Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

These slides form an outline of the curriculum and contain multiple references to the official Red Hat training materials (codes RH124, RH134 and RH254), since students of this subject are also eligible for the named RH learning materials. Some diagrams and pictures originating from the RH courses will show up in this outline as there is an agreement in place which allows for our Faculty to teach this subject based on the Red Hat curriculum.

All other pictures and diagrams sourced from third parties are explicitly marked with a reference to the origin.

Be aware that the mentioned parts and also this series of slides as a whole are property of their respective owners and are subject to copyright law.

Legal note

All students of PTE-MIK who have officially taken the course „Linux System Administration” are eligible to download these slides from the Faculty’s internal network, and are eligible for license keys for Red Hat System Administration online learning materials at rhlearn.gilmore.ca as part of their training program.

Do not share, redistribute, copy or otherwise offer these learning support materials, license keys or account information either for money or free of charge to anyone, as these materials contain **intellectual property.**

Unauthorized distribution is both against the law and university policy and will result in an in-campus inquiry, suing for damages and/or criminal prosecution by the University of Pécs, Red Hat Inc. and other parties.

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 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>-<acrh>.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 ponit).
- 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 3rd 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 3rd 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 3rd 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 13.6.**

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 13.8.** and **CH 13.9.**

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, than 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 fisrt 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. Its 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 it's functionality, e.g. other filesystem drivers, graphic drivers, images, etc...

GRUB kernel load

- In stage 2 grub also loads it’s 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 fisrt 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 partitons 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 partiton (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 it's 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/*srvcname*** **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/rc*N*.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-genrator, 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 **ctlr+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 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:

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

Read by next lecture:

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

Use your 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