config** GUI utility for managing firewalld. Use **firewall-cmd** for command line management.
- Firewalld distinguishes a runtime configuration and saved state stored on disk, called the permanent configuration. Use **firewall-cmd --reload** to drop runtime and load from disk. Use --permanent to add a setting directly to the permanent configuration.

# firewall-cmd

- You can print all the services with **firewall-cmd --get-services**. Issue **firewall-cmd --get-zones** for listing the zones, **--get-active-zones** for printing only the zones that have at least one interface in them.
- Use the **--get-default-zone** switch to see the default zone for new interfaces.
- Use **--list-all-zones** to print detailed information about zones (for. e.g. interfaces, enabled services, etc.)
- Use **firewall-cmd --get-services** to list the available services.
- Issue **firewall-cmd --list-all** for printing the whole configuration.
- Use **firewall-cmd --set-default-zone=zone** to set the default zone to *zone*.

# firewall-cmd

- Take the **--add-source=10.1.0.0/16 --zone=trusted** example to assign a network to a zone. Use **--remove-source=10.1.0.0/16 --zone=trusted** to revert such a change.
- Use the following example to add an interface to a zone:  
**firewall-cmd --add-interface=eth0 --zone=trusted**  
Use **--change-interface=eth0 --zone=dmz**, to reassign the interface to a different zone.
- Use **--add-service=mysql --zone=dmz** to enable a service such as mysql to be available in the dmz zone.
- Use **--add-port=17784/tcp --zone=dmz** to allow non-standard service ports to a zone, without creating a service for it.
- Use **--remove-port** and **--remove-service** logically.

## Relevant practices

- Complete practice **RH124 CH.11.4.** (ip utility)
- Complete practice **RH124 CH.11.6.** (nmcli)
- Complete practice **RH124 CH.11.8.** (ifcfg files manually)
- Complete practice **RH124 CH.11.10.** (hostnamectl)
- Complete practice **RH124 CH.11.11.** (nmcli con)
- Complete practice **RH134 CH.14.2.** (firewall-config GUI)
- Complete practice **RH134 CH.14.3.** (firewalld add service)

# Recommended reading

Lecture 8

Chapter 6

Relevant  
chapters in RH:

Read by next  
lecture:

- See list in Lectures 1-7

- RH124 CH10 (Logs)
- RH134 CH4 (Scheduling tasks)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 8, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# Logging

Lecture 9

Chapter 1

# Logs

- There are several reasons to keep logs in a computer system.
  - Troubleshooting: Daemons, as told earlier, don't communicate to a user directly, instead they write log messages. The occasional errors and warnings encountered by a daemon can be read from the logs.
  - Performance evaluation: It is useful to know how many request a daemon serves, what load the system is under in specific time periods, what the bottlenecks are.
  - Security/accountability: It is important to know what happens in a computer, who logged in and did what. Logs can also reveal where an attack originated from.

# System logs

- Logging is a feature that also involves the kernel. Programs call a syscall to submit log entries. The entries are buffered in the kernel, and periodically collected and then saved on permanent storage by a logger daemon. Permanent storage can either be a local file or through network, involving another log collector daemon, which in turn usually also stores the logs on permanent storage.
- It is possible that some embedded Linux systems don't have a permanent storage to store the logs on. In this case the logs are stored in memory until the device is rebooted.
- It is possible to have multiple logging daemons, it is also possible for a daemon to record the logs both locally and forward via network.

# Traditional system logs

- Traditional logs are **regular text files**, that can be viewed/searched with a pager or standard commands like: **less, cat, grep, tail, ...**
- Traditional logs don't have standardized storage locations, filenames, methods or formats. There are distribution specific conventions though.
- One of the earliest logger daemons is **syslogd**. (By Eric Allmann, 1980)
- Syslogd defines 24 log facilities and 8 severity levels for entries.
- There are several standards for the text format, like **RFC3146, RFC5424**.

# Log priorities

Number	Severity	Description
0	Emergency	Panic condition, system unusable.
1	Alert	Immediate correction required.
2	Critical	Critical condition.
3	Error	Non critical error, non urgent failure (for e.g. notify developer).
4	Warning	Not an error condition, but can indicate an upcoming failure (for e.g. FS is almost full).
5	Notice	Unusual or significant conditions that are not errors. No urgent action required.
6	Informational	Normal operation messages. Can be examined for throughput measurement.
7	Debug	Detailed information used for development and troubleshooting.

# Log facilities

- The sources of logs are categorized (based on the emitting process) into facilities:
- Priority = Facility \* 8 + Severity
  - 0-kernel messages 9-clock daemon
  - 1-user-level messages 10-security/authorization
  - 2-mail system 11-FTP daemon
  - 3-system daemons 12-NTP subsystem
  - 4-security/authorization 13-log audit
  - 5-internal Syslog msgs. 14-log alert
  - 6-line printer subsystem 15-clock daemon
  - 7-network news subsys 16-23 locally def. use (local0-7)
  - 8-UUCP subsystem

# Log format

- Log line format with example (RFC 5424)

2003-10-11T22:14:15.003Z mymachine.example.com su  
<47> ID: 73214 user.error – BOM 'su root' failed for  
lonvick on /dev/pts/8

- 2003-...              Timestamp RFC8601
- mymachine...      Hostname (FQDN)
- su                      Process name
- <47>                      Process ID (PID)
- ID: ...                Message ID
- user.error              Syslog facility.severity
- BOM                      Message marker
- 'su root' fai...      Message text

# Rsyslog

- Rsyslog is the log handler service in many legacy Linux systems. Systemd has its own built-in logger called **systemd-journald**. Many systemd computers still have **rsyslog** enabled for compatibility reasons. This does not have any drawbacks, other than all messages are being logged and saved at least twice.
- Rsyslog has it's configuration settings in **/etc/rsyslog.conf** and **/etc/rsyslog.d** directory. The settings describe what source (facility) and priority of messages go to which file in the filesystem.

`/etc/rsyslog.conf [excerpt]`

```
kern.* /var/adm/kernel
kern.crit /dev/console
kern.info;kern.!err /var/adm/kernel-info
*.emerg @syslog-host
mail.* @mail-log-host
```

# Conventional log locations

- There is no standardized scheme for sorting and storing logs. RH systems follow the conventions below (this is regulated in rsyslog.conf):
  - `/var/log/secure` login and auth. related messages
  - `/var/log/cron` msgs. of scheduled tasks
  - `/var/log/boot.log` startup related log entries
  - `/var/log/maillog` mail system log entries
  - `/var/log/messages` - all other messages that don't go to the first 4 files
- Some daemons implement their very own logging subsystem, unrelated to syslog or standard system calls and log features. An example would be the apache daemon, which writes several log files on its own to the `/var/log/httpd` directory. (like `access.log`, `error.log`)

# Logrotate

- Log files are constantly getting lines appended to. Therefore they can grow quite large, and difficult to handle/backup/search.
- There is an automated mechanism for slicing large log files to smaller pieces, numbering them (by sequence or by date), and removing the oldest slices. This is called logrotate.
- Rotation can be manually triggered by issuing **logrotate**. The automatic scheduling of log rotation is described in **/etc/logrotate.conf** and **/etc/logrotate.d** directory. The logrotate command itself is run periodically by **cron**.
- Look at logrotate.conf, note the following directives: create, missingok, notifempty, size, weekly, daily, compress, rotate, dateext, ...
- Look up **RH124 CH 10.1.** and **CH 10.3.** for further.

# Analyzing logs

- Use `grep "expression" /var/log/messages` to look for entries matching "expression".
- Use for e.g. `tail -n 30 /var/log/secure` to print the last N lines of a file.
- Use `tail -f filename` to "follow" a file. This will print any new lines added to the file immediately. Press `ctrl+c` to terminate watching.
- Issue `logger -p facility.severity "Your own message"` to create a log entry from a script or command line.
- Use `dmesg` to display kernel messages only. You can use grep or tail or less also with dmesg (using pipes), like `dmesg | grep something` or `dmesg | tail -n 20`.
- Complete practice **RH124 CH 10.4.** (find log entries)

# systemd-journald

- Systemd has its own full-featured log service, called `journald`. It is enabled by default on systemd computers. It is possible to have other logger(s) running along with **systemd-journald**.
- Configuration is stored in `/etc/systemd/journald.conf` file. Journald has its own logrotate mechanism built in. (See `SystemMaxUse=`, `SystemMaxFiles=` and related directives.)
- One of the biggest arguments against systemd was that it breaks the convention of storing the logs in plain text format. Journald has a binary database, that can hold the entries more efficiently. It is also easily searchable by time, boot sequence number, systemd unit, ...
- The tool for reading systemd logs is `journalctl`. This utility is built in all systemd systems, therefore the binary format does not have any practical drawbacks.

# journalctl

- Use **journalctl -f** to follow the journal, apply **-n** switch to limit output to the last N lines of the log, just like with the tail command.
- The output of journalctl will always be automatically piped to a pager, like **less** (if it is present on the system).
- Use **journalctl --since "2016-03-19 15:30:00" --until "2016-03-19 20:00:00"** to narrow down to a time period.
- Use **journalctl -b -1** to see the messages belonging to the previous boot and run of the computer, use **-b -2** instead to view the session 2 reboots ago, etc...
- Use **journalctl -p warn** to filter for priority, use **-u sshd.service** to filter to a specific systemd unit.
- Use advanced filters like **\_PID**, **\_EXE**, **\_SELINUX**. (List them with **journalctl -o verbose**)

# systemd-journald

- Issue **journalctl --rotate** to manually trigger a journal rotation.
- By default systemd autoselects the log destination. It first looks for a directory under `/var/log/journal`.
- If it is found, it will store logs there (`/var` is usually a disk mount – persistent logs).
- If not it will create `/run/log/journal` and write logs there. (`/run` is a `tmpfs`, it will be lost on reboot – volatile logs)
- Note the `Storage=` directive in `journald.conf`
- Read **RH124 CH 10.5.** and **10.7** for further about `journald` and `journalctl`.
- Complete practice **RH 124 CH 10.6.** and **10.8.** (`journalctl` and persistent journals.)

# System clock

Lecture 9

Chapter 2

# Time and timezones

- Keeping the system clock accurate is important for many reasons. For e.g. the timestamp of log entries, the timestamp of files synchronized to other computers. Some security protocols require that the clock of computers involved are exactly synchronized, because message signatures have a validity time window.
- To set the computers local RTC chip use **timedate-ctl** command. Linux computers have their clocks set to UTC by default, and calculate local time based on RTC and timezone offset. Windows computer on the other hand tend to set the RTC chip to the local time.
- Use the following format to set the local time: **timedate-ctl set-time 9:00:00** Issue for e.g. **timedate-ctl set-timezone Europe/Budapest** to set timezone. Issue **timedate-ctl list-timezones** for a list.

# NTP

- Issuing `timedate-ctl` without arguments will print current settings.
- NTP (Network Time Protocol) is a distributed network protocol for keeping the RTC accurate. NTP has a built-in algorithm for compensating varying network delay, that involves multiple network based time sources.
- To enable your system to automatically correct even a small skew or drift of the RTC: issue **`timedate-ctl set-ntp true`**.
- The daemon involved behind the scenes is called `chronyd`, this daemon talks the NTP "language". NTP uses a distributed peer-to-peer service model, any NTP synchronized peer can act as a time source itself.
- Issue **`chronyc sources -v`** to view the configured active time sources with the `chrony` client utility.

# timedate-ctl

## /etc/chrony.conf [excerpt]

```
server 1.europe.pool.ntp.org
server 2.europe.pool.ntp.org

maxupdateskew 5
driftfile /etc/chrony.drift
rtcdevice /dev/rtc

### ACTING AS AN NTP SERVER
allow 192.168/16
deny 192.168.100/24
```

- Read **RH124 CH 10.9.** for further details on timedate-ctl and chronyd.
- Complete practice **RH124 CH 10.10.** and **10.11.** (adjust system time, NTP)

# Scheduling tasks

Lecture 9

Chapter 3

# Scheduled tasks

- In Linux there are three basic concepts for settings execution of commands to a specific time. All of them is controlled by the **crond** daemon in a modern Linux.
- The **at** utility allows to postpone the execution of a command to a very specific time. This will be a one-time execution. The **crontab** mechanism allows for executing recurring tasks with a specific frequency at a specific time. The **anacron** mechanism is also used for scheduling an execution frequency for commands, but it will also consider missed events. This is useful for computers than are not powered-on at all times, but at irregular periods instead (like for e.g. notebooks or home PCs).
- Note that **systemd** has its own scheduling mechanism in the form of **.timer** units. Transient timers (**systemd-run**) are the equivalents of **at**.

# at

- The **at** command runs a set of commands or a script file at a specific time, but only once. There are 3 ways to tell at what to do and several formats to tell when to do it:
  - You may pass a filename to be executed with the **-f** switch. E.g. **at noon -f start-backup.sh**
  - You may redirect a set of commands to it. E.g. **echo "poweroff" | at noon Monday**
  - You may also use the built in editor of at. Finish editing with **ctrl+d**. Example below:

shell command line

```
# at 9:30 PM Tue
at> if [ -e /root/tuesday_sql_mess ];then
at>   cleanup_sql.sh
at> fi
at> ^D
job 1 at Tue Nov 16 09:30:00 2017
```

at

- The at command has a very flexible time input, it can accept several formats, including natural English language time specifiers. A few examples follow:
- at noon..., at midnight..., at teatime...
- at tomorrow... (the same time tomorrow), at Friday... (same time on Friday), at next Monday..., at next week... (the same day and time as command is given, only next week)
- at 2:34 PM..., at Oct 10..., at 10/21/2017 ... (absolute specifiers)
- at now..., at now + 30 minutes ... , at now +3 days... (relative specifiers)
- Specifiers can also be combined: at 4PM + 2 days, at 2 PM next Thursday, at 8:23 PM 21/10/2017, at 7:45 tomorrow

- You can list currently queued at jobs with **atq** (alias to **at -l**), remove a job with **atrm** alias to (**at -r**).
- To list the command(s) in a given job, issue **at -c 4**, where the number is the job number according to atq output.
- The **-q** switch to at will tell the queue name. At has queues designated by letters starting from **a** to **z**, **a** is default. The higher the queue letter, the higher the niceness of the process.
- Jobs are stored in text files under **/var/spool/cron** directory.
- Read **RH134 CH 4.1** for further on at.
- Complete practice **RH134 CH 4.2.** (**at, atq, atrm**)

# cron

- The crontab mechanism allows for scheduling recurring tasks.
- All users have their own crontab file. Use **crontab -l** command to print current crontab, use **crontab -e** to open the file in the default editor, like vi.
- Root can use **crontab -u *username*** to manage a specific users crontab. Regular users can only edit/list their own.
- There is a global crontab file (**/etc/crontab**), which has one additional column, that sets the username to the executed command.
- The global crontab file holds entries (run-parts) for running a set of scripts hourly, daily, weekly. You only have to copy or link executable files into **/etc/cron.hourly** , **/etc/cron.daily**, etc... directories respectively to have them run regularly.

# crontab format

## /etc/crontab

```
# Min Hour Day_of_M Month D_of_W User Command
  0   2     12          *      *     bob   /usr/bin/some
  30  8-16  *          *      1     root  /some/script1
  /5   8-12  1          *      *     joe   /some/script2
  0   1     3/7        1      *     jane  /some/script3
  /7   /2     13        *      5     root  /some/script4
```

- *Commands* can have any number of arguments. Before the command field: fields can be separated with any empty space characters like one or more space(s) or tab(s). Inside the command field: all chars (incl. spaces) are interpreted as they are part of the command line.
- Monday is considered the first day of week (1), Sunday can both be represented with 0 or 7.
- Intervals can be specified with a - (dash), repetitions can be specified with a / (slash). This is actually evaluated by calculating the division remainder. For e.g. /5 will run at 0, 5, 10, 15, etc... for e.g 3/7 will run at 3, 10, 17, 24, 31, etc..

# anacron

- Anacron can start jobs after their actual scheduled time if an event was skipped because of a poweroff or hibernation state.
- The **/etc/anacrontab** file contains 4 fields:
  - frequency in nr. of days.
  - delay in minutes from startup
  - time-tracking file's name
  - command
- Read **RH134 CH 4.3.** for further on cron and **4.5.** for anacron.
- Complete practice **RH134 CH 4.4.** and **4.6.** (crontab)

# cleaning temporary files

- Systemd has a service for cleaning up temporary files from both persistent (`/var/tmp`) and volatile storage (`/run`). It is called `systemd-tmpfiles-clean.service`, a `.timer` unit with the same name ensures, that this service is ran **15** minutes after system startup and then every **24** hours.
- The tasks to do when such an event triggers is described in configuration files under `/usr/lib/tmpfiles.d`, `/run/tmpfiles.d`, `/etc/tmpfiles.d`. If these directories have the same filename then the etc instance overrides those listed before.
- Consider the `d`, `D`, `Z`, and `L` directives. An example line:  
**D /home/guest 0700 guest guest 1d**
- Read **RH134 CH 4.7.** for further on systemd's temp file management.
- Complete practice **RH134 CH 4.8.** (temp files)

# FTP server

Lecture 9

Chapter 4

# FTP

- File Transfer Protocol involves two channels: a control channel (for commands and replies) that is always initiated by the client to tcp port 21 on the server, and a data channel (bulk file content) that depends on mode.
- The active mode is the original FTP mode. In this mode the server initiates the data connection for file transfer from it's tcp/20 port to a dynamic unprivileged port ( $>1024$ ) on the client side. This can be problematic if the client is behind a firewall or NAT, since this channel is not in the client side NAT table, because it was not initiated by the client.
- In case of the passive FTP mode the server offers a dynamic port above 1024 for the client to connect to. This complicates the server side firewall setup, but does not require any special measures from the client (customer).

# vsftpd

- RH repositories only contain the **vsftpd** (Very Secure FTP daemon) as a standalone FTP service.
- Vsftpd **separates** the privileged parent process and spawns un-privileged child processes for client sessions.
- The sensitive operations are handled by the parent process. Children communicate their requests to the parent through a local socket, all requests are verified.
- Child processes are **chroot**-ed to the relevant directory.
- Configuration files can be found under /etc/vsftpd directory. The configuration consists of **<directive>= <value>** style lines and some other files, that contain lists. (lists of allowed users for e.g.)
- You can manage the service with systemd, the unit name is vsftpd.service. E.g: **systemctl start vsftpd**

# vsftpd

- Let's see the most important configuration directives:
  - `listen_address`= tells what network IFs to listen on.
  - `listen_port`= defaults to standard ftp port (21)
  - `max_clients`= the max. number of simultaneous connections
  - `pasv_enable`, `pasv_max_port`, `pasv_min_port`: passive mode settings
  - `port_enable`= allows active mode connections.
  - `anon_max_rate`= max bandwidth for anonym. users
  - `dual_log_enable`= allows for both syslog and simultaneous custom vsftpd logging.
  - `cmds_allowed`= holds a list of commands that are allowed over ftp

# vsftpd

- Let's see the most important configuration directives:
  - `message_file`= tells filename for directory messages
  - `force_dot_files`= show hidden files (starting with a `.`)
  - `guest_enable`= enables guest user
  - `chroot_local_user`, `chroot_list_file`= a file containing a list of chroot-ed users.
  - `anon_upload_enable`= allows anon. user to upload
  - `ftp_username`= the local user mapped to anonymous logins
  - `local_enable`= enable locally defined users to log in.
  - `banner_file`= text to display at client side on conn.
  - `userlist_deny`, `userlist_file`, `userlist_enable`= specifies the set of user that are allowed to log in.

# SELinux and vsftpd

- Relevant SELinux types are **public\_content\_t** and **public\_content\_rw\_t**, you should set the file context of FTP accessible directories accordingly. E.g.: **semanage fcontext -a -t public\_content\_rw\_t "/myftp/pub(/.\*)?"**
- Relevant SELinux booleans are:
  - **allow\_ftp\_anon\_write** (allow ftp users to write to FS objects labeled with **public\_content\_rw\_t**)
  - **allow\_ftpd\_full\_access** (skips SELinux checks, DAC still in place)
  - **allow\_ftpd\_use\_nfs** (allow accessing files that are mounted through NFS and labeled **nfs\_t**)
  - **ftp\_home\_dir** (allow R/W access to authenticated users to their own home directories)

# vsftpd practice

- This example shows how to create a directory that is writeable through ftp. This directory can also be served as read-only web content at the same time. (E.g. this is a way for users to upload their web pages to a server.)

## Shell command line

```
# setsebool -P ftp_home_dir on
# mkdir -p /myftp/pub
# chown jane:root /myftp/pub
# chmod 775 /myftp/pub
# semanage fcontext -a -t public_content_t /myftp
# semanage fcontext -a -t public_content_rw_t
"/myftp/pub(/.*)?"
# restorecon -R /myftp/
# ls -dZ /myftp/pub
drwxrwxr-x. jane root unconfined_u:object_r:
                               default_t:s0 /myftp/pub/
# setsebool -P allow_ftpd_anon_write on
# systemctl start vsftpd
# systemctl enable vsftpd
```

# vsftpd practice

- Now test the FTP server from a client computer.

Shell command line

```
~]$ ftp ftp.test.net
Connected to ftp.test.net (10.10.0.1).
220 (vsFTPd 2.1.0)
Name (localhost:username):jane
331 Please specify the password.
Password: thisisasecret
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd myftp
250 Directory successfully changed.
ftp> put ftpupload
local: ftpupload remote: ftpupload
227 Entering Passive Mode (127,0,0,1,241,41).
150 Ok to send data.
226 File receive OK.
ftp> quit
221 Goodbye.
```

# Recommended reading

Lecture 9

Chapter 5

## Relevant chapters in RH:

## Read by next lecture:

- RH124 CH10 (Logs)
- RH134 CH4 (Scheduling tasks)

- RH 124 CH9 (SSH)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 9, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# SSH

Lecture 10

Chapter 1

# SSH

- SSH's (Secure Shell) first version was developed by Tatu Ylönen (Helsinki University) in 1995.
- Current version is v2, an open-source implementation called OpenSSH is available in practically all \*nix systems.
- SSH implements public key cryptography for authenticating both server machine and client user.
- SSH applies the Diffie-Hellman key exchange algorithm or one of its variants for securing the channel.
- SSH is considered very secure, it is mathematically proven to be very safe, no one has been able to exploit it so far.
- SSH offers additional features through the established secure channel, like TCP tunneling, file transfer, remote process execution and redirection of standard streams.

# Cryptography

- Symmetric cryptography algorithms use the the same key for encrypting and decryption of the message.
- Symmetric algorithms work on blocks of data (that are of the same size as the key).
- One of the most commonly known symmetric algorithms is AES.
- AES is based on the work of Joan Daemen and Vincent Rijmen (Belgium), called the Rijndael encryption.
- Symmetric algorithms work relatively fast, most modern CPUs (even low-powered embedded ones) have built-in hardware instruction(s) for calculating AES for a block with a given key.
- An Intel x86 Core i7-5820K CPU at 3.3 Ghz can calculate AES for more than 4GB of data per each second.

# Cryptography

- Asymmetric cryptography incorporates a pair of keys. Data encrypted with one of the keys can only be decrypted with the other and vice versa.
- They are based on irreversible mathematical problems like the discrete logarithm problem or the difficulty to break a number down to prime factors.
- Asymmetric algorithms run slower by magnitudes.
- Usually one part of the key pair is proclaimed public and get distributed through public databases (notary, trust chain), the other one is kept private. When encrypting a message with the private key, it will be readable to all, but this will prove the origin of the message (digital signature). When encrypting data with the public key, everyone can send a message to an addressee, that only the addressed individual can read (encryption).
- Widely known: RSA algorithm (Rivest, Shamir, Adleman)

# SSH session encryption

- When establishing an SSH session the following steps are involved.
  - Protocol version, algorithm exchange, agreement.
  - Server public key is sent to the client for comparison to clients records. (**known\_hosts**)
  - Client ensures the server has the private key corresponding to the public key received.
  - Diffie-Hellman key exchange (→ See next slide)
  - Symmetric session key is obtained by both parties, the connection is secure from now on.
  - EITHER Authentication with the users password (if enabled), the password can be sent safely by now.
  - OR Authentication with the users private key (if it is available) → Detailed in next slide

# SSH session encryption

- Authenticating the user with a key pair involves the following steps.
  - The server has a catalog of all allowed user's public keys. (**authorized\_keys**)
  - Server generates a random number sequence, encrypts it with the users public key, and sends it.
  - The client (if it has the private key) decrypts the number and combines it with the session key, then calculates its MD5 hash.
  - Client sends the result back to server.
  - Server calculates the same hash.
  - If response matches servers calculation, the client user proved that he/she is in possession of the private key, therefore is considered authenticated.

# Diffie-Hellman example

- Bob and Alice agree on a prime modulus (m) and a generator number (g). For e.g.  $g=3$   $m=17$  Both Alice (**A**) and Bob (**B**) selects a random **private** number. For e.g.  $A=15$  and  $B=13$ .
- Alice calculates her public number  $P_A = g^A \bmod m$ , and sends this to Bob. (e.g.  $3^{15} \bmod 17 = 6$ ) Bob does the same with his public number  $P_B = g^B \bmod m$ , and send it to Alice. (e.g.  $3^{13} \bmod 17 = 12$ )
- Alice takes Bob's public number and calculates  $S = P_B^A \bmod m$  (e.g.  $12^{15} \bmod 17 = 10$ ) Bob does the same with Alice's public number.  $S = P_A^B \bmod m$  (e.g.  $6^{13} \bmod 17 = 10$ ).
- S is their shared secret. Note that they did the same calculation:  $(3^{15})^{13} \bmod 17 = (3^{13})^{15} \bmod 17$ , but an eavesdropper can't follow this calculation without knowing numbers **A** and **B** ( $13$  and  $15$  in this example).

# SSH session encryption

- Note that the DH private and public numbers are unrelated to SSH private and public keys, rather they are randomly generated on the start of each session.
- Note that session encryption is done with a symmetrical algorithm, that provides much faster speeds. Asymmetrical algorithms are only involved on the start of a session, where the identity of the other party is established.
- Note that Diffie-Hellman by itself does not protect against Man-in-the-Middle attacks. That's why a server computer also has a keypair, not just client users.
- SSH clients ask for confirmation when encountering a server for the first time. Later it keeps track of the public keys of servers, and refuses to connect if a server key changes.

# SSH session encryption

- MITM attack can only succeed at the first encounter, when client does not yet know the public key of the server. Therefore it is encouraged to propagate the public key of a server via others means (like directly copying from a flash drive instead of over the network.)

## ssh first encounter example

```
local-host$ ssh remote-host
The authenticity of host 'remote-host(192.168.1.1)'
can't be established. RSA key fingerprint is
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
Are you sure you want to continue connecting
(yes/no)? yes
Warning: Permanently added 'somehost' (RSA) to the
list of known hosts.
user@somehost's password: [not shown]
remote-host$
```

# SSH host key change

## ssh host key change

```
@@@@@WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!@  
@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING  
NASTY! Someone could be eavesdropping on you right  
now! (man-in-the-middle attack)  
It is also possible that the RSA host key has just  
been changed. The fingerprint for the RSA key sent  
by the remote host is  
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.  
Please contact your system administrator.  
Add correct host key in /home/user/.ssh/known_hosts  
to get rid of this.  
Offending key in /home/user/.ssh/known_hosts:1  
Password authentication is disabled to avoid man-  
in-the-middle attacks.  
Agent forwarding is disabled to avoid man-in-the-  
middle attacks.  
X11 forwarding is disabled to avoid man-in-the-  
middle attacks.
```

# SSH client

- To start a remote interactive session with ssh simply issue **ssh -p 1822 user@hostname-or-ipaddress**. Where the -p switch defines a non standard port, if omitted, the standard port of 22/tcp will be used. Username can also be omitted, in this case the same username will be tried as the user running the local shell.
- Use **ssh user@host *command*** to execute a command on the remote computer. SSH will print the output of the remote command locally. When the remote execution ends, SSH will terminate the session.
- You can even use pipes and redirects to remote processes this way, ssh will handle the transfer.

shell command line

```
local-host$ dd if=disk1.img | ssh remote-host 'dd of=/datastore/disk1.img'
```

# SSH client

- Basically SSH provides an encrypted channel of standard streams for remote processes (like redirects or pseudo terminals for remote shell sessions). SSH has some additional features, like X11 forwarding, TCP port forwarding, and secure copying files remotely (scp).
- You can invoke remote GUI programs while making them draw to the local screen with the following example: **ssh -X bob@officecomputer word-processor**
- SSH port forwarding works by technically "mirroring" a TCP port on one side of the tunnel to the other side and connecting it to another destination.
- There are 3 types of SSH port forwards: local, remote and dynamic. Dynamic provides a full-featured SOCKS proxy, the other two will be explained in detail later.
- SSH itself can't provide any means for UDP forwarding.

# SSH local forward

- This can be used to bypass local firewalls, or to provide access to server side computers. Take the following example: **ssh -L 8080:redhat.com:80 bob@my\_server**  
In this example, (after connecting the SSH client to the daemon on remote my\_server machine), pointing a browser on the client computer to localhost:8080 will download a page from redhat.com's port 80. The redhat.com server will perceive as the page was downloaded from my\_server and not your local machine.
- The last part of the -L argument is interpreted from the SSH servers perspective. For e.g. **12345:localhost:5432** would connect local 12345 port to the servers 5432 (postgresql) port.
- Likewise you can access restricted computers on the server side like this: **-L 17380:192.168.1.173:80**

# SSH remote forward

- Remote forwards work similar but the opposite direction. An example of `ssh -R 9000:localhost:80 bob@serverXY` would do the following: Anyone opening port 9000 on the server would actually connect to your local computers web server (as long as your client is connected to the SSH session).
- Both local and remote forward requires server side agreement. Local forwarding is allowed by default, but remote forwarding is disabled in the default sshd config. Add **GatewayPorts yes** to your server configuration to allow it.
- In case of -R the second part of the argument is interpreted from the ssh client's perspective, localhost in this example means the ssh client that originated the SSH session.

# scp, sshfs

- One additional element of the OpenSSH suite is the scp utility, which allows for copying files to/from remote computers via an SSH encrypted channel. Either source or destination can be a remote file (or dir), but not both. Use this command like you would use the regular cp command. E.g.: **scp thesis.txt bob@univ:/home/bob**
- By executing scp and various filesystem operation commands remotely via ssh, you can implement a file server. For convenience there is wrapper for this functionality (openssh-sftp-server), the involved protocol is called SFTP (not FTPS!).
- You can even mount a remote directory and access it locally in the regular way. Behind the scenes the work is done by the fuser kernel module, the remote sftp-server and the local ssh client. Use sshfs like you would use mount. E.g. **sshfs bob@server:/work ~/bobs-work-dir**

# Client side ssh settings

- The settings for the SSH client are stored in the **/etc/ssh/ssh\_config** file. The directives can be overridden by each user's **~/.ssh/config** file.
- The key pairs of users are stored under **~/.ssh/id\_xxx** and **~/.ssh/id\_xxx.pub** for each user, where **xxx** refers to the algorithm, e.g. **id\_rsa**, **id\_dsa**, **id\_ecdsa**, ...
- The list of already encountered servers and their public key's fingerprints are stored in **~/.ssh/known\_hosts**.

**/etc/ssh/ssh\_config**

```
Host dev
  HostName dev.example.com
  Port 22000
  User fooey
  IdentityFile ~/.ssh/dev-example.key
Host tunnel
  HostName database.example.com
  LocalForward 9906 127.0.0.1:3306
```

# User keypairs

- You can use `ssh-keygen -t dsa -b 2048 -C "home comp"` to create a key pair. The location to save the files to will be prompted for. You can accept the defaults (see previous slide). You can use the `-t` parameter to specify the algorithm, e.g `rsa`, `dsa`, `ecdsa`. Use `-b` for setting the key size (default for `rsa` is 1024 bits), the optional `-C` argument gives a comment to the key.
- The private part of the key should always be kept secret and should assume a mode of **600**.
- You will also be prompted for a passphrase. It can protect your private key, without the passphrase it will be useless to anyone who steals the key file.
- In case you use the key for password-less login, you still have to enter the key passphrase (can be inconvenient). Use an ssh-agent to keep keys in memory, this way you only have to type the pass once, until turning your PC off .

# SSH server

- The server side configuration is stored in `/etc/ssh/sshd_config`. The server's host key is stored in `/etc/ssh/ssh_host_xxx_key` and `ssh_host_xxx_key.pub`, where xxx denotes the algorithm (e.g. rsa, dsa, ecdsa, ...) Use **ssh-keygen** (the same way as with user keys) to create a host key. The only difference is the file's location. You can also use the `-f` switch to ssh-keygen to specify the output filename directly from command line.
- Public keys of users are stored in `~/.ssh/authorized_keys` on the server side for each user. For e.g. by adding Alice's public key to both `/root/.ssh/authorized_keys` and `/home/bob/.ssh/authorized_keys` file, Alice will be able to log in to this server both as the root user and as Bob. The file mode has to be **600**.
- Run **ssh-copy-id user@server** on the client side to add a key to a users `authorized_keys` on the server. (password needed)

# SSH server

## /etc/ssh/sshd\_config

ListenAddress	0.0.0.0
Port	22
HostKey	/etc/ssh/ssh_host_rsa_key
HostKey	/etc/ssh/ssh_host_dsa_key
SyslogFacility	AUTHPRIV
AuthorizedKeysFile	.ssh/authorized_keys
StrictModes	yes
RhostsAuthentication	no
RSAAuthentication	no
PasswordAuthentication	yes
PubkeyAuthentication	yes
PermitEmptyPasswords	no
PermitRootLogin	without-password
AllowTcpForwarding	yes
GatewayPorts	no
X11Forwarding	yes
Subsystem sftp	/usr/libexec/openssh/sftp-server

# SSH server

- Complete practice **RH124 CH 9.2.** (ssh,w)
- Complete practice **RH124 CH 9.4.** (ssh-copy-id)
- Complete practice **RH124 CH 9.6.** (PermitRootLogin)
- Complete practice **RH124 CH 9.7.** (ssh-copy-id, PermitRootLogin)

# DHCP server

Lecture 10

Chapter 2

# dhcpd

- To add DHCP provider functionality to your server, install the package with **yum install dhcp-server**. This will set you up with a functioning example configuration file under **/etc/dhcp/dhcpd.conf**.
- The service can be controlled with **systemctl**, the unit name is **dhcpd.service**.
- The configuration file has options for different scopes. For. e.g there are global options (see next slide), subnet or host specific options.
- It is also possible to specify directives with conditions or matches. (see next slide: if and match constructs)
- Remember that `systemd-networkd` can also provide basic dhcp functions. Do not enable more than one dhcp service, usually it will lead to confusion.

# dhcpd.conf

/etc/dhcp/dhcpd.conf

```
#server options
authoritative;
allow bootp;
#you can define custom dhcp variables
option space gpxe;
option gpxe.keep-san code 8 = unsigned integer 8;

#global options, can be overriden in sub-sections
subnet mask 255.255.255.0
option domain-name-servers 8.8.8.8

#subnet specific options, dhcpcd will find the
#relevant interface from the IP address and mask.
subnet 192.168.5.0 netmask 255.255.255.0 {
    range 192.168.5.100 192.168.5.200;
    option broadcast-address 192.168.5.255;
    option routers 192.168.5.1;
    next-server 192.168.5.10;
    filename /pxelinux.0;
}
```

# dhcpd.conf

/etc/dhcp/dhcpd.conf continued

```
host DBServer {
    hardware ethernet c0:c4:c0:45:19:ac;
    fixed-address 192.168.5.20; }
#matching rules and classes
class "denied" {
    match if substring (hardware,1,3) = 00:54:36; }
pool {
    deny members of "denied";
    range 192.168.100.100 192.168.100.200; }
#conditional options, e.g. answer depends on what
#dhcp client asks for the value: pxe gets different
#response than actual operating system.
host test_1 {
    hardware ethernet 08:00:27:1c:b1:e6;
    if exists user-class and option user-class =
        "gPXE" {
        option root-path "aoe:e0.1";
    } else {
        filename "gpxe.kpxe"; }
}
```

# Apache

Lecture 10

Chapter 3

# HTTP

- Both CERN (in 1992) and NCSA (in 1993) defined the HTTP specification. The original goal was to access text files easily over network. Later on the protocol was extended with non-text transfers.(Mosaic - the first browser)
- After 1994 NCSA abandoned the project, enthusiasts continued to develop it. Enhancements were shared in the form of patches→ A Patchy → Apache
- This later became a formal organization (Apache Server Project), with an own open-source licensing scheme (Apache license).
- Look up <https://httpd.apache.org/> for a comprehensive documentation.
- HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT

# URL

- The client requests a remote resource from the server. If access is granted, the resource will be sent over network. Otherwise an error message is sent instead.
- The remote resource is described with a URL (Universal resource location) Based on RFC 1738, a URL has the following components:  
`protocol://user:password@host:port/path-inside-server`
- Any component of the URL can be omitted, except for host name. All others parts have sane defaults. For e.g. pointing a web browser to redhat.com will assume the following defaults: protocol is http since we are using a web browser, user and password may not be required for public content, port is 80 by default for http protocol, and path will have a server defined default, like for e.g. /index.html or /index.php.

# Example transfer

- The client sends the following to the server's port 80/tcp:

HTTP GET request

```
GET /hello.html HTTP/1.1
Host: webapp0.example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; [...])
Accept: text/html,application/xhtml+xml,application/xml,application/javascript,application/json
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Cache-Control: max-age 0
```

- The server responds with a header and contents:

HTTP GET reply

```
HTTP/1.1 200 OK
Date: Wed, 19 Sep 2017 08:34:34 GMT
Server: Apache 2.4.20 (Red Hat) OpenSSL/1.0.1e [...]
Content-Length: 12
Keep-Alive: timeout=5, max=82
Content-Type: text/plain; charset=UTF-8
Hello World!
```

# HTTP

- Reply status codes are categorized into series:
  - series 100: informational messages
  - series 200: client request successful
  - series 300: redirection happened
  - series 400: client error
  - series 500: server side error
- There are multiple HTTP servers available for different OSs. The biggest market share belongs to Apache, which is available on all common OSs including non-Unix ones, like Microsoft Windows.
- A competing web server is nginx, which is also full-featured and excels in serving static content more efficiently than Apache.

# Apache

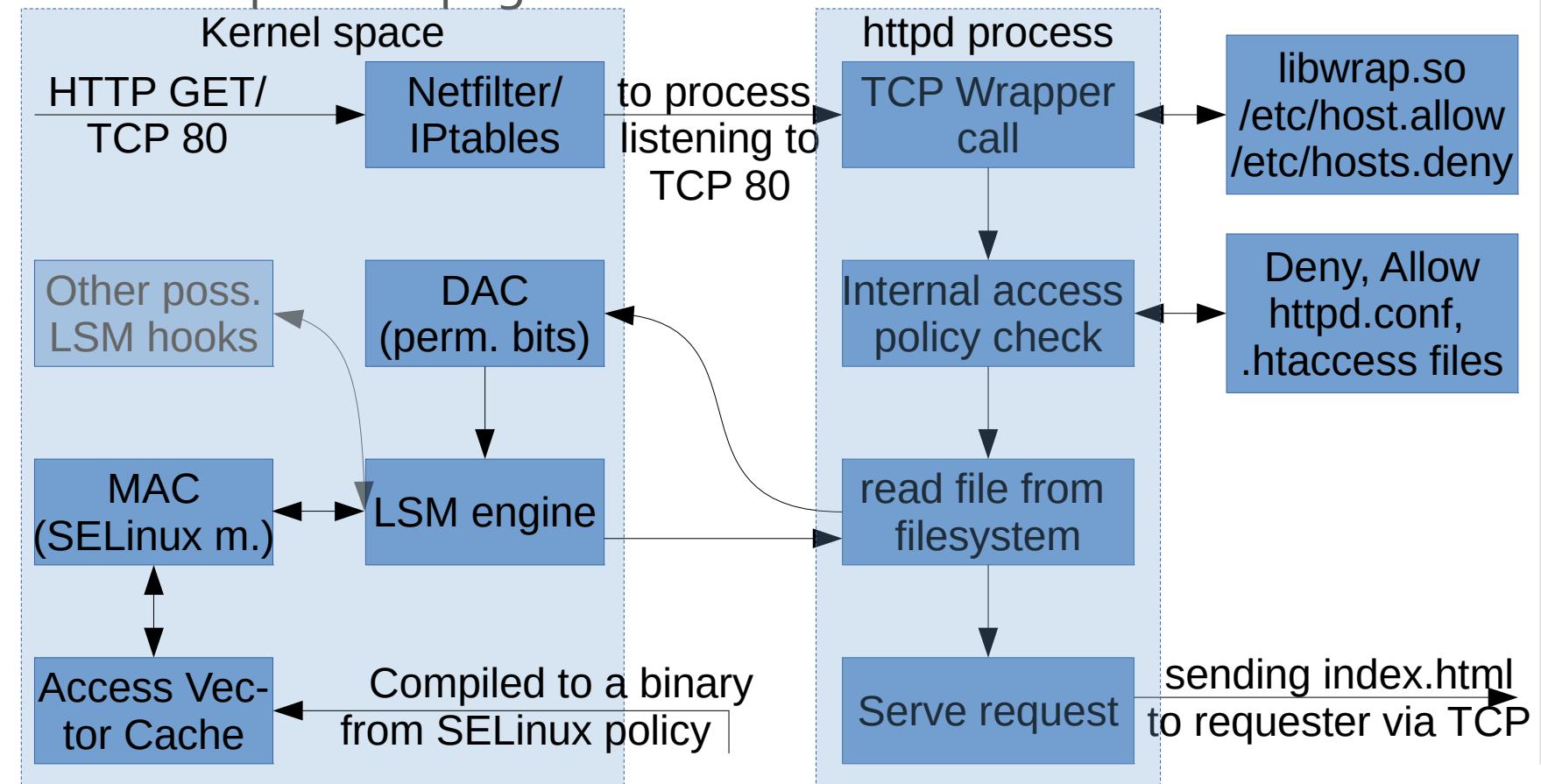
- Apache accesses several directories on the file system:
  - The apache binary, apache modules and shared libraries in `/usr/bin`, `/usr/lib`
  - Configuration files in `/etc/httpd/*`
  - Web content in `/var/www/html` (by default)
  - Server logs in `/var/log/httpd/*`
  - Executable scripts (cgi-bin, python, php, ...) in the web application directories for e.g. `/usr/share/owncloud/*`
  - Documentation in `/usr/share/doc/apache` (The documentation is available in html format served as a web site)
  - Possible more custom files/directories

# Apache

- Most important security considerations:
  - The apache process should run with a dedicated technical account (with no shell to log in). Not the 'root' user, neither 'nobody' user.
  - The apache user and relevant group (e.g. webmasters or apache) should only own the files, that are involved in web serving.
  - Set an ACL or regular Unix permissions for the files involved, e.g.:  
**setfacl -R -m d:g:webmasters:rwx /var/www/html**
  - Set SELinux labels properly: e.g: **semanage fcontext -a -t httpd\_sys\_content\_t '/location/of/web/content(/.\*)?'**
  - Open up firewall port(s) for the web server: e.g.:  
**firewall-cmd --permanent --add-service=http**

# General flow of security measures for network resources

- In Linux several checks are performed before a resource (e.g. file) is served to a requester. Consider this diagram depicting the flow of events when a user wants to access a simple web page over the network.



# Apache modules and multi- processing

- Apache has a modular build. Each set of functionalities is organized into binary modules, that can be loaded separately. Plugging more modules in Apache will add new features and new possible configuration directives to apache's knowledge.
- The more modules you use, the greater the memory impact.
- Apache has to load a few modules even for basic operation. E.g. you have to have a multi- processing module loaded even to start a basic server, like either one of the **prefork**, **worker** or **event** modules.
- Prefork starts a number of processes, each listening and serving one RQ at a time. Worker launches several threads inside processes: one listener and multiple work threads. Event is similar to worker, it tries to fix the 'keep alive' problem of open and stale connections from clients.

# Apache configuration

- Apache has a default configuration file, in RH systems it's located in **/etc/httpd/conf/httpd.conf**. The default config is quite complex and includes several other .conf files with a C-style include directive. This kind of configuration, when modified, requires the server to be restarted or reloaded.
- Apache also has a way of reading location sensitive configuration directives directly from the served content folder with .htaccess files. In this case security settings can be moved together with the web content folder. Also note, that changing these doesn't require the server to be reloaded. The downside is a performance impact, since access files have to be re-read and evaluated on every relevant request.
- The AllowOverride directive in the regular config decides whether the .htaccess files should be considered or not.

# Apache configuration

- The configuration file(s) contain some global directives, and some location or context specific (container) directives, like **<Directory /var/www> ... </Directory>**
- Directives inside a container only apply to the relevant subset of files, contexts. The more specific a container is, the higher priority it has, meaning that in case of conflicting settings, directives for a sub-directory override the directives for it's parent directory.
- The **Directory** container section applies to a whole filesystem directory, the **Files** container applies to a single file. The **Location** container applies to a match in the URL. e.g.: depending on symlinks, rewrites, aliases the two following examples are not necessarily pointing to the same object: `server.com/webapp/store_index` and the directory `/webapp/store_index` under the Document-Root directory of `server.com` are not always the same.

# Apache minimalist configuration

/etc/httpd/conf/httpd.conf

```
Listen 80
User apache
Group apache
ServerName example123.com
DocumentRoot "/var/www/html"
ServerRoot /etc/httpd
ErrorLog /var/log/httpd/error.log
PidFile run/httpd.pid
LoadModule mpm_event_module
                      modules/mod_mpm_event.so
LoadModule authz_core_module
                      modules/mod_authz_core.so
LoadModule unixd_module modules/mod_unixd.so
LoadModule systemd_module modules/mod_systemd.so
<Directory />
    Require all denied
</Directory>
<Directory /var/www/html>
    Require all granted
</Directory>
```

# Apache basic configuration examples

- Some additional important basic configuration directives:

```
/etc/httpd/conf/httpd.conf [excerpt]
```

```
LoadModule authz_host_module
          modules/mod_authz_host.so
LoadModule dir_module
          modules/mod_dir.so
LoadModule access_compat_module
          modules/mod_access_compat.so
#access_compat allows for old style apache 2.2
#directives like: Allow from all

<Directory /var/www/html>
  FollowSymLinks
  AllowOverride none
  Require ip 192.168.56.0/24 #mod_authz_host rqd.
  Require not ip 192.168.56.98 #mod_authz_host
</Directory>
AddDefaultCharset UTF-8
IncludeOptional conf.d/*.conf
DirectoryIndex index.html #dir_module is needed
```

# Apache configuration: password auth

- Several modules are required for this feature, one for the auth capability, one for reading users from file, groups from a file, and so on. Format examples below: first line for apache groups, second line for apache users file
- groupname: space separated list of user names
- username:\$apr1\$6O6rAQFK\$g2M/z214SKdN7KQ.

/etc/httpd/conf/httpd.conf [excerpt]

```
LoadModule authz_user_module
           modules/mod_authz_user.so
LoadModule authz_groupfile_module
           modules/mod_authz_groupfile.so
LoadModule auth_basic_module
           modules/mod_auth_basic.so
LoadModule authn_file_module
           modules/mod_authn_file.so
LoadModule authn_core_module
           modules/mod_authn_core.so
```

# Apache configuration: password auth

- Use **htpasswd** to create entries in the users file. E.g. **htpasswd -c /etc/httpd/apachusers jane**, use -c only the first time, it stands for creating the file, otherwise it will just append new users. Also note **require valid-user**.
- You can define several requirements for an access, use **RequireAny** for setting up an OR logical operation, use **RequireAll** to create an AND logical operation.

/etc/httpd/conf/httpd.conf [excerpt]

```
<Directory /some/path>
  AuthType basic
  AuthName "Give password for finance charts"
  AuthGroupFile /etc/apache2/apache_groups
  AuthUserFile /etc/apache2/apache_users
  <RequireAny>
    require group finance
    require user boss
  </RequireAny>
</Directory>
```

# Apache configuration virtual hosts

```
/etc/httpd/conf/httpd.conf [excerpt]
</VirtualHost *:80>
  ServerName www.butterflies.com
  DocumentRoot /var/www/html/butter
  ErrorLog /var/www/logs/butter_error_log
  ServerAdmin webmaster@butterflies.com
</VirtualHost>
</VirtualHost *:80>
  ServerName www.examples.com
  DocumentRoot /var/www/html/examples
  ErrorLog /var/www/logs/examples_error_log
</VirtualHost>
```

- You can run several isolated web sites/web applications on the same server. You can define virtual hosts based on IP address and/or port number. If you only have one IP address, and want to run several pages on default port 80, use name based virtual hosts. In this case the URL typed by the client will be evaluated (remember the GET header) and matched against the ServerName directive.

# Apache configuration SSL

# Apache configuration dynamic web content, scripts

- To allow the execution of cgi scripts: first label them as **httpd\_sys\_script\_exec\_t**, grant access to the cgi-bin directory with a **<Directory>** block and add the following to your apache configuration:  
**ScriptAlias /cgi-bin/ "/var/www-cgi-bin"**
- To serve dynamic PHP content: first install **mod\_php**, and add the following to your config:  
**<FilesMatch \.php\$>**  
**SetHandler application/x-httpd-php**  
**</FilesMatch>**  
**DirectoryIndex index.php**
- To serve dynamic python content: install **mod\_wsgi** first, label your files with **httpd\_sys\_content\_t** and add the following to Apache's configuration:  
**WSGIScriptAlias /mywebapp/ /srv/mywebapp/some.py**

# Apache getting help

- Install the `httpd-manual` package with **yum install httpd-manual** and point your browser to `http://localhost/manual` to view the documentation.
- This is also available at **[httpd.apache.org/docs/current](http://httpd.apache.org/docs/current)**
- The documentation details each configuration directive with examples, explaining what module has to be loaded for the feature/directive to become available.
- Complete practice **RH254 CH 10.2.** (`httpd-manual`)
- Complete practice **RH254 CH 10.4.** (virtual hosts)
- Complete practice **RH254 CH 10.6.** (TLS/SSL)
- Complete practice **RH254 CH 10.8.** (php web appl.)
- Complete practice **RH254 CH 10.9.** (python with TLS)

# Recommended reading

Lecture 10

Chapter 4

Relevant  
chapters in RH:

Read by next  
lecture:

- RH 124 CH9 (SSH)
- RH254 CH10 (Apache/httpd)

- RH134 CH12 (Samba)
- RH254 CH8 (File storage servers/Samba)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 10, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# MariaDB

Lecture 11

Chapter 1

# History of MySQL

- Ancestors: TcX (Sweden) web development company created it's own database engine (1995) for internal use. It was based on mSQL and Unireg. Later released under the GPL license.
- TcX was later renamed to MySQL AB. Sun Microsystems bought MySQL. Oracle bought Sun Microsystems.
- Oracle already had a professional enterprise level database product: OracleDB. Development of MySQL somewhat died down.
- MySQL was already open-source and was available under the GPL license almost from the beginning. → Enthusiasts created a fork and continued development → MariaDB was born. MariaDB 5.5 is a compatible drop in replacement for MySQL 5.5.
- RH/Fedora still keeps both as a package in repositories.

# Relational databases

- A relational database allows for storing data in persistent, structured way. It is organized as follows:
- A database consists of one or more tables, each describing an entity.
- Tables contain multiple records (lines).
- Each column stores a specific type of attribute for the record.
- Relations are usually created by linking entities with their IDs. For e.g. there is a table to store the attributes for all the products. Each product has an ID. In a table that stores invoices, invoiced items are not stored redundantly, instead they contain reference entries like: "twelve pieces of product 19". But a printed invoice will have lines with product name, color, price, etc... (Look up the JOIN sql operation).

# Challenges

- Concurrent access (a simplified example): two transactions happen at the same time. Let's say a shared bank account with a balance of 100\$ is being withdrawn from. The wife withdraws 20\$ in one branch (process 1 reads balance, subtracts 20\$ and calculates \$80 as new balance, tries to write this balance back.) The husband withdraws 10\$ at the same time in an other branch (process 2 reads the balance as 100\$, subtracts 10\$ and calculates a new balance of 90\$, writes this back to DB).
- For one there will be a race condition, either 90\$ or 80\$ gets written last, and only the last write will stick, but we cannot know beforehand which one. Even if we put the race condition aside, the fact is: neither 90\$ nor 80\$ is a correct result for this operation. Solution: locking.
- Problem: What/how to lock? R/W lock? How is speed affected? (low speed → queue buildup → more concurrent access?)

# Challenges

- Complex instructions and conditions, e.g.: A customer finalizes a lengthy order. All items have to be removed from stock, to prevent others from buying the same merchandise multiple times. Also at the same time customer has to be charged for the amount of purchase. Customer pays with a credit card, but for some reason money does not get through. How to cancel? Now both customer credit and stock inventory is inconsistent.
- Solution: Transactions. Problems: processing speed, memory need.
- Transactions enable for a complex set of instructions to either be fully carried out or the ability to return to the consistent state before the transaction started.
- This involves snapshotting and locking ranges of memory. (can be CPU/memory intensive, can affect caching, ...)

# ACID test

- Requirements for transactions:
  - Atomicity: Atomic, undivideable set of instructions. No partial execution allowed.
  - Consistency: The database will transition from one consistent state into another consistent state.
  - Isolation: The effects of a transaction are invisible to others until it is fully carried out.
  - Durability: Once a transaction finishes, it will be permanent, no partial rollback or leftover inconsistency can remain.

# DB engines

- As you can see different application requirements can be accommodated with different level of complexity (CPU and memory needs), different set of features.
- MySQL resolves this by offering several database engines. You can select different DB engines per table.
  - XtraDB (InnoDB): supports transactions, per record locking, no space limit
  - Aria (MyISAM): smaller footprint, quite fast, no transactions, per table locking, easily portable (copy between systems)
  - Memory: very fast, volatile, does not write to disk
  - Archive: only supports SELECT and INSERT, optimal for storing log entries

# DB engines

- Common storage engines (continued)
  - CSV: reads, stores and appends directly to "comma separated values" format text files.
  - MyRocks: Compressed storage, with reduced write amplification, optimal for flash drives, embedded systems (e.g. data loggers).
  - TokuDB: transactional engine for large workloads, that don't fit in memory, with good compression.
  - Spider: provides sharding, partitioned to multiple servers.
  - Ndb: network clustering engine (not active in MariaDB)
  - Galera (not an engine): synchronous multi-master clustering, preferred cluster method for MariaDB.

# Architecture, features

- MySQL/MariaDB serves the requests with a threading model, multi-core CPUs can be benefited from.
- Query results are cached, repeating SELECT queries are served from cache, if possible.
- MariaDB improved a lot and added some missing features recently: stored procedures, triggers, views.
- MariaDB still struggles with subqueries, and it is not based on object oriented concepts (ORDBMS)
- MySQL is the preferred DB back-end for the LAMP stack. For long it has been the fastest of open-source DBMS (now it is PostgreSQL). Many web apps rely on it.
- For an enterprise quality open-source Object-Relational DBMS look up PostgreSQL (it is also part of RH repos). Many business applications, like ERP systems rely on it.

# Architecture, features

- It is common to run the DB server on the same host, that runs the web application code (for e.g. a web store written in php). This has the advantage of almost no latency in communication with the DBMS. It is recommended to use Unix local domain sockets instead of the localhost loop-back adapter in this case.
- In same cases, for performance or partitioning reasons, the DB server runs on a dedicated host. Cooperating applications send SQL queries and receive result through a TCP network connection with the DB server.
- Just as TCP listen ports can instantiate numerous sessions (virtual end-to-end pipes, each connects the server process with a different client), Unix domain sockets can do the same with almost no overhead. Of course the restriction is that communicating processes have to run on the same host (kernel) to use this feature.

# Unix domain sockets

- From the programmers point of view, using local domain sockets is almost the same, as using TCP sockets.
- Unix local domain sockets are filesystem objects.
- From the system engineers point of view, using socket files requires a lot less from the system. A smaller latency and larger throughput can be achieved, because there is no TCP involvement: no packet ordering, no routing, no congestion/flow control, no retransmissions, no checksum calculation, no encapsulation happens.
- The loopback network adapter intentionally hides the fact that sender and receiver are on the same host, it is just a TCP/IP channel, like any other. Unix domain sockets allow for the sending thread to write the stream or datagrams directly into the receiving socket buffer.

# MariaDB server and client

- The **mariadb-client** package group contains utilities for connecting to a MySQL/MariaDB server from the command line. It allows for the user to type and send SQL queries to the server, and displays the results as text tables.
- The client package also contains some libraries and connectors for userspace applications that rely on submitting MySQL queries.
- The **mariadb** package group contains the server part (that can store databases in files receive and respond to SQL queries through domain sockets or TCP sockets). It also contains a test database and provides some management/backup tools like the mysqldump command.
- A well known GUI client is MySQL Workbench.

# my.cnf

- Both server and client store their configuration in **/etc/my.cnf**, but settings are separated to different sections.

## /etc/my.cnf [excerpt]

```
[client]
port          = 3306
socket        = /var/run/mysqld/mysql.sock
[mysql]
prompt        = '\u@\\h [\d]> '
default_character_set = utf8
[mysqld]
bind-address  = 0.0.0.0 #(single entry!)
port          = 3306
socket        = /var/run/mysqld/mysql.sock
datadir       = /var/lib/mysql
tmpdir        = /tmp
default_storage_engine = InnoDB
character_set_server = utf8
max_connections = 505
max_user_connections = 500
```

# MariaDB installation

- Run **yum groupinstall mariadb** to install relevant packages. The package manager will create a technical user for the daemon (mysql, see /var/lib/mysql folder)
- Edit **/etc/my.cnf** with the text editor of your choosing.
- Run **systemctl start mariadb** and **systemctl enable mariadb**.
- Run **firewall-cmd --add-service=mysql --permanent**; **firewall-cmd --reload** to open port 3306/tcp. (default) Remember that mysql also has TCP Wrapper bindings.
- Run **mysql\_secure\_installation** to start the security setup wizard and set up a root password. Note that though they have similar names, the **mysql root** user is not to be confused with the **system root** user. Run **mysql -u root -p** to test your login.
- Complete practice **RH254 CH 9.2.** (MariaDB install)

# MySQL client

- Use `mysql -u username -p -h hostname` to log in to a given server with a given user. The `-p` switch stand for password prompt. Otherwise mysql will not ask for a password, instead looks for it in environment variables (`$MYSQL_PWD`) or `my.cnf`. If `-h` switch is omitted, the local domain socket file will be used instead of TCP.
- Once logged in try the following commands to obtain general information:
  - `SHOW databases;` `USE dbname;` `SHOW tables;`
  - `DESCRIBE tablename;`
  - `SHOW open_tables FROM dbname;`
  - `SELECT current_user();`
  - `SHOW privileges;` `SHOW grants for user@host;`

# MySQL client

- Let's try some data definition commands:
  - `CREATE database first_db CHARACTER SET=utf-8;`
  - `USE first_db;`
  - `CREATE TABLE employees (name CHAR(20), job_descr CHAR(30), salary INTEGER);`
  - `ALTER TABLE employees ADD COLUMN entry_date DATE;`
  - `ALTER TABLE employees MODIFY salary FLOAT;`
  - `CREATE TABLE purchase_orders (quantity UNSIGNED INTEGER, date DATE) ENGINE=INNODB;`
  - `DROP TABLE purchase_orders;`

# MySQL client

- Let's try some data manipulation commands:
  - **INSERT INTO employees (name, salary) VALUES ('John Smith', 250);**
  - **INSERT INTO employees (name, salary, entry\_date, job\_descr) VALUES ('Foo Bar', 350, STR\_TO\_DATE ('23-11-2017', '%d-%m-%Y'), 'junior engineer');**
  - **SELECT \* FROM employees;**
  - **SELECT name, salary FROM employees WHERE salary > 300;**
  - **UPDATE employees SET salary=400 WHERE name='John Smith';**
  - **DELETE FROM employees WHERE salary > 380;**

# MySQL users and privileges

- Data control commands (GRANT, REVOKE) will be discussed with MySQL users.
- Rules are stored as table entries, and are evaluated as ACL's, with the general rule being of the lowest priority, and the specific rule overriding the general rules.
- There are 5 tables involved, providing increased granularity of permission control in the following order: **user, db, host, tables\_priv, columns\_priv**.
- The user table basically defines the user accounts and corresponding passwords. The db table defines what user can do what operations on the databases. The host table refines control depending where the user logged in from. (Users can have different privileges depending on for e.g.: whether they logged in from the office LAN or from a public site.

# MySQL users and privileges

- The tables\_priv and columns\_priv built-in tables allow for fine grained control over which user can have what rights on each table, or even each column. For e.g. a foreman can assign new work shifts to employees, but can not modify the salary column.
- You have two fundamental ways for managing users and access rights:
  - either use the CREATE USER, GRANT, REVOKE commands
  - or use basic data manipulation commands (like UPDATE, DELETE, INSERT) on mysql's built in user, db, host and other relevant tables. If choosing this solution you have to manually trigger the **FLUSH PRIVILEGES;** command to apply changes made to the permission settings.

# MySQL users and privileges

- To create a user use one of the following schemes:
  - **USE mysql; INSERT INTO user (user, host, password) VALUES ('bob', 'localhost', password('secret')); FLUSH PRIVILEGES;**
  - **CREATE USER bob@localhost IDENTIFIED BY 'secret';**
- The latter command will create a user, set passwords, also set host related records and automatically flushes privilege cache.
- The newly created user can log in, but can not actually execute any commands.
- Host can contain the following example expressions:  
`'localhost', '192.168.20.40', '192.168.20.%', '%'`

# MySQL users and privileges

- To add permissions use the following scheme: GRANT *list\_of\_COMMANDS* ON *db\_name.table\_name* TO *user@host*; For e.g. **GRANT SELECT, INSERT ON first\_db.employees TO bob@'localhost';**
- Practice: Look at the db table after previous command!
- You can use the \* wildcard on *table\_name* and/or *db\_name*.
- As for the privilege list, you can use the ALL PRIVILEGES keyword. The USAGE keyword is a synonym for no privileges, it only allows for a user to log in, nothing else.
- Use the REVOKE command with the same logic to take privileges away from a user.  
E.g.: **REVOKE ALTER, DROP ON \*.\* FROM bob@'%'**
- Complete practice **RH254 CH 9.6.** (MariaDB users)

# MySQL logs

- MySQL has its own 4 separate logs. (unrelated to the system logger daemon)

/etc/my.cnf [excerpt]

```
[mysqld]
log_error = <hostname>_error.log
log_warnings = 2

slow_query_log_file = <hostname>_slow.log
slow_query_log = 1
long_query_time = 0.5
min_examined_row_limit = 100

general_log_file = <hostname>_general.log
general_log = 0

server_id = 42
log_bin = <hostname>_binlog
binlog_cache_size = 1M
max_binlog_size = 128M
expire_logs_days = 5
```

# MySQL logs

- MySQL has it's own 4 separate logs. (unrelated to the system logger daemon)
- Error log and general log is self explanatory.
- Slow query log holds the events when the processing of SQL queries took too long. This is useful for development purposes or to filter out users, SQL clients that use a substantial amount of system resources.
- The binary log contains all the data alterations. A DB can be rolled back to former state with special tools, when the binary log is available. But the main purpose of this type of log is to facilitate the replication of a server. (High availability services, backup servers.)
- The binary log can't be read directly, use **mysqlbinlog** **/var/log/mysql/mysql-bin.000001** to view contents.

# Backup and restore

- When creating a backup of a DBMS you have to consider the role of memory caches. The state stored on disk files usually does not equal to a complete, consistent copy of all the table states.
- Therefore simply creating a copy of the directory that holds DBMS raw data (/var/lib/mysql) usually leaves you with an unusable backup.
- You can either make steps to ensure the consistent state of disk files and than create a **physical backup** of the backing-store. Or you can use the DBMS system to create dump of the current state to a separate backup file (called **logical backup**).
- Logical backups are clear text files containing SQL instructions, that if executed in order, will form a database matching the one that was backed up.

# Backup and restore

- Logical backups are highly portable, and may even be compatible with other DBMS system, since they only contain standard SQL text instructions. Logical backups can be created while server is running, no interruption needed.
- Physical backups have a smaller footprint, and can be created faster than logical backups. Physical backups require the server to be brought offline or at least in a locked state to ensure consistency. Physical backups can also contain the logs and configuration files, while logical backups can't. Physical backups can only be ported to a narrow subset of versions of the same DBMS.
- Complete practice **RH254 CH 9.8.** (restore backup)
- Complete practice **RH254 CH 9.9.** (configure MariaDB)

# Logical backups

- Use the `mysqldump -p -u root db_name table1 table2 > /backups/backup_db_2017-11-27.sql` command to create a text sql dump of tables table1 and table2 inside db\_name to a file called backup\_db\_2017-11-27.sql. The -u and -p switches work similar to the mysql command.
- If you want to create a backup of all tables, omit the table names. If you would like to backup everything, omit the db\_name argument and add `--all-databases` switch. You also specify a list of databases when using the `--databases db1,db2, ...` switch.
- Some additional useful switches: `--no-data` (only saves structure, but not the records), `--lock-all-tables` (prevent writing while dump happens), `--no-create-info` (only save records, do not create structure, assumes compatible existing structure), `--add-drop-table`, `--add-drop-database` (on restore, existing table or db will be dropped before restore operation)

# Physical backups

- Create a copy of the mysql datadir (/var/lib/mysql by default) with cp, rsync, tar or similar (Refer to Lecture 12 for more) for. e.g.:  
`cp -a /var/lib/mysql /backups/mysql/2017-11-27`
- Or you can also choose to create an lvm snapshot of the volume that holds the data directory. Note that creating a snapshot takes the fraction of a second, later on you can use tar and compress or copy the directory from the snapshot, while DBMS resumed normal operation.
- To bring the disk files to a consistent state **always do this** before a physical backup: either use **systemctl stop mariadb** or use mysql client to flush and lock the tables the following way: **FLUSH TABLES WITH READ LOCK**; After snapshot: **UNLOCK TABLES**; or **systemctl start mariadb**. (See RH254 CH 9.7 for more.)

# Reset mysql root password

- In case you forget the password of the **mysql root** user, you can perform the following emergency steps to reset the password.
- Stop mysql /mariadb (**systemctl stop mariadb**).
- Open up a dedicated terminal and execute:  
**mysqld\_safe --skip-grant-tables** (this will keep running)
- **mysql -u root → use mysql; UPDATE user SET password=password('new') WHERE user='root'; FLUSH PRIVILEGES;**
- Now terminate the mysqld\_safe process in the other terminal with **ctrl+c**.
- Start the daemon as usual: **systemctl start mariadb**
- In case a regular user's password gets lost, just use the root user to modify the user table with a new password.

# Samba

Lecture 11

Chapter 2

# Samba

- Microsoft Windows File and Printer Sharing services implement the Server Message Block (SMB) protocol. Common Internet Filesystem (CIFS) is a dialect of SMB, in practice these names are often used interchangeably.
- The name Samba (on some linux systems smd or smbd) corresponds to the Linux implementation of the SMB protocol (and a set of relevant tools).
- Samba integrates perfectly with Windows networks. It can be both used as a client to services (shares) provided by a Microsoft product, both the other way around: a Windows computer can connect to a Samba provided share. In fact Samba can even work as Primary Domain Controller (PDC) for Windows workstations in an mixed-OS enterprise environment.

# Samba security

- When creating a share, samba can follow one of the following 4 security schemes:
  - User level security: User authenticates to the server with a username/password upon session initiation. Once authenticated, all allowed shares can be accessed by the user.
  - Share level security: User authenticates against each shared directory separately.
  - Domain security: All authentication request are passed through the PDC. The samba server requires a machine account to be added to the Server Manager of the DC.
  - ADS security: Samba (vers. 4) can join as a native member to Microsoft Active Directory Services.

# Defining SMB shares

- Install the **samba** package. Use your favorite editor on **/etc/smb.conf** file.

`/etc/smb.conf [excerpt]`

```
[global]
    server string = Samba Server version %v
    netbios name = server-example
    hosts allow=172.25.0.0/24
    workgroup = Workgroup
    encrypt passwords = true
    guest account = nobody
    invalid users = root
    max protocol = SMB2
    passdb backend = tdbsam
    security = user

    vfs objects = acl_xattr
    map acl inherit = yes
    store dos attributes = yes

    allow insecure wide links = yes
```

# Defining SMB shares

## /etc/smb.conf [excerpt]

### [homes]

```
comment = %u's Home Directory
browsable = no
read only = no
```

### [media]

```
comment = Share for TV connected media player
path = /shares/media
valid users = bob,alice, @media
read only = no
guest ok = no
create mask = 0660
directory mask = 0770
browseable = yes
```

### [printers]

```
path = /var/spool/samba
print ok = yes
printable = yes
browseable = no
```

# Enabling SMB shares

- After setting up the shares in `smb.conf`, make sure the shared directory has a proper SELinux context:  
**`semanage fcontext -a -t samba_share_t '/shares(/.*)?'`**  
**`restorecon -Rv /shares`**
- Sharing home directories through Samba requires the following SE Boolean set: **`setsebool -P samba_enable_home_dirs on`**
- Change the owner and mode of the shares directory, remember to set the SETGID bit for the directory: **`chgrp -R media /shares/media; chmod -R 2775 /shares/media`**
- Create corresponding Unix users. For e.g.: **`useradd -s /sbin/nologin -G media john`**
- And finally create a samba password (LM, NTLM hashes) for the user: **`smbpasswd -a john`**
- Enable the relevant firewall services: **`firewall-cmd --add-service=samba --permanent; firewall-cmd --reload`**

# Enabling SMB shares

- This will open the following 4 ports: `137/udp` (`nmbd`), `138/udp` (`nmbd`), `139/tcp` (`nmbd`), `445/tcp` (`smbd`)
- The `nmbd` ports are required for NetBIOS name resolution to work. If you want to refer to Windows computer by their NetBIOS name, other than opening the ports you also have to start the `nmb` service: **`systemctl start nmb`**.
- Finally you may start the Samba service: **`systemctl start smb`**. You may also set the services for autostart: **`systemctl enable nmb smb`**
- Remember that changing `smb.conf` requires **`systemctl restart smb`** for changes to take effect. You can use **`testparm`** to check `smb.conf` and **`pdbedit`** to manage Samba users and passwords.
- Complete practice **RH254 CH 8.6.** (provide SMB services)

# Samba client

- To identify a remote share: **smbclient -L //server\_name**
- This will list all the shares on the server that are set to `browsable=yes`. You can still mount the others, if you know the name of the share exactly.
- To perform a mount, consider the following example:  
**mount.cifs -o username=bob,password=secret //server\_name/share\_name /mnt/samba.**
- Typing a password like this may not be secure, since it will be preserved in your Bash history. You can omit the password option, `mount.cifs` will ask you for a password from `stdin`, if required.
- You can also use the **-o guest** option, if the share allows anonymous access.
- Note the following options for local permission mapping:  
**-o file\_mode=0777,dir\_mode=0777,uid=bob,gid=bob**

# Samba client credentials

- The `-o credentials=~/smb.cred` option to `mount.cifs` specifies a text file (e.g. `smb.cred` in your home directory) to read credentials from. This way you can authenticate without typing the password or leaving it in the shell's history. The format of a credentials file is the following:

```
~/smb.cred
```

```
username=bob
password=secret
domain=my_company
```

- The credentials file should only be readable by you:  
`chmod 600 ~/smb.cred`
- With a credentials file you can even do fstab mounts:  

```
/etc/fstab [excerpt]
```

  
`//serverX/media /mnt/samba cifs  
credentials=/secure/credentials.file 0 0`
- Complete practice **RH134 CH 12.2.** (mount SMB shares)

# Recommended reading

Lecture 11

Chapter 3

## Relevant chapters in RH:

## Read by next lecture:

- RH134 CH12 (Samba)
- RH254 CH8 (File storage servers/Samba)

- RH134 CH11 (NFS, autofs)
- RH254 CH8 (File storage servers/NFS)
- RH124 CH12 (Archiving and copying files)
- RH124 CH15 (Virtualized systems)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 11, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# NFS

Lecture 12

Chapter 1

# NFS

- Network File System is an open standard and it is the native protocol for sharing directories over a network for Unix like operating systems.
- Directory sub-trees made available to network users (shared directories) are called **exports** in NFS terminology. A server exports directories, a client mounts them.
- NFS is an internet standard, Windows 7 and newer versions can connect to NFS exports without any third-party software.
- NFS versions 2 and 3 support both TCP and UDP as a transport layer, NFSv4 only supports TCP. NFSv4 also introduced advanced security and authentication features, efficient sparse file support and more.
- By default RH 7 uses version 4 of the NFS protocol but can fall back to previous version if needed.

# Shared filesystems, locking

- Multiple processes can open the same file. The kernel has knowledge about all such operations (they happen via system calls) on a local filesystem and keeps track of what file/range is being read and/or written.
- Network shares on the other hand make it possible for multiple processes of different kernels (hosts) to open the same file. In this case there is no single kernel on the network that would be aware of all the open files involved in an export. This is a universal problem of networked filesystems (not just NFS).
- NFSv2 and 3 doesn't support file locking at all by itself, but has an ancillary protocol available called NLM (Network Lock Manager).
- File locking is supported as part of the NFSv4 protocol, all the locking information is available at the server and is propagated through the network to clients and back.

# NFS security

- Version 2 and 3 of NFS authorizes clients based on their IP address, there is no authentication involved. If your computer can assume a specific IP address (for e.g. you have privileges to set your own IP address), then you have access to the relevant share.
- File and directory permissions are enforced on exports, but this is only effective if the users and groups are managed centrally on all the computers involved.
- If you can assume a UID or GID on your local computer (for e.g. you have root access to your own workstation, then you can locally su or sudo to anyone), you have the same permissions on the export as the given UID and GID would have on the server side filesystem.
- To mitigate the risks arising from that, the **squashing** mechanism was introduced.

# NFS security

- Squashing means, that the a users permissions are remapped to the nfsnobody user (UID=65534) or another anonymous user (specified with the **anongid=** and **anonuid=** export options). The root user perms are intentionally squashed on the server side by default (use **no\_root\_squash** to defer), the **all\_squash** export option squashes all user permissions to anonymous.
- NFSv4 introduced additional security methods:
  - none: no access check at all, with all user mapped to the anonymous user (nfsnobody).
  - sys: default, the same as NFS v2 and v3 security, standard Unix permissions and ACL checks work.
  - krb5: like sys (standard Unix permissions apply), but client must also prove identity with Kerberos.
  - krb5i: like krb5 with all request getting signed (no tampering with requests possible).
  - krb5p: in addition to Kerberos authentication all traffic is encrypted, has a performance impact.

# Exporting shares

- The **exports** file format: It contains one line for each share, and at least two fields for each line: First column is the shared directory path, second column contains the list of allowed clients, and the export options in effect for the client, in braces. Further clients(options) pairs can optionally be specified in additional columns.
- The client specifier can be a single IPv4 or IPv6 address, a subnet, a resolvable FQDN or a domain name with wildcards.

/etc/exports (example)

```
/share/exported_directory 192.168.10.0/24(rw)
/another/share 10.0.0.1(rw) 10.0.0.2(ro,all_squash)
/some/export workstation[0-20].example.com(sync)
```

- Some important export options: sync, async, rw, ro, root\_squash, all\_squash, anongid, anonuid, no\_acl, fsid, sec=krb5p, sec=sys, sec=none, ...

# Exporting shares

- To use NFS, install the `nfs-utils` package (both on server and client computers): **yum install nfs-utils**.
- Edit the `/etc/exports` and/or the `.exports` files in the `/etc/exports.d` drop-in directory of the server.
- Open up port 2049/tcp on the server's firewall:  
**firewall-cmd --add-service=nfs --permanent; firewall-cmd --reload;**
- Finally start and enable the service with `systemd`:  
**systemctl start nfs-server; systemctl enable nfs-server**
- Whenever modifying the exports file(s) either restart or notify the nfs-server about the changes with **systemctl restart nfs-server** or **exportfs -r**.
- Complete practice **RH254 CH 8.2. (NFS export)**

# Mounting shares

- Once identified the correct share, you can simply mount it with e.g. **mount.nfs -o sync,nfsvers=3 serverN:/share /mnt/nfs\_share**. Just like any other mount command, this also has options, a source, and a target mount point. Use the **nfsvers** option if you would like to defer from the default NFS version of 4. Use the **sec** mount option to set security (eg. **sec=krb5i**)
- You can create permanent nfs mounts in **/etc/fstab** with the method learned before. (what, where, type, opts)
- To identify an exported resource use the following command on the client for NFSv3 and v2: **showmount -e server\_name\_or\_ip**. To identify available exports on NFSv4: mount the root directory of the server: **mount.nfs serverX:/ /mnt\_point** then **ls /mnt\_point**. Do this as root. You won't have access to Kerberos shares, but they will also be listed.
- Complete practice

# Secure exports

- Kerberos authentication requires at least the presence of a proper /etc/krb5.keytab file. You usually get this file from the admin responsible for security, when joining a Kerberos Realm. Creating such a realm is not covered in this class, but a ready made keytab will be provided for the purpose of the practices.
- Kerberos security options require the **nfs-secure-server** systemd unit to also be enabled and started besides normal **nfs-server** unit. On the client side the **nfs-secure** unit is the only one needed for this.
- Have an export based on the following example:  
**/secured\_export \*.example.com(sec=kerb5p,rw)**
- Mount it on the client with: **mount.nfs -o sec=krb5p serverx:/secured\_export /mnt/secure\_nfs\_mount**
- Complete practices **RH134 CH 11.2.** (NFS mount) and **RH254 CH 8.4.** (Secure NFS)

# NFS and SELinux

- By default the `nfs_export_all_ro` and `nfs_export_all_rw` booleans are enabled, allowing access to almost any file over an NFS export. To tighten this, turn these booleans off and label your shared files with either `nfs_t` or `public_content_t`. To allow write access use `nfs_t` or `public_content_rw_t` in conjunction with the `nfs_anon_write` boolean.
- SELinux context is not propagated to clients by default. All files have the **same** (by default: `nfs_t`) label on the client's mount point. The default can be overridden with the `-o context="something"` mount option: still all files will have the same context on the client.
- NFSv4.2 supports SELinux context propagation. To enable this, change the relevant line in `/etc/sysconfig/nfs` to `RPCNFSDARGS="-V 4.2"` and mount with `-o v4.2` on the client.

# Autofs

Lecture 12

Chapter 2

# Autofs

- Autofs service can be instructed to watch directories and mount filesystems on them as the directory is accessed. Autofs defaults to handling nfs mounts, but can be used with many other types of filesystems.
- Autofs provides a convenient way of letting users without root privilege to mount a restricted set of filesystems. It is also commonly used to define wildcard mounts (e.g. each user can have their respective home directories auto-mounted over nfs without creating a constant mount for all the users on all the hosts of the whole network.)
- Autofs is able to dismount and clean up after itself when a mount point is not used anymore, alleviating both server and client side resource load.

# Autofs

- Autofs has two mapping modes: direct and indirect.
- In both cases autofs mounts directories under /tmp/mnt based on mapping rules described in configuration files. In both cases, when a path under autofs's supervision gets accessed (e.g. ls) it will immediately get mounted.
- Direct mounts can be used for absolute paths. Each mount point entry will have a separate mttab entry, (it can grow quite large). Changes to configuration require autofs to be reloaded.
- In case of indirect maps, autofs creates a virtual file-system on a given directory, and works more like a symlink server, that manipulates links to the /tmp/mnt directory dynamically, based on current needs. This scheme is used for relative paths under the indirect directory and only the indirect directory itself gets into the mttab.

# Autofs master configuration

- Autofs has a master configuration, stored in the **/etc/auto.master** file and **/etc/.autofs.master.d** drop-in directory. The master configuration contains references to other configuration files that contain the direct or indirect mappings for specific directories. A reference to a direct map configuration file always has a path of **/-** while an indirect reference contains the relevant directory path.

`/etc/auto.master [excerpt]`

```
/-          /etc/config1.direct
/automnt_shares /etc/config2.indirect
```

- Config file names are freely selectable, but usually something informative is used.

# Autofs configuration

- Indirect mapping files have relative paths. This example with the former auto.master file means the following: Automount provider.com:/shared/somethg to the local mountpoint of /automnt\_shares/something whenever the local path gets accessed.

```
/etc/config2.indirect
```

```
something -fstype=nfs,rw provide.com:/share/somethg
```

- Direct mapping file's format is very similar, but it contains an absolute path.

```
/etc/config1.direct
```

```
/local/something -sync,rw provide.com:/share/someth
```

- The middle column contains the regular mount options in a comma separated list, without spaces. The fstype=nfs option can be omitted, since it is the default.

# Autofs configuration

- Any type of filesystem can be automounted with a correct fstype option. An SMB automount example looks like this: **samba\_mount -fstype=cifs,credentials=/etc/smb.cred ::/smb\_server/sharename**
- The indirect mount can be used very efficiently with wildcards. For e.g. when an NFS server exports the user home directories, they can be automounted on any workstation in the network with the following simple line.

```
/etc/homedirs.indirect
* -rw, sync server:/shares/home_dirs/&
```

- If accessing any directory in the indirect root, the \* (asterisk) will match against it, and the & (ampersand) will substitute whatever the first match was. E.g.: **ls /automnt/bob** command will automatically mount **server:/shares/home\_dirs/bob** to **/automnt/bob** directory.

# Autofs setup

- Issue **yum install autofs** to install necessary packages.
- Edit the master autofs file or a file in the master drop-in directory, and name a direct or indirect mapping file.
- Create the named direct/indirect mapping file and set at least one line in it.
- Execute **systemctl enable autofs** and **systemctl start autofs** to start and enable the service.
- Whenever modifying direct mounts, remember to restart the service. When modifying an indirect mount no restart is necessary, changes take effect automatically the next time the directory is mounted.
- Complete practice **RH134 CH 11.4. (automount NFS)**

# Backup, archiving

Lecture 12

Chapter 3

# Copying files between systems

- In this chapter a few well-known ways will be shown on how to copy files between systems. Remember that making a copy of files/database dumps inside the same filesystem is not considered a backup, since in case of a hard disk failure all copies are lost at once.
- One common way for backing up whole directory structures is to collect all the FS object under a path to **tarball file**, compress it and copy it to another disk or another host. This can be done manually or by the means of a script that gets scheduled for. e.g. by cron.
- The other well known way of creating backups is via the **rsync** utility, which can differentially replicate a whole directory tree to a local or even directly to a remote destination. Also consider cron for scheduling this.
- Use the regular **cp** command with the **-a** switch (archive) for preserving perms, owner, times, ... (root may be reqd.)

# Creating tar backups

- Let's suppose you already have a database dumped to an `sql` file or you already have a directory ready for archival.
- First you should run the `tar` command, which will collect all files and their basic metadata (e.g. timestamps) to one single file. This was originally intended for tape backup drives, which are not made to retrieve individual files quickly anyway.
- The `tar` command can compress collected data on the fly with many different algorithms, directly producing a `.tar.gz`, `.tar.bzz`, `.tar.xz`, etc... file.
- Then you should copy the file with the regular `cp` command to another local disk/network share or using the `scp` command as discussed in Lecture 10.

# Tar switches

- Tar is a BSD style command, does not need a - (dash) for arguments. The most common switches are:
  - t for testing a tar archive
  - c for creating a tar archive
  - x for extracting the files and dirs in the archive
  - v for verbose output
  - f for telling the archive filename (this switch has an argument: the filename)
  - z to also compress data while processing (gzip alg.)
  - j to compress with bzip2 algorithm
  - J for compressing with the xz algorithm
  - p for preserving permissions (while extracting)

# Tar switches

- Remember that switches can be combined and reordered, but some switches have arguments, like the f switch. Therefore f is usually left last.
- Tar example for creating an archive of a specific folder in gz format: **tar czvf /backups/jane.tar.gz /home/jane**  
To extract it to a directory: **cd /extracted; tar xzvf /backups/jane.tar.gz**
- To transfer files securely to another host: **scp /backups/\*.tar.gz backupserver:/data/workstation1** or **scp -r /backups bcksrv:/datastore** for transferring an entire directory.
- You may also use the **sftp** command. This will execute remote system commands through an SSH tunnel, but usage is the same as of the regular **ftp** CLI client.
- Complete practices **RH 124 CH 12.2.** (tar create, extract) and **CH 12.4.** (scp)

# Rsync principles

- Rsync can be used synchronize complete Filesystem subtrees to another location. It can operate in two modes: EITHER source and destination are both local OR one of them is a remote computer.
- Remote transfer can be done via the rsync protocol or over SSH. Former requires that the server runs an rsyncd process. Latter only requires that both sides have the rsync command installed, it will be automatically called on the remote side via an SSH session.
- Rsync can efficiently update large data sets over narrow-band networks, because it only send the differences, that occurred since the last synchronization. Of course it takes about the regular amount of time to transfer a backupset for the first time. Besides the difference transfer capabilities rsync can also implement regular compression.

# Rsync principles

- Rsync's simplified operating method is the following:
  - First it will compare source and dest. file metadata (size, timestamps, ...) one-by-one for each element of the directory tree. In case of exact match, nothing further will be done, they are considered to be in sync, otherwise they are marked in memory for further examination.
  - Rsync will transfer a marked file as a whole file if both src. and dst. are on local filesystems (like cp). If one of them is remote, than the file will get sliced to logical blocks. Rsync will calculate a hash for each of the blocks on both sides and only transfer the hash over the network, for comparison.
  - In case the current blocks hash does not match, it will transfer the block only, not the whole file. This is repeated for all the blocks in all the marked files.

# Rsync usage

- The general structure of the rsync command is the following:  
**rsync -switches source\_dir user@remotehost:/dest\_dir**  
where one of source or destination can be remote host  
OR both can be a local directory or mount point.
- The most important switches are:
  - -v for verbose
  - -a for archive, the same as -rlptgoD
  - -r for recurse into directories
  - -l for copy symlinks as symlink (do not dereference)
  - -p to preserve permissions
  - -t to preserve modification timestamp
  - -g to preserve group
  - -o to preserve owner

# Rsync usage

- The most important switches are (continued):
  - -D to preserve device and special files
  - -H to preserve hard links
  - -A to preserve ACLs
  - -X to preserve extended attributes (SELinux labels)
  - -z to compress transferred data
  - -u for update-only (skip files newer on the dest.)
  - -h for human readable units
  - -P for show progress during transfer (pv)
  - -m for delete empty directories on destination
  - -E to preserve executability
  - --inplace for updating file blocks in place (no tmp)
  - --sparse for treating blocks of zeroes efficiently

# Rsync usage

- The most important switches are (continued):
  - `--compress-level=N` where `N=[1..9]`
  - `--numeric-ids` for skipping `passwd` file `id<→name` resolution (just copy the numeric value)
  - `--del` for deleting files on destination, that are not present at the source anymore
  - `--log-file=filename` for record output in a log file
  - `--stats` to show statistics at the end of transfer
  - `-e 'ssh -p 12022'` for telling the underlying remote execution method, for e.g. can hold arguments for keyfile, ssh port, or non-standard ssh command
  - `--filter=filterlist.file` for file filtering rules to be read from given filename
- Complete practices **RH 124 CH 12.6.** (rsync) and **12.7.**

# Virtualization basics

Lecture 12

Chapter 4

# KVM

- KVM stands for Kernel-based Virtual Machine. The Linux Kernel itself can act as a full-virtualization hypervisor for VMs. It competes with commercial products like Vmware.
- KVM was Originally developed by Avi Kivity at Qumranet. This company was bought by RedHat in 2008.
- KVM is a component of the mainline kernel. The kvm-intel or kvm-amd kernel module (and a proper CPU) is required for the kernel to act as a fully hardware assisted hypervisor. Use **modprobe** command to load a module.
- Note that KVM can be categorized as both Type I and Type II hypervisor. This is a matter of active debate. KVM turns the kernel to a bare-metal hypervisor (Type I) while the OS functionalities still work at full extent. KVM works in close cooperation with the qemu-system-x86 emulator. All Vms are processes from the OSs perspective (trait of Type II hypervisors).

# qemu

- Qemu is an open-source, efficient emulator family. It can be compiled to run on a wide range of systems to emulate another range of systems. For e.g. `qemu-system-x86`, `qemu-system-arm`, `qemu-system-mips64el`, etc.
- In case of hardware assisted full virtualization (for e.g. running `qemu-system-x86` on a modern x86 CPU) qemu doesn't need to trap, binary translate or emulate the x86 instructions. Instead its role is mainly to emulate hardware devices, such as virtual or para-virtual storage controllers, network cards, display adapters.
- The paravirtualized hardware devices emulated by qemu are natively supported by Linux and also supported by Windows guest when installing the certified, digitally signed, mature RedHat `virtio-win` drivers.

- The qemu-KVM technology, with proper hardware, also allows for assigning physical hardware devices (like PCI cards) directly to VMs via IOMMU. This can offer uniform hardware (storage, network, etc... → easy backup and migration) for VMs with for e.g. physical GPUs attached for ultimate parallel processing performance. The hybrid physical-virtual systems backed by qemu-KVM are extremely efficient. If properly configured, they can provide up to 97-98% of the native performance.
- The libvirt library and API is an open source framework for managing virtualized systems. The qemu-KVM backend is only one of libvirt's supported technologies. (with connectors it can manage VirtualBox hosts, Xen hosts, the OpenStack platform and more...)
- Libvirtd is the resident daemon process that manages hypervisors and receives connections from user frontend.

# virsh

- Just like many other daemons, libvirtd has a CLI client as well as a GUI client. There are also several Web front-ends working with the libvirt API.
- Libvirtd stores virtual resource settings (for individual resources or complete virtual domains) in xml files under /etc/libvirtd. Look up the format at: <https://libvirt.org/formatdomain.html>
- The standard command line interface to libvirtd is **virsh**. Virsh can be used to manage domains (VMs) defined in libvirt. Take the following command examples: **virsh list** will list all active domains. The **--all** switch lists every domain, including the ones in shutdown state.
- To manage the running state of domains use **virsh start/shutdown/reboot/reset/destroy VMname** to start, gracefully shut down, gracefully reboot, force reset or force power-off a VM respectively.

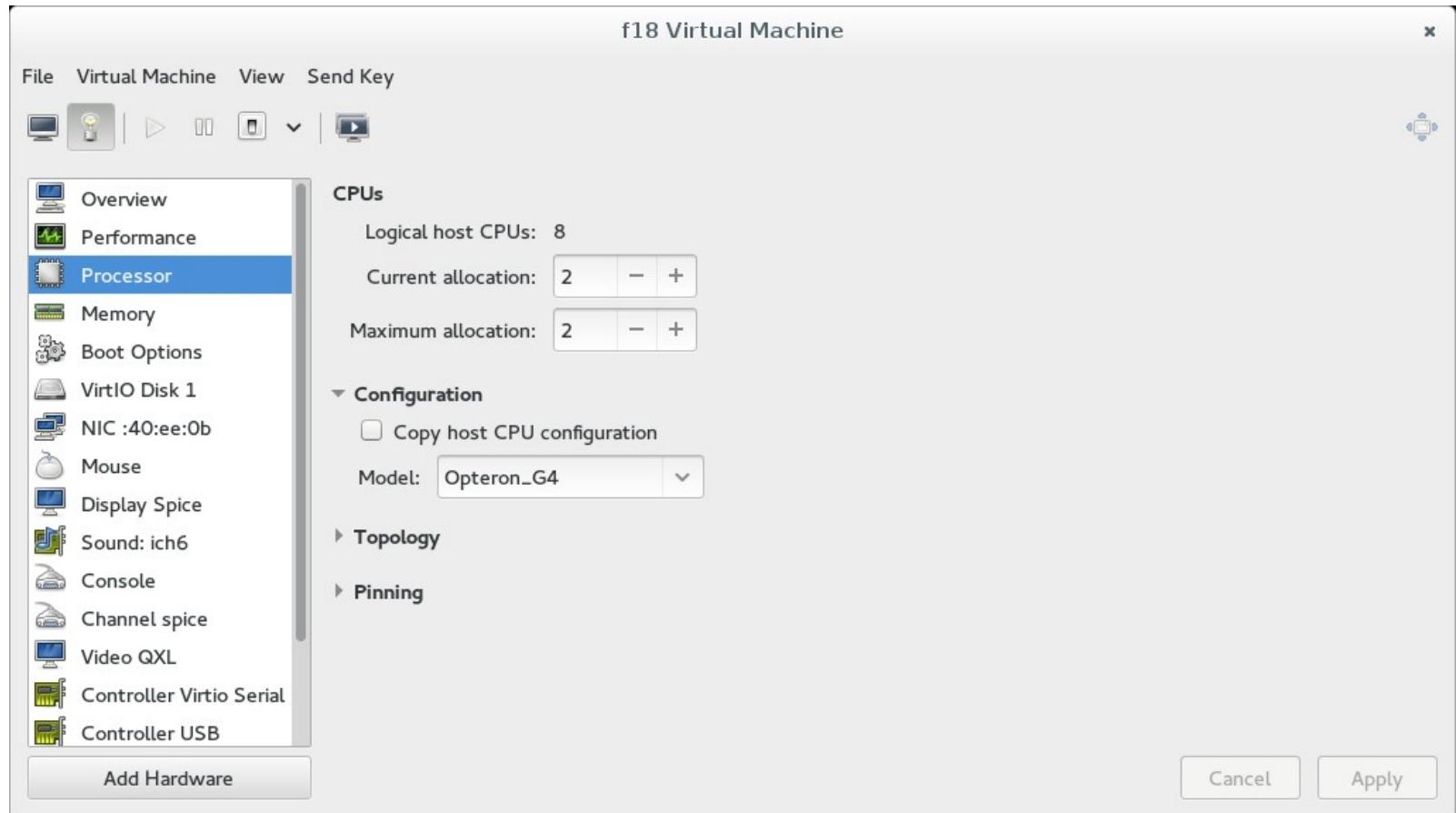
# virsh

- Power management command examples:  
**virsh dompmsuspend/dompmwakeup *VMname***
- To edit an existing VM's xml configuration with the default editor use **virsh edit *VMname***.
- Use **virsh create *vm.xml*** to cretae a new VM from a domain xml file and start it. Use **virsh define *vm.xml*** to create/update the xml desciption of a VM without starting it. Use **virsh undefine *VMname*** to remove a VM.
- Use **virsh console *VMname*** to connect current terminal to the serial console of the guest.
- Issue **virsh nodeinfo *VMname*** to get a domain's details.
- Look up **man virsh** for the snapshot-create-as, snapshot-delete, snapshot-info, snapshot-list, snapshot-revert sub-commands and further details.

# virt-manager

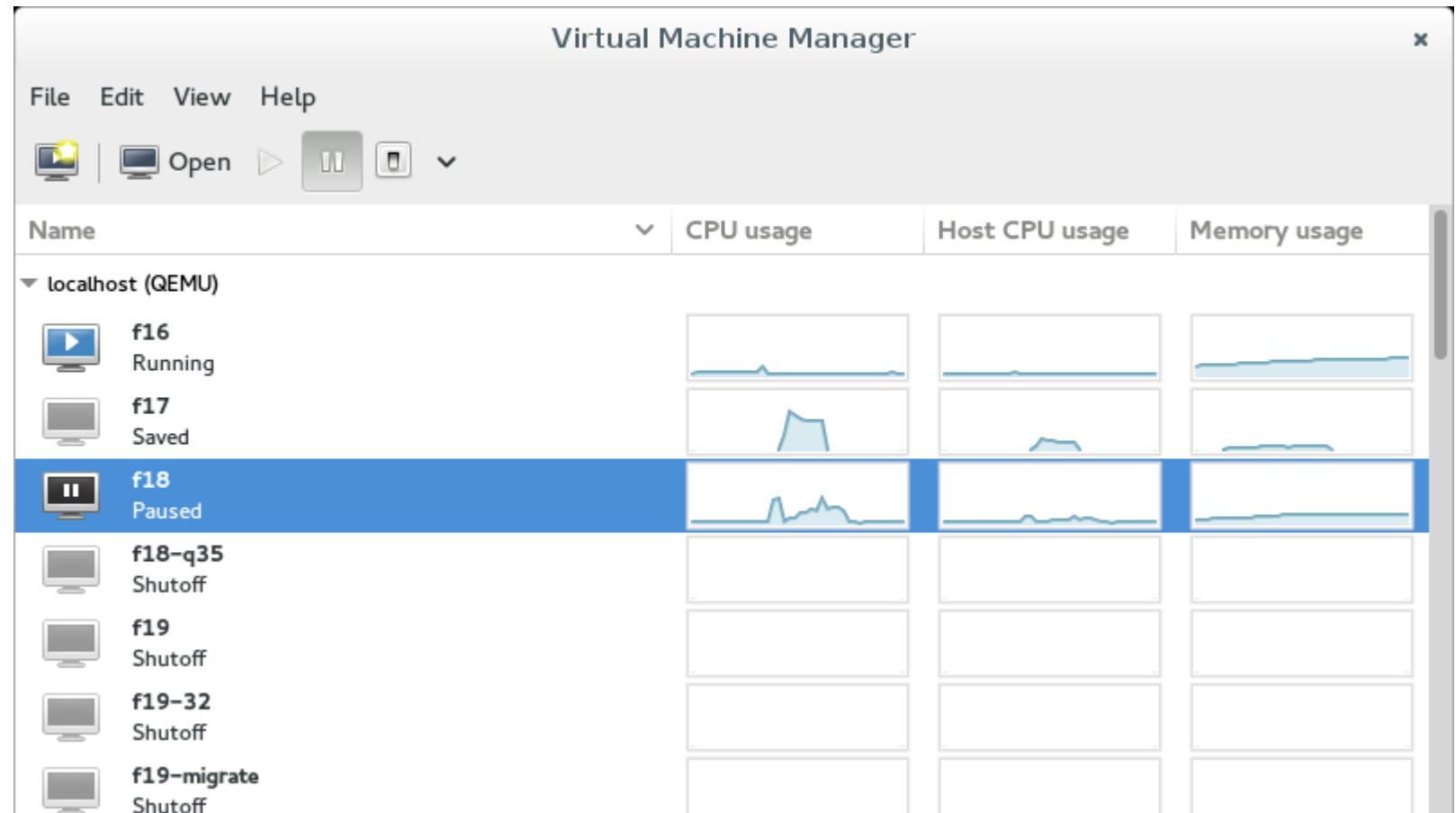
- The virt-manager package installs a GUI client to libvirtd. The command name is also virt-manager.
- It has a very clear user interface and provides wizard-like interface for creating new VMs, managing storage pools, virtual networks. The user interface is quite comfortable (somewhat resembling to VirtualBox's GUI concepts).
- Virt-manager can connect to multiple virtualization host computers (multiple libvirt daemons) and manage whole server sets (like cloud infrastructure) at the same time.
- Virt-manager also has a built-in spice client. Spice is a VNC-like open source protocol, with additional features like sound and USB redirect. Spice is the default interface to VMs with emulated graphics and input devices.
- Follow **RH124 CH15.3.** for the GUI demonstration.

# virt-manager



<https://www.virt-manager.org/wp-content/uploads/2014/01/details.png>

# virt-manager



<https://www.virt-manager.org/wp-content/uploads/2014/01/manager.png>

# Recommended reading

Lecture 12

Chapter 5

## Relevant chapters in RH:

## Read by next lecture:

- RH134 CH11 (NFS, autofs)
- RH254 CH8 (File storage servers/NFS)
- RH124 CH12 (Archiving and copying files)
- RH124 CH15 (Virtualized systems)

- RH254 CH5 (E-mail transmission)
- RH254 CH6 (DNS server)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 12, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK

# Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

# DNS server

Lecture 13

Chapter 1

# DNS history

- In the early 70's Arpanet was a small community, with a few hundred hosts. All stations had their own **static** name resolution data, similar to what the /etc/hosts file is now.
- SRI NIC (Stanford Research Inst. Network Inf. Center) was responsible for maintaining a consistent instance of the whole database. Changes were propagated by sending notification e-mails.
- NIC got overloaded by this, it became almost impossible to follow changes. Name conflicts became quite frequent
- The goal was to create **decentralized**, but globally available datastore, where every participant can administer data related to it's own responsibility/authority.
- Paul Mockapetris in 1984 founded today's DNS: a **distributed database with local control** and isolated responsibilities.

# DNS terminology

- **Resolver:** this can be considered the **client** in DNS's client-server model. The resolver is integrated in all OS's of today. When entering a domain name in an address field, the built-in resolver will find out an IP address belonging to that name and starts sending packets to the given address. Remember that the TCP/IP protocol stack can only take socket addresses (IP:port) into account, names don't play a role in packet forming, filtering and making routing decisions.
- **Authoritative server:** Such a server is **responsible** (has knowledge) only for a relatively small part of the distributed data store. It can emit **authoritative answers** regarding the information that is **directly and primarily** stored in itself. This is the **origin** of the name mapping data. Every organization/entity, which owns a domain, can adjust/update the data in it's own name server, under its own authority for the given zone.

# DNS terminology

- **Recursor** server: This type of DNS server is not the primary origin of any zone, but **holds (cached) and obtains (new) information about many zones** and serves this information to a set of clients (resolvers). The resolvers are in contact with at least one recursor, typically they don't contact auth. nameservers directly.
- **Zone**: A zone is a **portion** of a domain or subdomain, that a given server is **responsible** (authoritative) for. This can be an entire multilevel sub-tree of the domain system or one specific level of the domain system or even only a part of a sub-domain.
- **Delegation**: If a server doesn't hold direct information about a child element, then it holds a **reference** for such a sub-element. Another zone is created for such a child object and it is being '**outsourced**' or delegated to **another server**.

# DNS hierarchy

- DNS involves a strong **hierarchy**, it has a clean **reverse tree structure**. The FQDN (Fully Qualified Domain Name) exactly identifies a specific node in the tree. The starting point of the tree is the . (dot), called the **root node**. There are multiple root server **spread around the globe** for robustness.
- The levels of the tree are **separated with dots**. Domain names can have 127 levels at most, and a nodes from each level can only have a name not longer than 63 characters. Only the root node's name is empty.
- Note that all domain names end with a dot, but we usually don't write it down, because it is implicitly assumed to be there. (e.g. maps.google.com.)
- **TLDs** (Top Level Domains) are under the authority of ICANN (Internet Corp. for Assigned Names and Numbers) TLDs are the **direct children of the root node**.

# DNS hierarchy

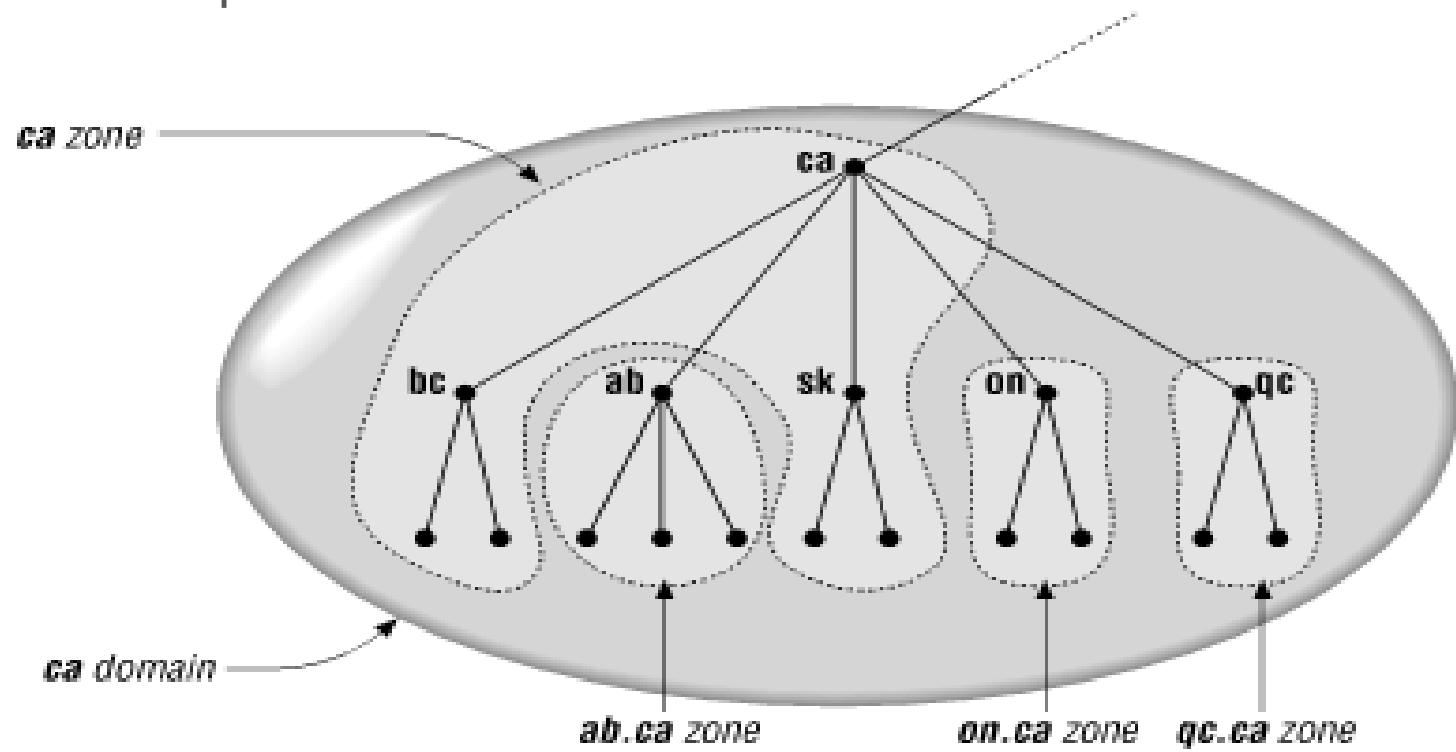
- Originally there were only 7 themed general domains in the US (**gTLDs**: .edu, .gov, .mil, .com, .net, .org, .int) Later on the list of TLDs got expanded with **ccTLDs** (country code domains), like for e.g.: .hu, .uk, .ru, .ch and with **sTLDs** (sponsored TLDs, like .biz, .info)
- Delegation of second level domains (under ccTLDs) usually fall under the **control** of telecommunication agencies/authorities of their **respective countries**. Only licensed registrars are allowed to add/modify entries of a country's TLD zone, gTLDs require an ICANN registrar license.
- There are some technical requirements for registering a second level domain in Hungary, for e.g. myawesome-product.hu. It requires at least two name servers on independent subnets, a valid hostmaster (responsible) mail address and passing of a scripted checking algorithm (<http://www.domain.hu/domain/regcheck>)

# DNS hierarchy, registry

- If requirements are met, a **registrar** will modify the TLD zone's recordset (for a one time fee), and **delegates a zone for the requester**. Maintaining a delegation also goes with a fee that recurs yearly.
- The TLD zone **only contains information necessary for delegation**: A lot of NS records and maybe A (GLUE) records of the related NS entries. (see NS records later)
- The requester selects the primary and secondary authoritative name servers for the given zone (second level domain). This zone then **falls under the requester's control**, but addresses of NSs can't change.
- **Any number of sub-domains can be created and/or delegated freely** below this level, without any technical or legal approval from an administrative office or agency. For. e.g.: shop.myawesomeproduct.hu

# DNS hierarchy

- The **difference** between zones and sub-domains is that domain are always organized into strict levels, but a zone (server) can have the authoritative information of multiple levels of domains, it can also delegate just some specific sub-domains to other zones.



[https://docstore.mik.ua/orelly/networking\\_2ndEd/dns/figs/dns4\\_0211.gif](https://docstore.mik.ua/orelly/networking_2ndEd/dns/figs/dns4_0211.gif)

# DNS hierarchy

- The DNS data retrieval works by a **recursive algorithm**, hence the recursor's name. It involves **crawling** through the levels of the tree structure.
- First assume that there is **no cached** information in any of the nodes (servers) of the graph. A request for `host1.subdomain.example.com`'s address will result in the following steps being taken:
  - One of the **root** servers is being contacted with the **full request** (full address).
  - The root server will determine that it doesn't hold any details about the request, but will notice that it has **delegation information** about the **.com** zone's NS. It will send this referral to the recursor.
  - The recursor contacts the **.com** zone's server and sends the **full request**.

# DNS hierarchy

- Based on the same logic, the .com zone's server will reply with the **example.com** zone's **delegation information**.
- The recursor will contact the example.com's NS, and sends the **full request**.
- Now two options are possible: example.com can have direct information or it can refer subdomain.example.com. In the end one will have **primary** information about host1, the responsible node replies with an **authoritative answer** (aa).
- The IP address of the root servers is to be found in a pre-populated configuration file (root hints).
- This method allows for **global access** with clearly isolated administrative boundaries of the distributed information, but creates an **immense** amount of **load** at the root and near-top zone servers. No computer can possibly have that amount of storage/process capacity.

# DNS hierarchy

- This is where **caching** becomes of great importance: in case the recursor already knows the address from a previous request, it doesn't crawl through all the levels but **replies immediately** from cache.
- A **multilevel cache** is implemented in the recursor, meaning that if another domain of the .com zone is requested from the recursor, it **won't start from the root** node, but from the .com zones NS. Similarly, if a request is received for host2.subdomain.example.com, it won't even reach the example.com NS.
- Now consider the fact that such a cache is implemented in all nodes on all levels: only a very **small percentage** of requests hits the root node. (Also note negative cache)
- The cached information's validity is limited by the **TTL** (time-to-live) value, which can be set separately for **each resource record**. (Don't confuse with TCP/IP TTL!)

# DNS replication

- It is required for **robustness**, that zones of the second level domains have at least **two independent** name servers (on independent networks).
- This type of redundancy is only **recommended** but not required of servers below the second level.
- The **primary** or **master** name server holds the resource records **directly in zone configuration** files created by the administrator.
- The **secondary** or **slave** servers query the DNS information **from the master** server and store it directly in a zone data file. (No manual zone data input required.)
- Both types of servers reply with **authoritative** answers.
- The replication of the whole zone data from master to slave is called **zone transfer**.

# Resource records

- The most common building block of DNS is an **A record**. Eventually this is the whole point of DNS: to map IP addresses to names. An A record holds the mapping information between IPv4 address and domain names.
- A **CNAME record** maps an alias to an existing name. Canonical names can be chained, but should eventually end with a resolvable A record. (Avoid loops!)
- An **AAAA record** is just the same as an A record, only it maps an IPv6 address to a name.
- A **TXT record** can hold a freely selectable text value.

## DNS resource record examples

#owner	TTL	clss	type	data
www.example.com.	300	IN	A	192.168.10.11
dev.example.com.	86400	IN	CNAME	lab.example.com.
example.com.	86400	IN	MX 10	smtp.example.com.

# Resource records

- An **MX record** holds the name of the mail exchange server for a specific e-mail domain. For e.g. if a mail is sent to `george@demo.org`, then the MX record of `demo.org` will be looked up. Lets say it is `smtp.mailservice.net`, then the mail is going to be transferred to the `smtp.mailserver.com` computer (via SMTP protocol) to george's mailbox. A domain can have multiple mail exchangers, MX records have an additional field: priority.
- **NS records** hold the delegation information. E.g. if `example.com`'s zone holds an NS record indicating that `division.example.com`'s name server is `ns1.freedns.org`, it means the zone is delegated to another server.
- A **PTR record** holds information about reverse lookups (see the reverse resolution slide for details)
- DNS can also hold certificates, service info, geo-location, SSH pub. keys, and many more with other record types.

# Resource records

- Every zone has exactly one **SOA record** (start of authority), which states the authority of a server over the zone. If anything (e.g. a sub-domain) gets delegated to another zone, then there will be a SOA record for the sub-object on the targeted server.
- A SOA record holds the name of the primary nameserver for the zone, the responsible persons e-mail address, a serial number and several aging counters.

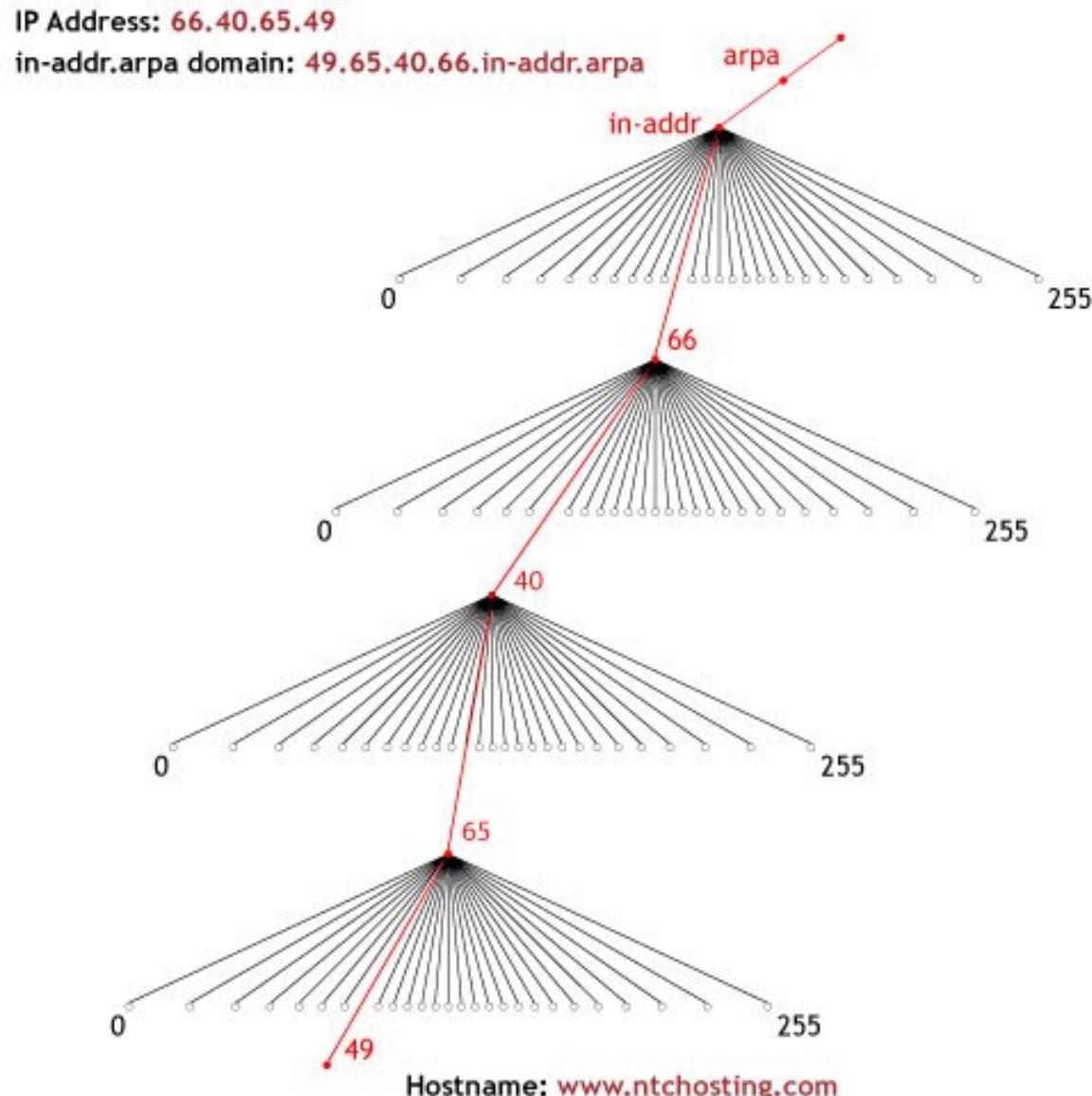
## DNS resource record examples

```
example.com.    IN    SOA    ns1.example.com.  
hostmaster.example.com.  ( 2003080800 ; sn = serial number  
                           ; ref = refresh = 2d  
                           ; ret = update retry = 15m  
                           ; ex = expiry = 2w  
                           ; nx = nxdomain ttl = 1h )
```

# Reverse resolution

- To do a reverse lookup (search for a name with the knowledge of an IP) one would need to do a **full search** on distributed DB, which is not feasible in practice.
- Instead a **separate tree** is maintained for reverse lookups with the same structure, that can be searched with the same algorithm.
- An IP address (in dotted decimal format) has the **same structure** as a domain name. A special root point of this separate tree is the **in-addr.arpa.** node.
- In case of IPv4 addresses there will only be 4 effective levels to look through, with each node having 256 branches. **Delegation** is also possible in this tree too.
- The **downside** is that the tree has to be maintained **separately**, and ISPs don't tend to update/delegate PTR records for end-users. Also note the case of virtual-hosting, where a reverse lookup would be **ambiguous**.

# Reverse resolution



<https://www.ntchosting.com/images/reverse-mapping.jpg>

# Named

- The two most common name servers in Linux enterprise environment are named (BIND) and PowerDNS.
- Berkley Internet Name Domain (bind), in some Linux systems simply name daemon (named), provides both recursor and authoritative functionality in one binary.
- The configuration holds global settings (**/etc/named.conf**) and references to zone files.

```
/etc/named.conf [slave auth. server example]
```

```
options {  
    allow-transfer {none;};  
    allow-recursion {none;};  
    allow-query-cache {none;};  
    additional-from-cache no;  
    listen-on port 53 { !192.168.54.1; any; };  
    directory "/tmp/named";  
};
```

# Named

- Note, that **recursor** functions and listening on LAN interface is intentionally **disabled** in this example. It only works as a publicly accessible auth. **slave** for two zones.

```
/etc/named.conf [slave auth. server example, continued]
```

```
zone "mycompany.hu." IN {  
    type slave;  
    file "/etc/bind/slaves/mycompany.hu.db";  
    masters { 11.12.13.14 port 53; };  
};  
zone "shop.myproduct.hu." IN {  
    type slave;  
    fájl "/etc/bind/slaves/shop.myprdct.hu.db";  
    masters { 13.14.15.16 port 53; };  
};  
#This is not necessary for a non recursor server  
#zone "." {  
#    type hint;  
#    file "/etc/bind/db.root";  
#};
```

# Named

- On a master server the zone-transfer should be allowed only for the intended slaves with **allow-transfer {ip\_of\_slave1;ip\_of\_slave2;};** in the options section.
- A slave downloads (replicates) zone records from the master, a master should always have a local description of the **zone**, usually read from a **zone file**.

```
/etc/named.conf [master server example]
```

```
zone "mycompany.hu." IN {  
    type master;  
    file "/etc/bind/mycompany.hu.zone";  
    allow-update { none; };  
};  
zone "dyndns.myproduct.hu." IN {  
    type master;  
    file "/etc/bind/dyndns.myproduct.hu.zone";  
    allow-update { any; };  
};
```

# Named zone file example

/etc/bind/example.com.zone

```
$ORIGIN example.com.
$TTL 86400
@ IN SOA dns1.example.com. hostmaster.example.com.(
    2001062501 ; serial
    21600       ; refresh after 6 hours
    3600        ; retry after 1 hour
    604800      ; expire after 1 week
    86400 )     ; minimum TTL of 1 day

@ IN NS dns1.example.com.
@ IN NS dns2.example.com.
@ IN MX 10 mail.example.com.
@ IN MX 20 mail2.example.com.
dns1 IN A 10.0.1.1
dns2 IN A 10.0.1.2
ftp IN A 10.0.1.3
mail IN A 10.0.1.4
_ftp._tcp SRV 0 0 21 ftp.example.com.
_http._tcp SRV 0 0 80 www.provider.com.
_ldap._tcp SRV 0 0 389 dc1.example.com.
```

# PowerDNS

- PowerDNS besides being fully compatible with the named zone file format, can also read the zone data from additional sources, most importantly from PostgreSQL or MySQL tables.
- This allows for faster startup but limits for a slightly lower request/second ratio. (This can be important for DNS provider enterprises: when restarting a named servers with 1000's of zones, it opens and reads all the zone files, which can take tens of minutes.)
- The SQL data source also allows for another replication method. Instead of transferring the zones from the master via DNS protocol (over 53/udp or 53/tcp) a PowerDNS slave can simply read the data from the same SQL server or an SQL node from the same SQL cluster.
- PowerDNS has separate binaries for implementing the recursor and the authoritative server functionality.

# Dnsmasq

- In embedded network devices such as Linux based SOHO routers, the most common name server solution is **dnsmasq**. Dnsmasq is the default name server for OpenWRT and similar devices.
- It integrates a **DHCP server** and a **caching recursor**. You have probably noticed before, that in case of connecting to a home Wi-Fi your DHCP assigned DNS server and gateway both have the same IP address. Dnsmasq running on the router acts as a caching name server and **forwards** the queries (by default) to your ISP provided recursor server's address.
- Dnsmasq also provides basic authoritative functions useful for e.g. to overriding name-IP mappings and creating **split horizon DNS** for the office. Dnsmasq also integrates a **TFTP server** for e.g: PXE boot support of disk-less workstations in the office.

# Dynamic DNS

- It is common that ISPs don't assign a **dedicated IP address** to their customers, instead they give one **dynamically** from the available **pool**. It is also possible that an office has mobile workstations that roam from network to network (such as employees with laptops)
- It may be required to access the branch office or the workstations in a branch office regardless of their current IP address. The purpose of Dynamic DNS is exactly to provide an easy interface for **mobile endpoints** to **notify the name server** about their active IP address. The server then changes the corresponding **A records as needed**.
- There are free dynamic DNS providers around. You can also set up your own *named* or *pdns* server with dynamic DNS functionality in mind. Look up **django-powerdns-manager** for a full featured nameserver solution.

# DNS CLI clients

- There are many command line DNS clients, that are mostly used for **diagnostic purposes**. You can use **host**, **dig**, **nslookup** or some other tools to query specific recursing or authoritative name servers about their point of view regarding a resource record.
- By doing this, you can debug the cases where your OS's integrated resolver or your own name server does not return the expected results.
- The **getent** and **gethostip** commands can be used to check the operating systems name resolution answers. Note that the OS does not use the DNS protocol as the exclusive source for names. There can be /etc/hosts entries, or network sources, like an LDAP domain, etc... Windows computers also use the NetBIOS protocol for local name resolution (Samba related).

# DNS CLI clients

- Use the host command in the following manner to query name servers: **host -v -t MX example.com ns1.diag.com** where -v stands for verbose output, the -t switch tells the record type, the first name argument is the subject of the search query and the last argument is the name server to address the request to.
- Use dig the following way: **dig @192.168.1.1 A google.com**, where the @ signifies the server to send the request to, then follows the record type and the name to query.
- Issue **dig axfr @dns-serve tested.domain.name** to initiate a complete zone transfer, like a slave server would do.
- Complete practices **RH254 CH 5.6. and CH 5.7.** (Troubleshooting DNS)

# Mail server

Lecture 13

Chapter 2

# Exam tips

Lecture 13

Chapter 3

# Remote-viewer

- The primary client for qemu/KVM virtual machines is the **remote-viewer** application.
- It uses the **spice://** protocol for delivering content to the client from the virtualization host. This protocol builds on the basics of VNC but also supports sound and USB device redirection. USB end-points connected to the client can be attached through spice to the VM running on a remote server.
- Remote-viewer is integrated into virt-manager, the GUI client to libvirt. It comes already pre-installed on the RH lab's foundationX computers.
- All student will get a personal port number to connect to their individual VMs. For e.g. `spice://172.15.254.251:15001` `spice://172.15.254.251:15002`, etc...
- If not specified otherwise, root password for the exam VM will be 'redhat'.

## Useful tips

- If a task also requires a client computer, other than the exam VM, you can use the built-in destopX virtual computers of the lab.
- The lab's network routing (which falls outside of the scope of the exam VM) is set up correctly to support the exam. If setting the correct network settings for your own VM (as described in the objectives), it will be accessible from the foundationX and desktopX computers, so that you can run tests on your proposed solutions. (E.g. open a page in the web browser.)
- During the exam you are encouraged to install any necessary packages required for completing the objectives, including packages on the client computer used for verifying your solutions. (E.g. install the ftp client into your desktopX VM to test your freshly configured FTP server.)

# Exam objectives

- The point of the exam is to measure the **synthesized knowledge** the students have acquired from learning all the individual topics discussed in the lectures.
- There will be **3 complex tasks**, which will have **multiple topics** covered. For. e.g.: *Migrate an existing MySQL datastore to a dedicated hard drive!* This involves:
  - Creating a partition/LVM volume and a filesystem on top of it.
  - Mounting filesystems, setting automount (fstab) and/or modifying the MySQL service configuration.
  - Copying files while preserving metadata and context.
  - Starting and stopping services.
  - **Planning the required steps and selecting the correct order** to execute them is as much an **important part** of the exam as knowing the commands to be typed.

# Exam objectives

- Another complex task example would be a **service configured with several mistakes** (like wrong file permissions, mistyped configuration directives, incorrect SELinux labels, etc...) to be corrected by the students. The test measures the ability to **analyze the logs** and **concludeing the appropriate actions** to be taken as well as executing them by issuing the relevant commands.
- As part of the exam there will also be 4-5 **simple tasks or questions** that require only a one-line command, for e.g. setting the IP address and gateway to a specific value. A question example: *How many bytes does the initrd image for the currently running kernel occupy on the disk?* To answer this you have to know where such an image is stored, the correct version, and how to find out its size.
- Remember that knowing the commands has only a partial role in passing the exam, the most important thing is to **know when and how to use them**.

# Recommended reading

Lecture 13

Chapter 4

## Relevant chapters in RH:

## Read by next lecture:

- RH254 CH5 (E-mail transmission)
- RH254 CH6 (DNS server)

- RH124 CH16 (Comprehensive review)
- RH134 CH15 (Comprehensive review)
- RH254 CH14 (Comprehensive review)

Use your [rhlearn.gilmore.ca](http://rhlearn.gilmore.ca) login to access the learning materials.

# Thank you for your attention!

Lecture 13, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK