Linux Foundation LFS201 / Linux Foundation Certified System Administrator Exam Prep Course

This repo contains all of the downloaded PDFs, some code, and my Markdown notes taken during the self-paced online LFS201 course hosted by the Linux Foundation. I hope to take the certification exam in October 2019.

Disclaimer: Most emphasis (whether *italics* or **bold** type) used in these documents are from the course materials, but some have been added by me. All screenshots are from course materials. I do not own the images. PDFs are created from the Markdown notes, and as such the Markdown notes will (usually) be more current than the PDFs. The PDF containing all chapters *may* be updated periodically, but will usually be out of date if any changes are made to the Markdown notes.

Table of Contents

40. Backup and Recovery Methods
41. Linux Security Modules
42. Local System Security
43. Basic Troubleshooting
44. System Rescue

Chapter 1 Introduction - Notes

1.3 Introduction

Course primarily designed for *system administrators*.

Tasks you need/want to do: configuring, running, troubleshooting multiple machines in Enterprise environment

Want to learn how to do things properly and natively in Linux

Course **not** designed primarily for:

1. New users without much experience with *any* operating system
2. Kernel or application developers
3. Experienced administrators concentrating on performance tuning

If 1, take LFS101x - Introduction to Linux (found on edX)

If 2, may find material interesting, especially if you have more knowledge regarding:

- operating system theory
- kernel internals
- applications

The more you know about above, the more you will get from course.

If 3, will be skimmed in course, but take more advanced courses dedicated to performance tuning, such as LFS426.

1.4 Relationship to LFS101x

Assumption: You've taken LFS101x, or have equivalent or better knowledge than gained from course.

Recommendation: sign up, or at least do a quick browse through material. It's free!

Some content areas covered in LFS101x covered in LFS201, but at greater detail. Eg. package management

Other topics necessary for system administrators, but not covered in course:

- text editors (vi, emacs, nano, gedit, etc.)
- finding Linux documentation (getting help)
- Manipulating text (sed, grep, awk, cut, paste, tail, head, etc.)
- file utilities (find, locate, file, etc.)
- printing (configuring printers, managing print jobs)
- graphical interfaces and their administration
- bach shell scripting
- system installation

Strong recommendation: at least take a look at LFS101x. If not completely familiar with some basic utilities and procedures used in LFS201, can probably find in LFS101x.

1.5 Course Formatting

Various types of content contained in course, and color coding and formats used to distinguish them:

- **Bold: names of programs or services (or used for emphasis)**
- Light blue: hyperlinks
- Dark blue: text typed at the command line (in notes, shown as distinct page-spanning code blocks as of 2018/01/28, unless embedded inline, in which case they are usually bolded)
- Green: system output at the command line (in notes, shown as part of page-spanning code blocks along with text typed at command line, without $ prefix, as of 2018/01/28)
- Brown: files names and file content (in notes, shown as inline code blocks as of 2019/01/17)

(Will come back to add actual color to text after learning more about Markdown styles)

1.13 Read the Documentation

After discussion of crafting command using Linux utility program, may only show one or few examples of choice or argument, options, etc. May also not consider less common modes of usage -> many programs *incredibly* versatile and have many rarely-used features.

May **explicitly** say "read the **man** page for more detail" from time to time, but assume that it is **implicitly** stated all the time. Must develop habit of reading easily access documentation for your Linux distribution without prompting, for all but simplest utilities.

Besides **man**, most utility programs also have built-in help and usage information: usually access by using **--help** option, eg. **df --help**.

1.14 Target Platform

Course designed to work on x86-based platforms, either on native hardware or running as **virtual machine (VM)**. Almost everything also relevant for othr architectures, but real or virtual x86 cover majority of deployments.

No consideration of 32-bit; assume 64-bit throughout. If running 32-bit x86, mostly everything will still be valid, although not tested by course.

Also *not* targeting **embedded Linux** platforms. Although deployments based on same Linux kernel and use many same user space ingredients, usually no talk about system administration for devices used as appliances. Has important basic ingredients that are very quite different, eg. using different boot loader programs and filesystems. Resources, eg. memory, processor power, and storage capacity, also differ greatly.

Same considerations apply to Android (particular form of embedded Linux). While kernel is still Linux, filesystem and application space extremely different from what is found on standard Linux systems.

1.15 Command Line vs. Graphical Interface

Many administrative tasks can be accomplished either from command line or from within graphical application. More flexibility and additional capability in command line approach (no indirection layer). Drawback for command line: administrator may have more to remember or need to look up info when task needs to be accomplished.

Variety of graphical desktop environments in Linux. Two most common:

- GNOME
- KDE

Each with multiple versions (generations).

Course not involved with graphical interfaces: too much variation, many servers do not have graphical interface installed.

For course, work from command line on local machine using either console or terminal emulator such as **gnome-terminal** running on graphical desktop. Can also work remotely through **ssh** or **vpn** session. Almost everything apply equally well in remote session case, except for when physical access to machine needed, eg. booting into rescue environment.

1.16 Target Linux Distributions

Infinite list of Linux distributions.

Course concentrating on Enterprise Linux distributions. Vast majority use:

1. **Red Hat Enterprise Linux**: RHEL, including derived distributions CentOS and Scientific OS (some differences in package updating for offical RHEL systems). Oracle Linux just copy of RHEL. **Fedora** in Red Hat family, actually be seen as upstream for RHEL. Current versions quite similar to RHEL 7. However, rare to use Fedora in Enterprise deployments, as too cutting edge and changes important features (such as kernel version) too often for comfort where stability is key.
2. **SUSE**: full-blown Enterprise distribution with significant market share, closely related to openSUSE (Fedora:RHEL, SUSE:openSUSE). No clone like CentOS and RHEL, so openSUSE target system.
3. **Debian**: also includes Ubuntu, derived from Debian. Most discussion of Ubuntu LTS 16.04, LTS 18.04 or 17.04 release. Other Debian-derivatives, eg. Linux Mint, not much different.

Arch Linux and Gentoo also crop up in Enterprise deployments, but since both rolling distributions (no specific release, constantly updated), best used only by very expert administrators in Enterprise environments.

1.17 Installation: What to Use for this Course

Refer to document prepared by Linux Foundation.

1.19 Change, Repetition, Holy Wars

1. Things change in Linux: constantly evolving, both at technical level (including kernel features) and distribution/interface level.
2. Some repeated things in course material: with comprehensive course, revisit topics previously covered + short reviews are helpful to refresh memory without searching for previous section.
3. Course avoids *holy wars*: many areas with strong preference disagreements in Linux (and open-source) community. Eg. best editor, **emacs** vs **vi**; best graphical desktop, **GNOME** vs **KDE**, etc. Particular emphasis chosen simply to keep content clean, eg. more on GNOME simply because bigger user base.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 2 Linux Filesystem Tree Layout - Notes

2.3 Learning Objectives:

- Explain why Linux requires the organization of one big filesystem tree, and what the major considerations are for how it is done.
- Explain the role played by the Filesystem Hierarchy Standard.
- Describe what must be available at boot in the root (`/`) directory, and what can be available only once the system has started.
- Explore each of the main subdirectory trees, explain their purposes, and examine their contents.

2.4 One Big Filesystem

Linux: consists of one big filesystem tree (similar to all UNIX-based operating systems) -> usually diagrammed as inverted tree with **root directory**, `/` , at top of tree.

May be more than one (or even many) distinct file systems **mounted** at various points within this one large logical filesystem -> appear as subdirectories. Distinct filesystems usually on different partitions, which can also be on any number of different devices, including on network.

Applications: do not care about what physical device files reside on, just looks like one big filesystem.

In past, different UNIX-like OS organized tree in various ways + even various Linux distributions had differences -> resulted in difficulty and frustration writing applications/accomplishing system administration tasks on more than one kind of system

Result -> Linux ecosystem worked hard establish standardized procedures to minimize pain.

2.5 Data Distinctions

Taxonomy for what kind of info has to be read/written when talking about how files/data organized in one big directory tree. Two kinds of distinctions:

1. **Shareable vs non-shareable**: Shareable data can be shared between different hosts, non-shareable data must be specific to particular host. Eg. home directories -> shareable, device lock files -> non-shareable.
2. **Variable vs. static**: static data -> binaries, libraries, documentation + anything that doesn't change without system administrator assistance. Variable data -> anything that may change without systems administrator's help

These logical distinctions embodied as different kinds of information residing in various directories (or partitions and filesystems).

2.6 FHS Linux Standard Directory Tree

Filesystem Hierarchy Standard (FHS): administered by The Linux Foundation (originally by the Free Standards Group), specifies main directories that must be present and their purposes.

Specifies standard layout -> simplifies predictions of file locations.

Most Linux distributions respect FHS, but none follow exactly. Last official version -> old enough, does not take into account some new developments.

Distributions -> like to experiment, and some experiments become generally accepted.

FHS document for more information.

2.7 Main Directory Layout

Linux distributors -> spend lot of time making sure filesystem layout is coherent and evolves correctly over time. List of main directories normally found under `/`:

Directory	In FHS?	Purpose
`/`	Yes	Primary directory of entire file system hierarchy
`/bin`	Yes	Essential executable programs that much be available in **single user mode**
`/boot`	Yes	Files needed to boot system, such as kernel, **initrd** or **initramfs** images, boot configuration files, bootloader programs
`/dev`	Yes	**Device nodes**, used to interact with hardware and software devices
`/etc`	Yes	System wide configuration files
`/home`	Yes	Use **home directories** including personal settings, files, etc
`/lib`	Yes	Libraries required by executable binaries in `/bin` and `/sbin`
`/lib64`	No	64-bit libraries required by executable binaries in `/bin` and `/sbin`, for systems which can run both 32-bit/64-bit programs
`/media`	Yes	Mount points for removable media eg. CDs, DVDs, USB sticks etc
`/mnt`	Yes	Temporarily mounted filesystems
`/opt`	Yes	Optional application software packages
`/proc`	Yes	Virtual pseudo-filesystem giving information about the system and processes running on it. Can be used to alter system parameters
`/sys`	No	Same as `/proc`. Similar to **device tree** and is part of **Unified Device Model**
`/root`	Yes	Home directory for the **root** user
`/sbin`	Yes	Essential system binaries
`/srv`	Yes	Site-specific data served up by system. Seldom used.
`/tmp`	Yes	Temporary files; on many distributions lost by reboot and may be ramdisk in memory
`/usr`	Yes	Multi-user applications, utilities, data: theoretically read-only
`/var`	Yes	Variable data that changes during system operation

May be additional distribution-specific directories found under root directory:

- `/misc`, for miscellaneous data
- `/tfpboot`, for booting using **tftp**. If files in directory, related to diskless workstation booting

Note: does not violate FHS to have other directories, does violate to have components in directories *other* than those dictated by standard.

2.8 The Root (`/`) Directory and Filesystems

There may be multiple partitions/filesystems joined together w hile entire filesystem can be view ed as one big tree.

Partition and filesystem that contains root directory itself: rather special, often in special dedicated partition. Other components mounted later (`/home` , `/var` , `/opt`)

Root partition: must contain all essential files required to boot system and then mount all other filesystems. Needs utilities, configuration files, boot loader info, other essential startup data. It must be able to:

- Boot system
- Restore system from system backups on external media, eg. tapes, other removable media, NAS etc
- Recover/repair system; experienced maintainer must have tools to diagnose + reconstruct damaged system

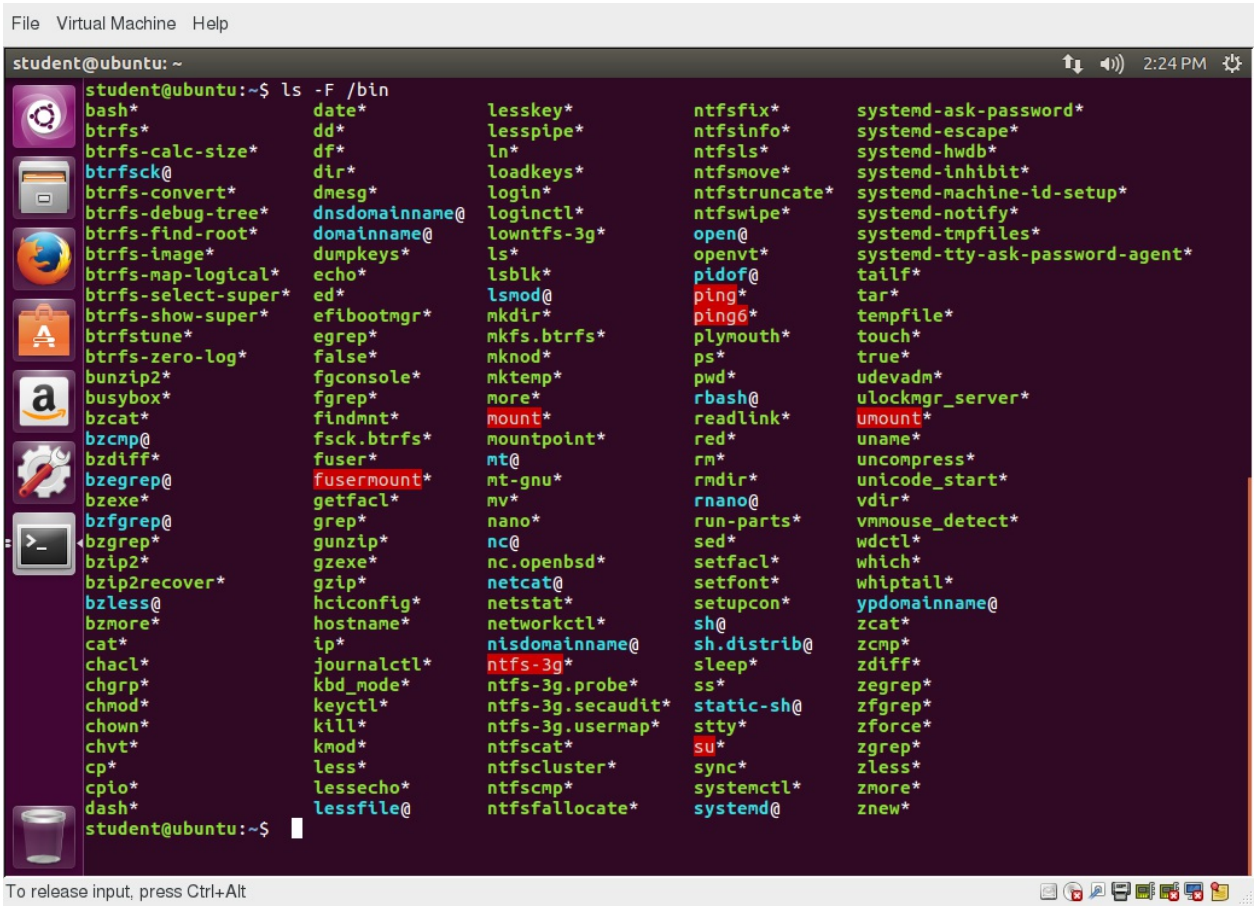
FHS: "no application or package should create new subdirectories of the root directory".

2.9 `/bin`

Very important:

- Contains executable programs + scripts needed by both system administrators/unprivileged users, required w hen no other filesystems have yet been mounted, eg. w hen booting into **single user** or recovery mode
- May also contain executables w hich are used indirectly by scripts
- May *not* include any subdirectories

Ubuntu



Required programs in `/bin/` :

cat, chgrp, chmod, chown, cp, date, dd, df, dmesg, echo, false, hostname, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd, rm, rmdir, sed, sh, stty, su, sync, true, umount, uname [may include **test**]

Optionally may include: **csh, ed, tar, cpio, gunzip, zcat, netstat, ping**

Nonessential command binaries -> `/usr/bin` if not good enough to be placed in `/bin` (eg. programs required only by non-root users)

Note: some recent distributions -> no separation between `/bin` and `/usr/bin` (also `/sbin` and `/usr/sbin`), just one directory with symbolic links (to preserve two directory view). Believe time-honored concept of enabling possibility of placing `/usr` on separate partition mounted after boot -> obsolete.

2.10 `/boot`

Must contain essential files for booting system in `/boot` directory and its subdirectories. Two files, absolutely essential:

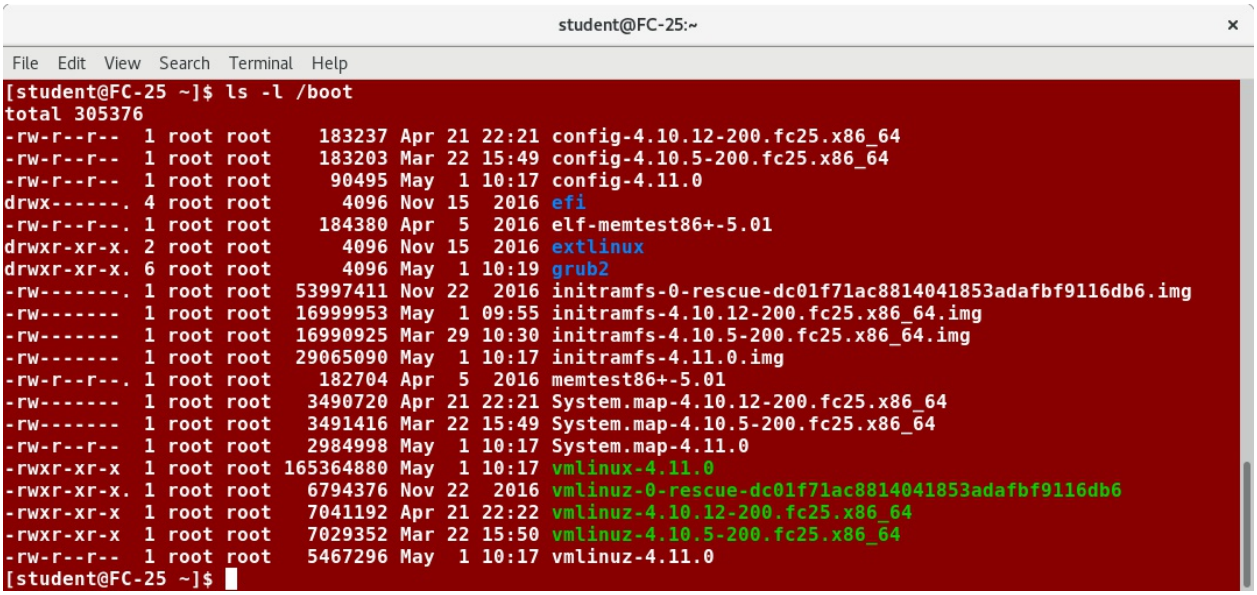
- **vmlinuz**: compressed Linux kernel
- **initramfs**: **initial RAM filesystem**, mounted before real root filesystem becomes available

May have longer names, which depend on kernel version (exact form depends on Linux distribution). **initramfs** may be called **initrd**, **initial RAM disk** (older method but name survives).

Stores data used before kernel begins executing user-mode programs. May include: saved master boot sectors, sector map files, other data not directly edited by hand. Exact contents vary by distribution + time.

Before, essential files often placed directly in `/` instead of separate `/boot` directory (following traditional UNIX practices) -> now considered obsolete.

RHEL



2.11 Other Files and Directories in `/boot`

Multiple kernel versions available in `/boot`, with four files available for each version (Choice between kernels make by using GRUB at boot time).

Two other files besides **vmlinuz** and **initramfs**:

- **config**: configuration file used when compiling kernel, here just for bookkeeping and reference when debugging

- **System.map**: kernel **symbol table**, very useful for debugging. Gives hexadecimal addresses of all kernel symbols

Neither required for booting or running system.

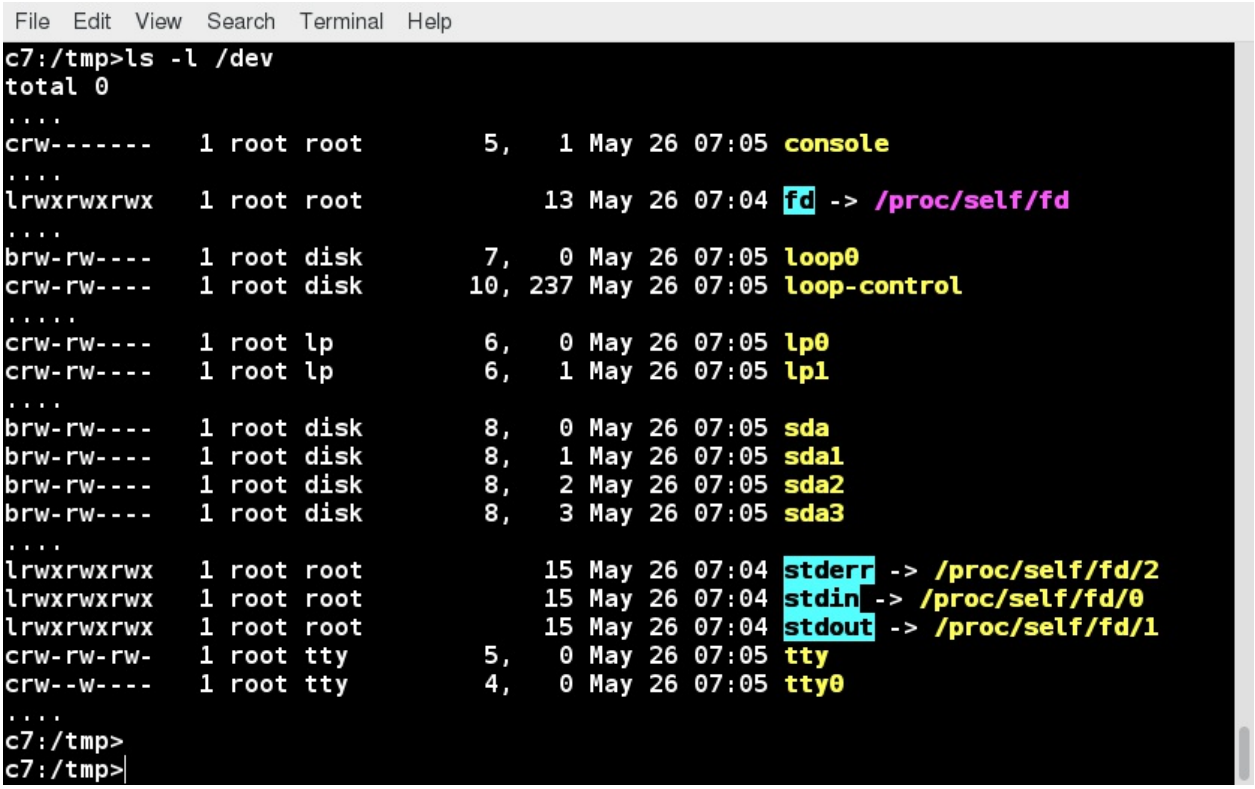
Distributions may place other files/directories in `/boot`, eg. saved master boot sectors, other data not hand-edited.

2.12 `/dev`

Contains **special device files** (**device nodes**) -> represent devices built into or connected to system. Files essential for proper system function.

Device files represent **character** (byte-stream) + **block I/O** devices.

Network devices -> no device nodes in Linux, instead referenced by name, eg. **eth1**, **wlan0**.



2.13 `/etc`

Contains machine-local configuration files and some startup scripts; *no* executable binary programs. Files and directories include:

csh.login, exports, fstab, ftpusers, gateways, gettydefs, group, host.conf, hosts.allow, hosts.deny, hosts.equiv, hosts.lpd, inetd.conf, inittab, issue, ld.so.conf, motd, mtab, mtools.conf, networks, passwd, printcap, profile, protocols, resolv.conf, rpc, securetty, services, shells, syslog.conf.

Configuration files and directories added to `/etc` by distrbutions. Eg. Red Hat: number of other directories (eg. `/etc/sysconfig`, where number of system configuration files live).

Other important subdirectories: `/etc/skel` (contains **skeleton** files used to populate newly created home directories), `/etc/init.d` (contains start up + shut down scripts when using System V initialization)

2.14 `/home`

User directories conventionally placed under `/home` on Linux systems (eg. `/home/coop`, `/home/student`, etc). All personal configuration, data, executable programs contained in `/home` hierarchy. May also contain subdirectories for various groups/associations of users (eg. `/home/students`, `/home/staff`, `/home/aliens`, etc).

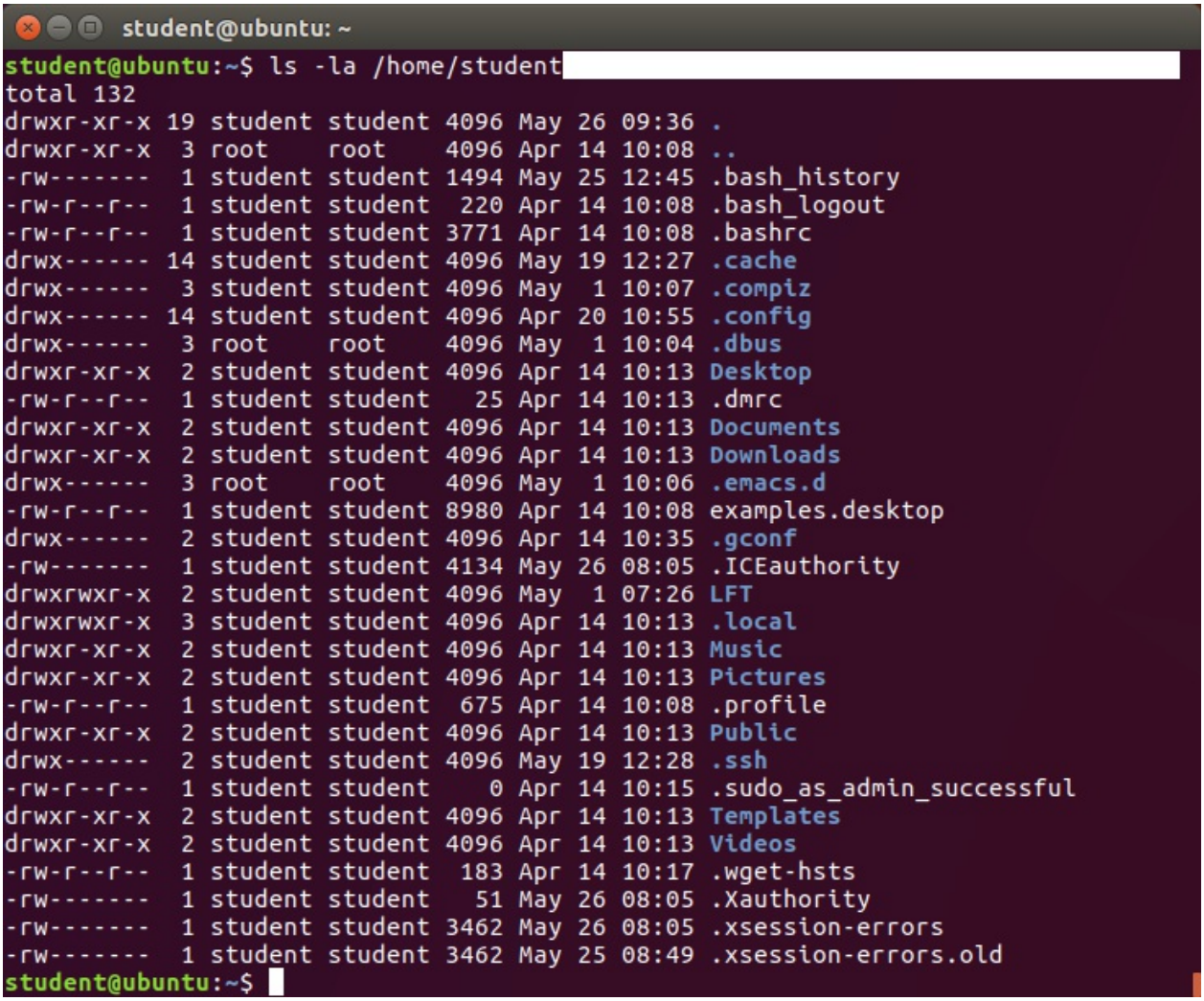
Other UNIX-like OS -> concept of `/home` exists, but subtly different. Eg. Solaris: user directories created in `/export/home`, then **automount** facility eventually mount in `/home`. Why -> usual situation: home directory may be anyw here on corporate netw ork (probably on NFS server), and home directory mounted automatically upon use.

Linux has same **automount** facilities, but many users not aw are + on self-contained systems, concept of NFS mounts -> not apply.

User can alw ays substitute environment variable **$HOME** for their root directory, or shorthand ~; so, follow ing commands equivalent:

```
shell $ ls -l $HOME/public_html $ ls -l ~/public_html
```

One exception: home directory for **root** user in Linux systems *always* found under `/root`. Some older UNIX systems may use `/` instead, causing clutter.



2.15 `/lib` and `/lib64`

Should contain only libraries needed to execute binaries in `/bin` and `/sbin`. Libraries particularly important for booting system + executing commands w ithin root filesystem.

Kernel modules (often device/filesystem drivers) located under `/lib/modules/<kernel-version-number>`.

PAM (**P**luggable **A**uthentication **M**odules) files stored in /lib/security.

Systems supporting both 32-bit/64-bit binaries must keep both libraries on system. On Red Hat-based systems, separate directories for 32-bit (`/lib`) and 64-bit (`/lib64`) libraries.

2.16 `/media`

Typically used to mount filesystems on removable media, eg. CDs, DVDs, USB drives, floppies (heh).

Modern Linux systems: mount media dynamically upon insertion. **udev** creates directories under `/media` + mounts removable filesystems there (names set w ith **udev** rules specified in configuration files). Directories used as mount points under `/media` -> disappear upon unmounting + removal.

If more than one partition and filesystem on media -> more than one entry on `/media` . On many Linux distributions, file manager (eg. Nautilus) pops up upon media mounting.

Note: removable media pop up under `/run/media/[username]...` for some new er distributions (eg. SUSE, RHEL 7). `/run` discussed later.

2.17 `/mnt`

Provided so system administrator can temporarily mount filesystem w hen needed. Common use: netw ork filesystems:

- **NFS**
- **Sam ba**
- **CIFS**
- **AFS**

Old systems used `/mnt` for files now mounted under `/media` (or `/run/media`) in modern systems

Generally, should *not* be used by installation programs. Another temporary directory not currently used serves better.

2.18 `/opt`

Designed for softw are packages that w ish to keep all/most files in one isolated place, rather than scatter across system in directories shared by other softw are. Eg. package name **dolphy_app**: all files reside in directories under `/opt/dolphy_app` , including `/opt/dolphy_app/bin` for binaries, `/opt/dolphy_app/man` for **m an** pages.

Makes installing/uninstalling softw are relatively easy -> all files in one convenient isolated location in predictable + structured manner. Also easy for system administrators to determine nature of each file w ithin package.

Note: also easy to install/uninstall w ith clear sense of manifests/locations w ithout antisocial behavior if using packaging systems eg. **RPM**, **APT**.

In Linux, `/opt` directory often used by proprietary softw are providers, or those w ho like to avoid distribution variance complications. Eg. `/opt/skype` , `/opt/google` (containing `chrome` , `earth` , `talkplugin`).

Reserved for local system administrator use: `/opt/bin` , `/opt/doc` , `/opt/include` , `/opt/info` , `/opt/lib` , `/opt/man` . Packages must be able to function w ithout programs being in these special directories (but may also provide files linked/copied to these reserved directories).

2.19 `/proc`

Mount point for **pseudo-filesystem**, w here all info resides only in memory (not on disk). `/proc` directory empty on non-running

system (like `/dev`).

kernel -> exposes some important data structures through `/proc` entries. Each active process on system -> has own subdirectory, gives detailed info about state of process, used resources, history.

`/proc` entries -> often termed **virtual files**, have interesting qualities. Most listed as zero bytes in size, but contain large amount of info when viewed.

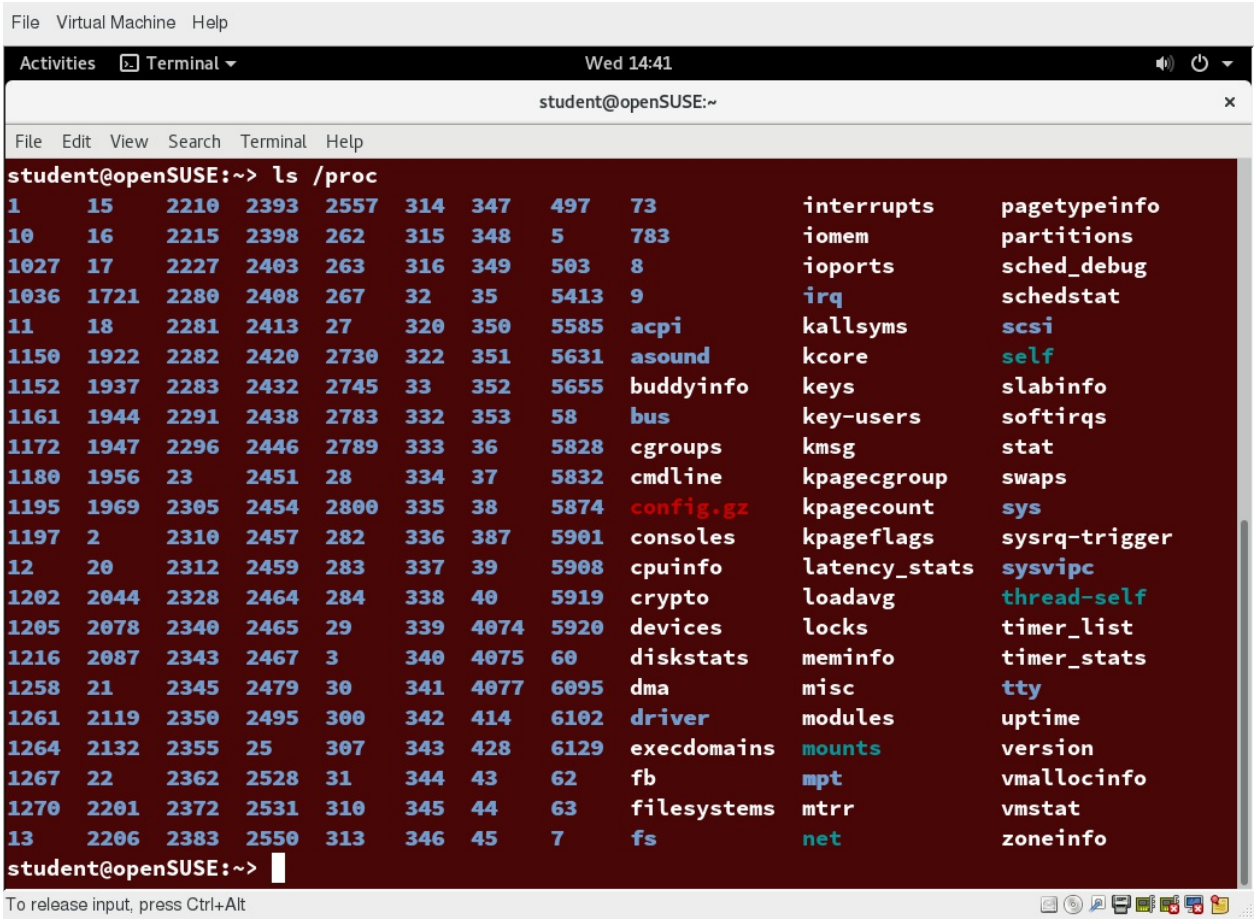
Most virtual files time/date settings -> current time/date, reflects constant change. Info in file *only* obtained when viewed, not constantly/periodically updated.

Important pseudo-files provide up-to-date moment glimpse of system's hardware, eg. `/proc/interrupts`, `/proc/meminfo`, `/proc/mounts`, `/proc/partitions`

Also provide system configuration info + interface, eg. `/proc/filesystems`, `/proc/sys`.

Files containing info on similar topic grouped in virtual directories/sub-directories, eg. `/proc/scsi` -> info for all SCSI devices, **process** directories -> info about each running process on system

Entries in `/proc` examined throughout course -> detailed look for kernel configuration + system monitoring.



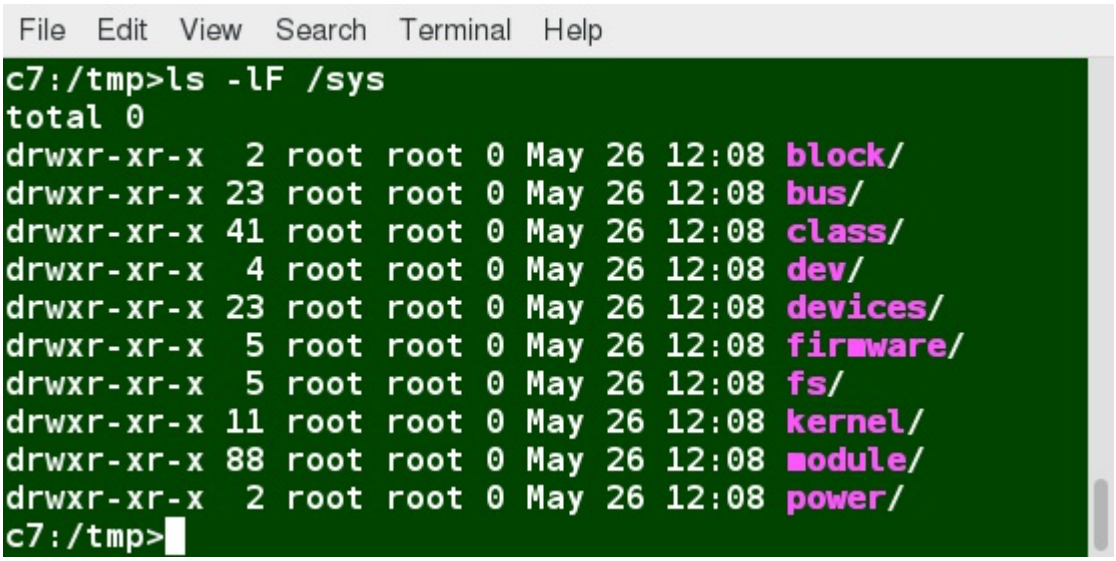
2.20 `/sys`

Mount point for **sysfs pseudo-filesystem** where all info resides in memory, not on disk. Empty on non-running system (like `/dev`, `/proc`).

sysf -> used to gather info about system, modify behavior while running. Resembles `/proc`, but younger + adheres to strict standards about entries contained, eg. almost all pseudo-files in `/sys` contain only one line or value; no long entries like found in

`/proc`.

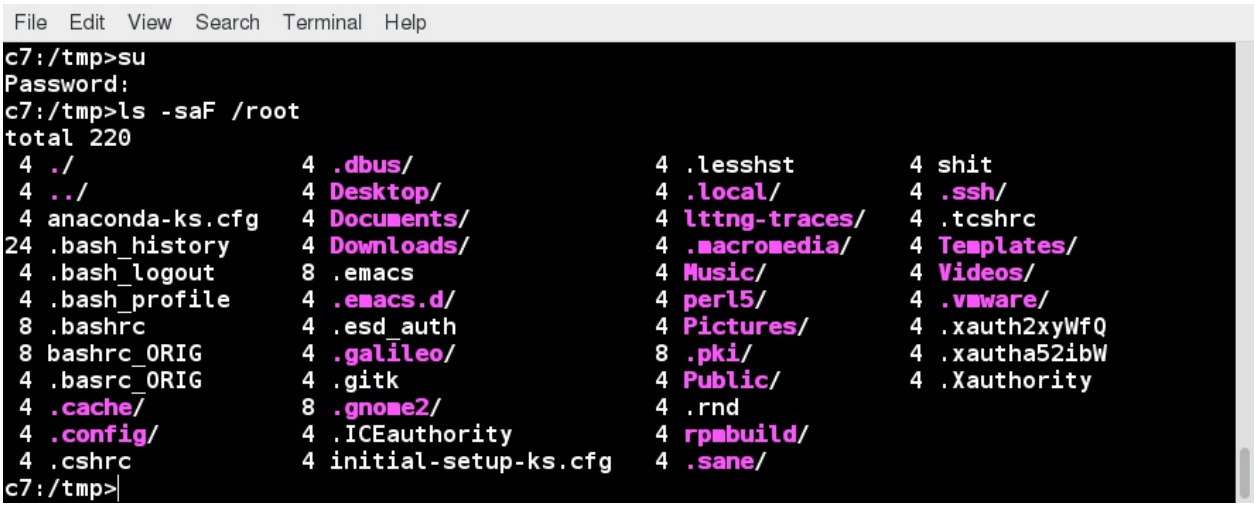
Entries in `/sys` also examined throughout course.



2.21 `/root`

Home directory for root user (pronounced "slash-root")

Root account that owns `/root` -> *only* use for actions requiring superuser privilege. Use another account for non-privileged user actions.



2.22 `/sbin`

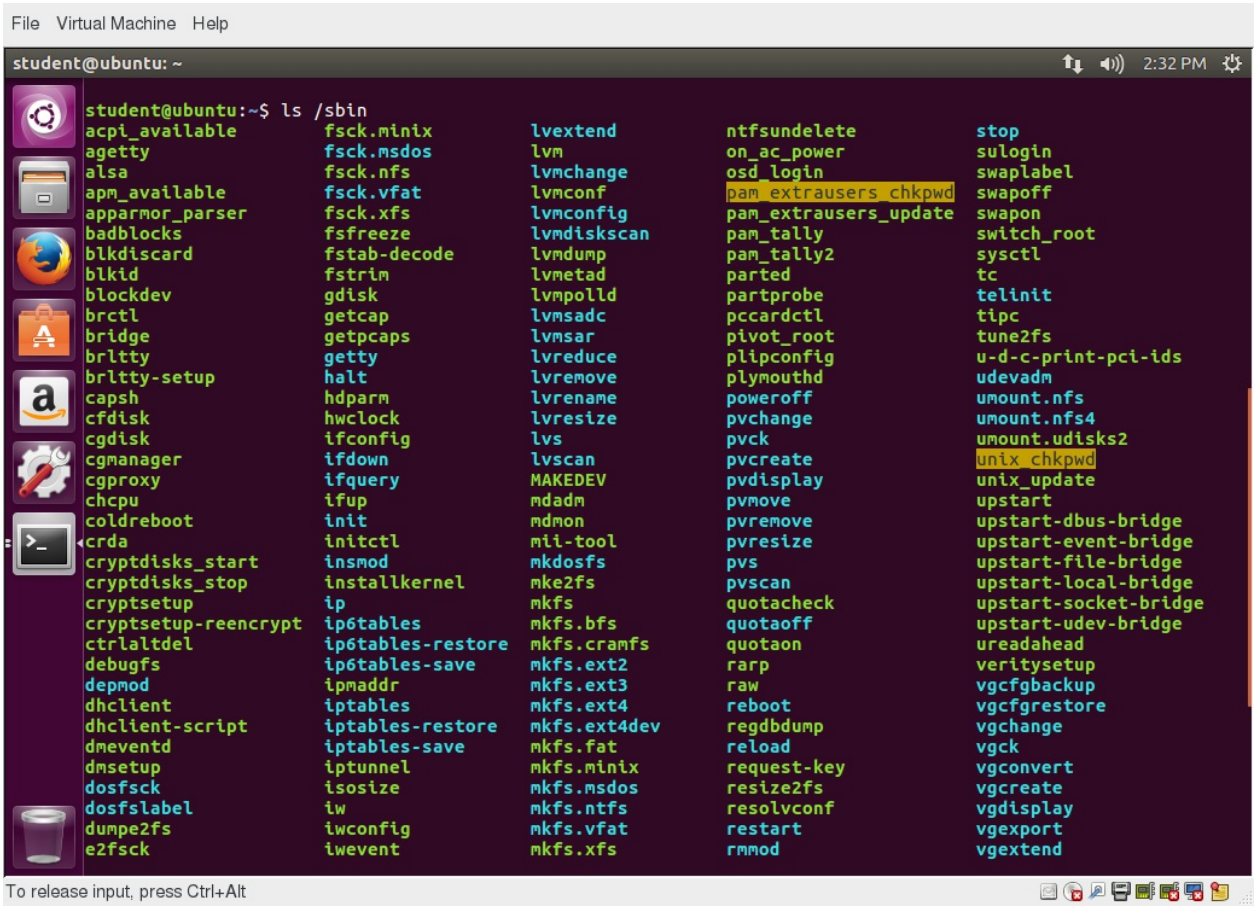
Contains binaries essential for booting, restoring, recovering, repairing (in addition to binaries in `/bin`). Must also be able to mount other filesystems on `/usr`, /home, other locations if needed (once root filesystem known to be in good health during boot).

Included programs (if subsystems installed):

fdisk, fsck, getty, halt, ifconfig, init, mkfs, mkswap, reboot, route, swapon, swapoff, update.

Note: some recent distributions -> no separation between `/sbin` and `/usr/sbin` (also `/bin` and `/usr/bin`), just one directory with symbolic links (to preserve two directory view). Believe time-honored concept of enabling possibility of placing `/usr` on

separate partition mounted after boot -> obsolete.



2.23 `/srv`

Contains site-specific date, served by this system.

Main purpose of specifying this: users may find location of data files for particular service, services w hich require single tree for read-only data w ritable data, scripts (eg. **cgi** scripts) reasonable placed.

Methodology of `/srv` subdirectory naming: unspecified, currently no consensus. One method for structuring data: by protocol, eg. **ftp, rsync, www**, and **cvs**.

Often confusion about w hat is best to go in `/var` vs. `/srv`. Some system administrators rely heavily on `/srv`, others ignore.

2.24 `/tmp`

Used to store temporary files, can be accessed by any user of application. Files on `/tmp` cannot be depended to stay around long-term. Some distributions:

- Run automated **cron** jobs -> remove any file older than 10 days typically, unless purge scripts modified to exclude files. RHEL 6 policy.
- Remove contents of `/tmp` w ith every reboot. Ubuntu policy.
- Modern: utilize virtual filesystem, using `/tmp` only as mount point for **ram disk** using **tmpfs** filesystem. Default policy of recent Fedora systems. All info lost during system reboot; `/tmp` indeed temporary.

In last case, avoid creating large files on `/tmp`: occupy memory instead of disk, easy to hard/crash system through memory exhaustion. Guideline: avoid large files in `/tmp`, but plenty of applications violating policy. Possible to put them elsew here (eg. by

specifying environment variable), but many users not aware of how to configure + all users have access to `/tmp`.

Can cancel policy on systems using **systemd**, eg. Fedora, by:
`shell $ sudo systemctl mask tmp.mount`
followed by system reboot.

2.25 `/usr`

Can be thought of as **secondary hierarchy**. Used for files not needed for system booting. Need not reside in same partition as root directory, can be shared among hosts using the same system architecture across network.

Software packages -> should *not* create subdirectories directly under `/usr`. Some symbolic links to other locations may exist for compatibility purposes.

Typically read-only data. Contains binaries not needed in single user mode. Contains `/usr/local` (local binaries and such), `/usr/share/man` (**man** pages)

Note: some recent distributions -> no separation between `/bin` and `/usr/bin` (also `/sbin` and `/usr/sbin`), just one directory with symbolic links (to preserve two directory view). Believe time-honored concept of enabling possibility of placing `/usr` on separate partition mounted after boot -> obsolete.

Directories under `/usr`:

Directory	Purpose
`/usr/bin`	Non-essential binaries and scripts, not needed for single user mode. Generally, means user applications not needed to start system
`/usr/etc`	Non-essential configuration files (usually empty)
`/usr/games`	Game data
`/usr/include`	Header files used to compile applications
`/usr/lib`	Library files
`/usr/lib64`	Library files for 64-bit
`/usr/local`	Third-level hierarchy (for machine local files)
`/usr/sbin`	Non-essential system binaries
`/usr/share`	Read-only architecture-independent files
`/usr/src`	Source code and headers for the Linux kernel
`/usr/tmp`	Secondary temporary directory

2.26 `/var`

Contains **variable** (or **volatile**) data files that change frequently during system operation, including:

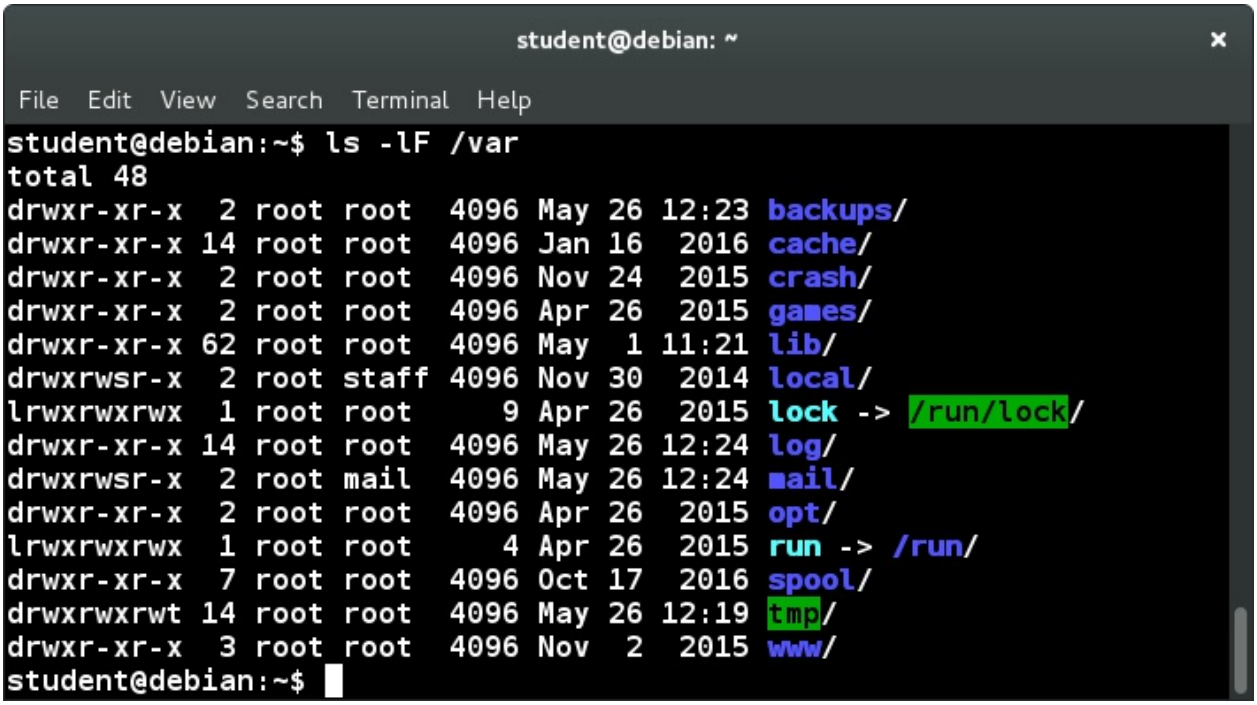
- Log files
- Spool directories, files for printing, mail queues, etc
- Adminsitrative data files
- Transient/temporary files, eg. cache contents

Cannot be mounted as read-only filesystem (obviously).

Often considered good idea to mount `/var/` as separate file system for security reasons. If directory filled up, should not lock up system.

`/var/log` directory -> most log files located, `/var/spool` directory -> local files for processes such as mail, printing, cron jobs stored while awaiting action.

Directory	Purpose
`/var/ftp`	Used for ftp server base
`/var/lib`	Persistent data modified by programs as they run
`/var/lock`	Lock files used to control simultaneous access to resources
`/var/log`	Log files
`/var/mail`	User mailboxes
`/var/run`	Information about the running system since the last boot
`/var/spool`	Tasks spooled or waiting to be processed, eg. print queues
`/var/tmp`	Temporary files to be preserved across system reboot, sometimes linked to `/tmp`
`/var/www`	Root for website hierarchies



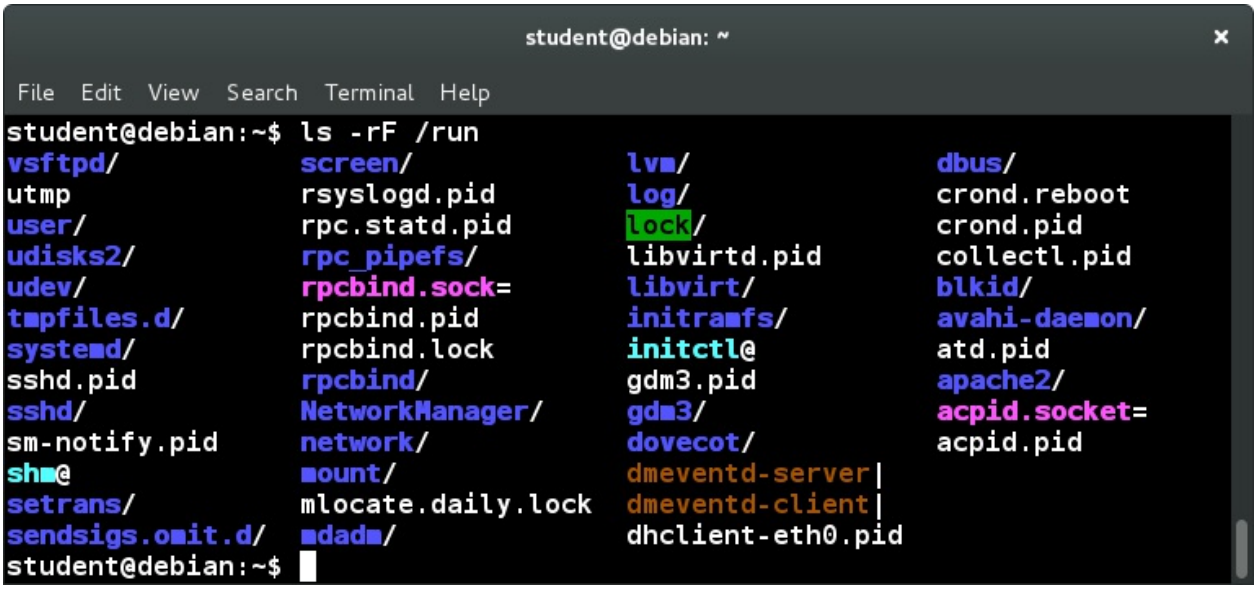
2.27 `/run`

Not formally accepted by FHS, although proposed. New directory tree mounted at `/run` in use for several years by major Linux distributions.

Purpose: store **transient** files: containing runtime information, need to be written early in system startup, do not need to be preserved when rebooting.

Implemented as empty mount point with **tmpfs** ram disk (like `/dev/shm`) mounted there at runtime. Pseudo-filesystem existing only in memory.

Some existing locations (`/var/run`, `/var/lock`) just symbolic links to directories under `/run`. Other locations, depending on distribution taste, may also just point to locations under `/run`.



##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 3 Processes - Notes

3.3 Learning Objectives:

- Describe a process and the resources associated with it.
- Describe the role of the **init** process.
- Distinguish between processes, programs, and threads.
- Understand process attributes, permissions, and states, and know how to control limits.
- Explain the difference between running in user and kernel modes.
- Describe the **daemon** processes.
- Understand how new processes are forked (created).
- Use **nice** and **renice** to set and modify priorities.
- Understand how shared and static libraries are used.

3.4 Processes, Programs, and Threads

Process: executing program and associated resources, including environment, open files, signal handlers, etc. Same program may be executing more than once simultaneously -> responsible for multiple processes.

Two or more tasks (threads of execution) can share various resources (eg. entire memory spaces or particular memory areas, open files, etc.). When **everything shared** circumstance -> **multi-threaded** process.

Other OS -> distinction between full **heavy weight** and **light weight** processes. Heavy weight process may include number of light weight processes, or just one.

Linux -> different situation. Each thread of execution considered individually, difference between heavy and light -> sharing of resources, somewhat faster context switching between threads of execution.

Linux -> fast job of creating, destroying, switching between processes (unlike other OS) -> model adopted for multi-threaded application resembles multiple processes: each thread scheduled individually and normally (like stand-alone process). Done instead of involving more levels of complication (eg. having separate method of scheduling along threads of process, having scheduling method between different processes).

Linux -> also respects POSIX and other standards for multi-threaded processes, eg. each thread returns same process ID (called thread group ID internally) while returning distinct thread ID (called process ID internally). May lead to confusion for developers, invisible to administrators.

3.5 The init Process

First user process on system -> **init**, has `process ID` **= 1**. Started as soon as kernel initialized + root filesystem mounted.

Runs until system shutdown -> last user process to be terminated. Serves as ancestral parent of all other user processes, both directly/indirectly.

3.6 Processes

Process -> instance of program in execution. Number of different states (eg. running, sleeping). Every process has **pid** (Process ID), **ppid** (Parent Process ID), **pgid** (Process Group ID); also has program code, data, variables, file descriptors, environment.

init: first user process run on system, becomes ancestor of all subsequent processes running on system (except for those initiated directly from kernel, show up with **[]** around name in **ps** listing).

If parent process dies before child -> **ppid** of child set to 1 (ie. process id adopted by **init**). (Note: in recent Linux systems using systemd, **ppid** set to 2, corresponding to internal kernel thread **kthreadd** that has taken over role of adopter of orphaned children from **init**).

zombie (or defunct) process: child process which terminated (either normally/abnormally) before parent that has not waited for it and examined its exit code. Releases almost all resources and remains only to convey exit status. Checking on adopted children + let terminated process die gracefully -> one function of **init** process -> sometimes known as **zombie killer**, or **child reaper** (yeesh).

Processes controlled by **scheduling** (completely preemptive). Only kernel has right to preempt process, processes cannot do it to each other.

Largest `PID` limited to 16-bit number, 32768, for historical reasons. Possible to alter this value by changing `/proc/sys/kernel/pid_max`, since may be inadequate for larger servers. As processes created, will eventually reach `pid_max`, and then will start again at `PID = 300`.

3.7 Process Attributes

Attributes for all processes:

- Program being executed
- Context (state)
- Permissions
- Associated resources

Every process -> executing some program.

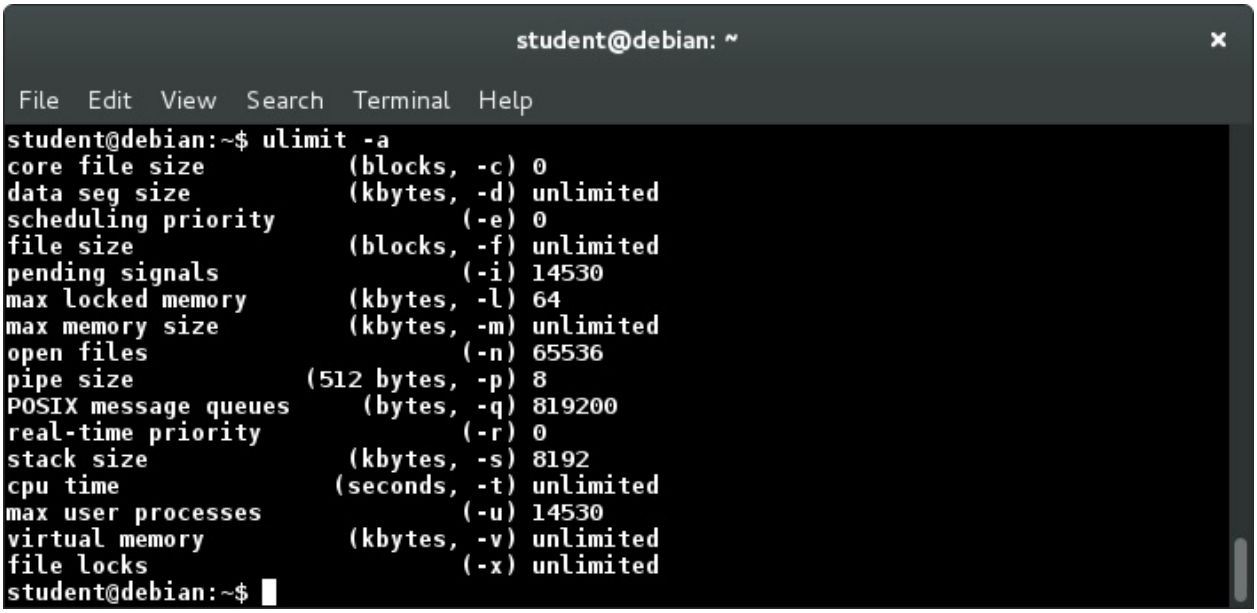
Context of process: snapshot of itself by trapping state of its CPU registers, where it is executing in program, what is in process' memory, and other information.

Processes scheduled in and out when sharing CPU time with other (or put to sleep while waiting for some condition to be fulfilled, such as user request or data arrival) -> ability to store entire context when swapping out process, and restore upon execution resumption *critical* to kernel's ability to do **context switching**.

3.8 Controlling Processes with ulimit

ulimit: built-in bash command, displays or sets a number of resource limits associated with processes running under shell.

Screenshot: running **ulimit** with `-a` argument.



If run as root, result in `Command not found`, since limits shell-specific.

System administrator may need to change some above values in either direction to:

- **Restrict** capabilities so user/process cannot exhaust system resources (eg. memory, cpu time, max number of processes on system)
- **Expand** capabilities so process does not run into resource limits, eg. server handling many clients -> default of 1024 open files, impossible to perform w ork

Tw o kinds of limits:

- **Hard**: max value that user can raise resource limit to. Set only by root user
- **Soft**: current limiting value. Can be modified by user, but cannot exceed hard limit

Can set any particular limit by:

```
$ ulimit [options] [limit]
```

as in

```
$ ulimit -n 1600
```

w hich increases max number of file descriptors to 1600.

Note: changes only affect current shell. TO make changes effective for all logged-in users, need to modify `/etc/security/limits.conf` (nicely self-documented file), and reboot.

3.9 Process Permissions and setuid

Every process -> permissions based on w hich user invoked + on w ho ow ns program file.

Programs w ith **s** execute bit -> have different **effective user id** than their **real user id** (explained later in local security section). Referred to as **setuid** programs. Run w ith user id of user w ho **ow ns** program. Non-**setuid** pgroams run w ith permissions of user w ho **runs** them.

setuid programs owned by root -> well-known security problem.

passwd program -> example of setuid program. Any user can run. When user executes program, process must run with root permission to update write-restricted `/etc/passwd` and `/etc/shadow` files where user passwords maintained.

3.10 More on Process States

Processes can be in on of several possible states. Main ones:

- **Running**: process currently executing on CPU/CPU core, or sitting in **run queue** waiting new time slice. Will resume when scheduler decides deserving to occupy CPU, or when another CPU idle and scheduler migrates process to idle CPU.
- **Sleeping** (ie, **Waiting**): waiting on request (usually I/O) that it has made + cannot proceed until request completed. When request completed, kernel will wake up process, put back on run queue, given time slice on CPU when scheduler decides to do so.
- **Stopped**: suspended process. Commonly experienced when programmer wants to examine executing program's memory, CPU registers, flags, other attributes. One examination done, process may be resumed. Generally done when process run under debugger or user hits `Ctrl-Z`.
- **Zombie**: states when process terminates, and no other process (usually parent) inquires about its exit state (ie, reaped it). ALso called **defunct** process. Has released all its resources, except exit state and entry in process table. If parent of any process dies, process **adopted** by **init** (`PID = 1`) or **kthreadd** (`PID = 2`).

3.11 Execution Modes

Process (or any particular thread of multi-threaded process) -> may be executing in **user mode** or **system mode** (usually called **kernel mode** by kernel developers) at any given time.

Instructions that can be executed -> depends on mode + enforced at hardware level (not software).

Mode: state of processor (eg. in multi-core/multi-CPU system, each unit can be in own individual state). *Not* state of system.

Intel parlance:

- User mode: **Ring 3**
- System mode: **Ring 0**

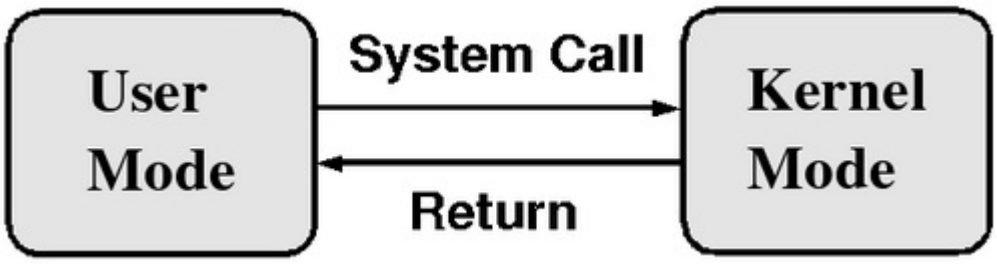
3.12 User Mode

Processes execute in **user mode** (lesser privileges than in kernel mode), *except* when executing **system call** (described in next section).

When process started -> isolated in own user space to protect from other processes -> promotes security + greater stability. Sometimes called **process resource isolation**.

Each user mode process -> own memory space, where parts may be shared with other processes. Except shared memory segments, user process not able to read/write into/from memory space of any other process.

Even process run by root user or as **setuid** program -> runs in user mode, *except* when jumping into system call. Has only limited ability to access hardware.



3.13 Kernel Mode

Kernel (system) mode -> CPU full access to all hardware on system including peripherals, memory, disks etc. If application needs access to these resources, must issue **system call** -> causes **context switch** from user mode to kernel mode. Must follow this procedure when reading/writing files, creating new process etc.

Application code -> never runs in kernel code, only system call itself (which is kernel code). When system call complete, return value produced -> process returns to user mode with inverse context switch.

Other times when system is in kernel mode that have nothing to do with processes -> handling hardware interrupts, running scheduling routines, other management tasks for system.

3.14 Daemons

Daemon process: background process, sole purpose to provide specific service to users of system:

- Can be quite efficient because only operate when needed
- Many started at boot time
- Names often (but not always) end with **d**
- Examples: **httpd**, **systemd-udevd**
- May respond to external events (**systemd-udevd**) or elapsed time (**crond**)
- Generally have no controlling terminal and no standard input/output devices
- Sometimes provide better security control

When using SysVinit, scripts in `/etc/init.d` directory start various system daemons. These scripts invoke commands as arguments to shell function named `daemon`, defined in `/etc/init.d/functions` file.

3.15 Creating Processes in a Command Shell

When user executes command in command shell interpreter such as **bash**:

- New process created (forked from user's login shell)
- Wait system call puts parent shell process to sleep
- Command loaded onto child process' space via **exec** system call (other words, code for command replaces **bash** program in child process' memory space)
- Command completes executing, child process dies via exit system call
- Parent shell re-awakened by death of child process, proceeds to issue new shell prompt. Parent shell then waits for next command request from user, at which time cycle repeated

If command issued for **background** processing (by adding ampersand - `&` - at end of command line), parent shell skips wait request and free to issue new shell prompt immediately, allowing background process to execute in parallel. Otherwise, for **foreground** requests, shell waits until child process completes or stopped via signal.

Some shell commands (eg. echo, kill) built into shell itself, do not involve loading of program files. For these commands, no **fork** or **exec** issued for execution.

3.16 Kernel-Created Processes

Not all processes created, or **forked** from user parents. Linux kernel directly creates two kinds of processes on own initiative:

- **Internal kernel processes**: take care of maintenance work (eg. making sure buffers get flushed out to disk, load on different CPUs balanced evenly, device drivers handle work queued up for them to do, etc.). Often run as long as system running, sleeping except when they have something to do.
- **External user processes**: run in user space like normal applications, but started by kernel. Very few, usually short lived.

Easy to see which processes are of this nature; when you run command such as

```
$ ps -elf
```

to list all processes on system while showing parent process IDs, all will have `PPID = 2` referring to **kthreadd**, internal kernel thread whose job is to create such processes, names will be encapsulated in square brackets such as `[ksoftirqd/0]`.

3.17 Process Creating and Forking

Average Linux system -> always creating new processes. Often called **forking**; original parent process continues running while new child process starts.

Back when most computers had single processor: usually configured so parent initially pause while child started to run; UNIX expression: "Children come first." However with modern multi-CPU systems, both tend to run simultaneously on different CPUs.

Often, rather than just fork, one follow with **exex**, where parent process terminates and child process inherits process ID of parent. Term **fork and exec** used so often, sometimes though of as one word.

spawn: program often used by older UNIX systems, similar in many ways to **fork and exec**, differing in details. Not part of POSIX standard, not normal part of Linux.

To see how new processes may start:

Example 1: Web server that handles many clients

- May launch new process every time new connection is made with client
- May also simply start only new **thread** as part of same process

Linux -> not much difference on technical level between creating full process or just new thread, each mechanism takes about same time and roughly same amount of resources

Example 2: **sshd** daemon

- started when **init** process executes **sshd** init script
- Script responsible for launching **sshd daemon**
- Daemon process listens for **ssh** requests from remote users
- When request received, **sshd** *creates a new copy of itself* to service request
- Each remote user get own copy of **sshd** daemon running to service their remote login
- **sshd** process will start login program to validate remote user
- if authentication successful, login process will fork off shell (eg. **bash**) to interpret user commands, and so on

3.18 Using nice to Set Priorities

Process priority -> can be controlled through **nice** and **renice** commands. Since early days of UNIX, idea: *nice* process lowers

priority to yield to others. Thus, higher **niceness**, lower priority.

Niceness values range from -20 (highest priority) to +19 (lowest priority). Normal way to run **nice**:

```
$ nice -n 5 command [ARGS]
```

which would increase niceness by 5. Equivalent to doing:

```
$ nice -5 command [ARGS]
```

If no **nice** value given, default to increase niceness by 10. If no arguments given at all, reports current niceness:

```
$ nice
0

$ nice cat &
[1] 24908

$ ps -l
F S UID   PID  PPID C PRI NI ADDR SZ WCHAN  TTY       TIME CMD
0 S 500  4670  4603 0 80   0 - 16618 wait   pts/0 00:00:00 bash
0 S 500 24855  4670 0 80   0 - 16560 wait   pts/0 00:00:00 bash
0 T 500 24908 24855 0 90  10 - 14738 signal pts/0 00:00:00 cat
0 R 500 24909 24855 0 80   0 - 15887 -      pts/0 00:00:00 ps
```

Note: increasing niceness of process does not mean it won't run. May even get all CPU time if nothing else with which to compete.

If large increment/decrement value outside -20 to 19 range supplied, increment value truncated.

3.19 Modifying the Nice Value

Default: only superuser can decrease niceness; i.e., increase priority. However, possible to give normal users ability to decrease niceness within predetermined range by editing `/etc/security/limits.conf`.

To change niceness of already running process, easy to use **renice** command:

```
$ renice +3 13848
```

which will increase niceness by 3 of the process with `pid = 13848`. More than one process can be done at same time and also some other options, so see **man renice**.

3.21 Static and Shared Libraries

Programs built using **libraries** of code, developed for multiple purposes and used/reused in many contexts.

Two types of libraries:

- **Static**: code for library functions inserted in program at **compile time**. Does not change thereafter even if library updated
- **Shared**: code for library funcions loaded into program at **run time**. If library changed after, running program runs with new library modifications

Using shared libraries -> more efficient, because then can be used by many applications at once. Memory usage, executable sizes, application load time reduced.

Shared libraries also called **DLL** s (**D**ynamic **L**ink **L**ibraries).

3.22 Shared Library Versions

Shared libraries must be carefully versioned. If significant change to library and program not equipped to handle it, serious problems expected. Sometimes known as **DLL Hell**.

Thus, programs can request specific **major** library version rather than latest one on system. Usually program will always use latest **minor** version available.

Some application providers will use static libraries bundled into program to avoid problems. Down side: if improvements or bugs/security hole fixes in libraries, may not make it into applications in timely fashion.

Shared libraries file extension: **.so**. Typical full name: `libc.so.N`, where `N` is major version number.

Under Linux, shared libraries carefully versioned:

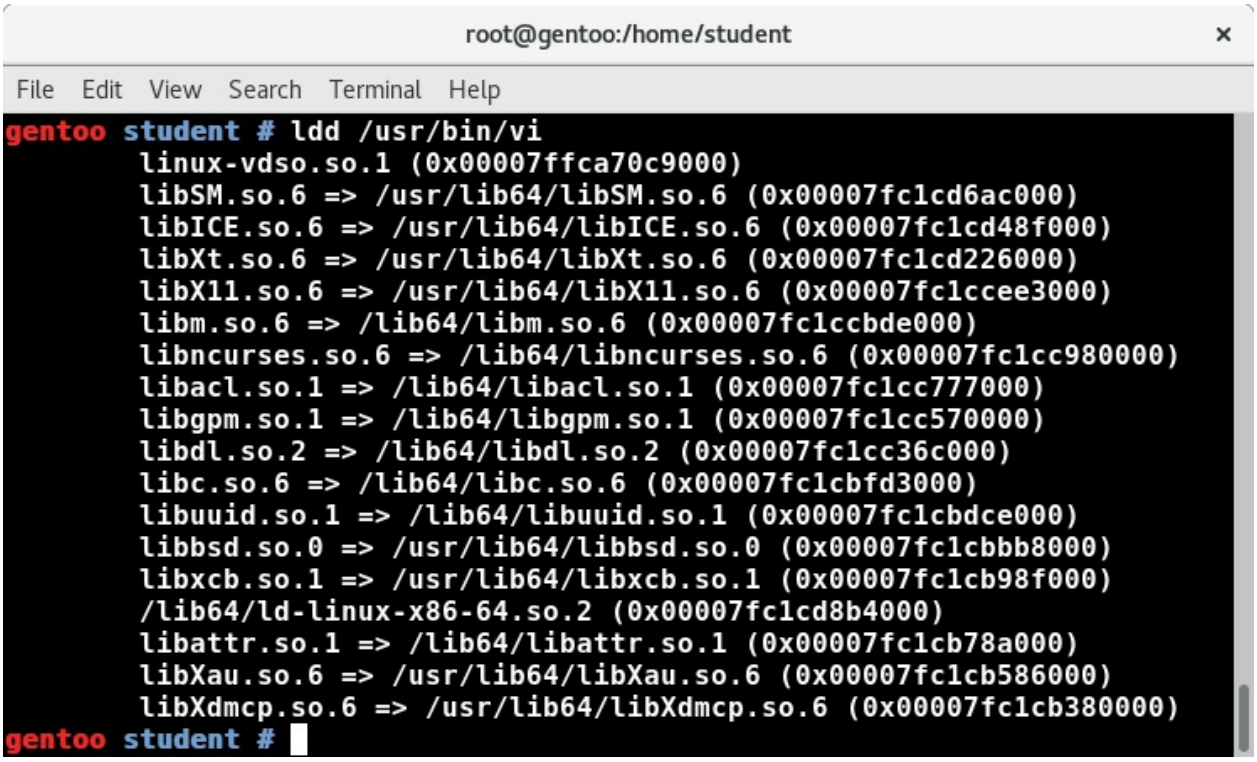
```
c7:/usr/lib64>ls -lF libgdbm.so*
lrwxrwxrwx 1 root root 16 Apr 9 2015 libgdbm.so -> libgdbm.so.4.0.0*
lrwxrwxrwx 1 root root 16 Apr 9 2015 libgdbm.so.4 -> libgdbm.so.4.0.0*
-rwxr-xr-x 1 root root 36720 Jan 24 2014 libgdbm.so.4.0.0*
c7:/usr/lib64>
```

so program that just asks for `libgdbm` gets `libgdbm.so` and the others for specific major and minor versions.

3.23 Finding Shared Libraries

Program which uses shared libraries has to be able to find them at runtime.

ldd can be used to ascertain which libraries required by executable. Shows **soname** of library and what files it points to.



ldconfig generally run at boot time (but can be run anytime). Uses file `/etc/ld.so.conf` which lists directories searched for shared libraries. Must run **ldconfig** as root and shared libraries should be stored in system directories only when stable and useful.

Besides searching data base built up by **ldconfig**, linker will first search any directories specified in environment variable `LD_LIBRARY_PATH` (colon separated list of directories, as in `PATH` variable):

```
$ LD_LIBRARY_PATH=$HOME/foo/lib
$ foo [args]
```

or

```
$ LD_LIBRARY_PATH=$HOME/foo/lib foo [args]
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 4 Signals - Notes

4.2 Introduction

Signals: used to emit notifications for processes to take action in response to often unpredictable events. May be caused from within process itself, or external events such as other processes.

Many signals fatal, resulting in process termination. Death can sometimes be averted if program designers decide to handle (subvert) certain termination signals.

Many signals more benign, just informative or request other kinds of actions. Possible to send signals (including those that induce termination) from command line using **kill**, **killall**, **pkill**.

4.3 Learning Objectives:

- Explain what signals are and how they are used.
- Know the available signals and types of signals available in Linux.
- Use kill, killall, and pkill to send signals from the command line.

4.4 What are Signals?

Signals: one of oldest methods of **Inter-Process Communication** (**IPC**), used to notify processes about **asynchronous** events (or exceptions).

By asynchronous, the signal-receiving process may:

- Not expect event to occur
- Expect event, but not know when it is most likely to occur

Example: user decides to terminate running program. Could send signal to process through kernel to interrupt and kill process.

Two paths by which signals sent to process:

- From kernel to user process, as result of exception or programming error
- From user process (using system call) to the kernel which will then send to user process. Process sending signal can actually be same as one receiving signal

Signals only sent between processes owned by same user or from process owned by superuser to any process.

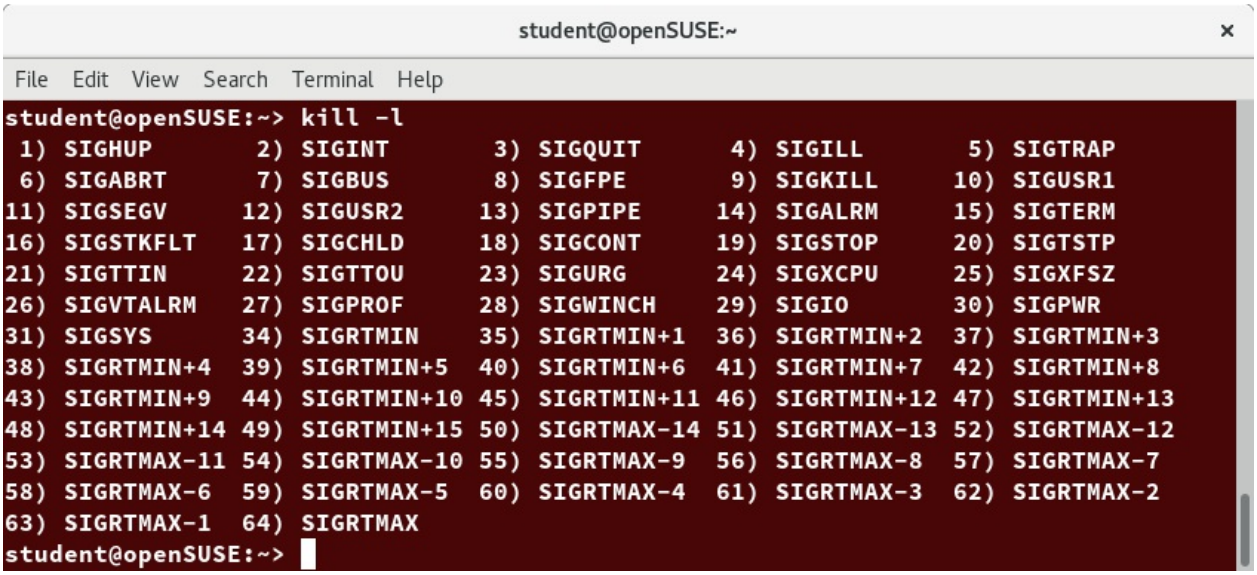
When process receives signal, what it does depends on way program is written. Can take specific actions (coded into program) to **handle** signal or it can just respond according to system defaults. Two signals (`SIGKILL` and `SIGSTOP`) cannot be handled and always terminate program.

4.5 Types of Signals

Number of different types of signals, particular signal dispatched indicates type of event (or exception) occurred. Generally, signals used to handle:

1. Exceptions detected by hardware (eg. illegal memory reference)
2. Exceptions generated by environment (eg. premature death of process from user's terminal)

TO see list of signals in Linus, along with numbers, do `kill -l`, as reflected in screenshot.



Signals from `SIGRTMIN` on termed **real-time signals**, relatively recent addition. No predefined purpose, differ in some important ways from normal signals. Can be queued up and handled in **FIFO** (**F**irst **I**n **F**irst **O**ut) order.

Meaning attached to signal type indicates event that caused signal to be sent. While users can explicitly send any signal type to their processes, meaning attached may not longer be implied by signal number or type, can be used in any way that the process desires.

Further documentation: **man 7 signal**.

Available Signals for the x86 Plarform

Signal	Value	Default Action	POSIX?	Meaning
`SIGHUP`	1	Terminate	Yes	Hangup detected on controlling terminal or death of controlling process
`SIGINT`	2	Terminate	Yes	Interrupt from keyboard
`SIGQUIT`	3	Core dump	Yes	Quit from keyboard
`SIGILL`	4	Core dump	Yes	Illegal Instruction
`SIGTRAP`	5	Core dump	No	Trace/breakpoint trap for debugging
`SIGABRT/SIGIOT`	6	Core dump	Yes	Abnormal termination
`SIGBUS`	7	Core dump	Yes	Bus error
`SIGFPE`	8	Core dump	Yes	Floating point exception
`SIGKILL`	9	Terminate	Yes	Kill signal (cannot be caught or ignored)
`SIGUSR1`	10	Terminate	Yes	User-defined signal 1
`SIGSEGV`	11	Core dump	Yes	Invalid memory reference

`SIGUSR2`	12	Terminate	Yes	User-defined signal 2
`SIGPIPE`	13	Terminate	Yes	Broken pipe: write to pipe with no readers
`SIGALRM`	14	Terminate	Yes	Timer signal from alarm
`SIGTERM`	15	Terminate	Yes	Process termination
`SIGSTKFLT`	16	Terminate	No	Stack fault on math co-processor
`SIGCHLD`	17	Ignore	Yes	Child stopped or terminated
`SIGCONT`	18	Continue	Yes	Continue if stopped
`SIGSTOP`	19	Stop	Yes	Stop process (cannot be caught or ignored)
`SIGTSTP`	20	Stop	Yes	Stop types at tty
`SIGTTIN`	21	Stop	Yes	Background process requires tty input
`SIGTTOU`	22	Stop	Yes	Background process requires tty output
`SIGURG`	23	Ignore	No	Urgent condition on socket (4.2 BSD)
`SIGXCPU`	24	Core dump	Yes	CPU time limit exceeded (4.2 BSD)
`SIGXFSZ`	25	Core dump	Yes	File size limit exceeded (4.2 BSD)
`SIGVTALRM`	26	Terminate	No	Virtual alarm clock (4.2 BSD)
`SIGPROF`	27	Terminate	No	Profile alarm clock (4.2 BSD)
`SIGWINCH`	28	Ignore	No	Window resize signal (4.3 BSD, Sun)
`SIGIO/SIGPOLL`	29	Terminate	No	I/O now possible (4.2BSD) (System V)
`SIGPWR`	30	Terminate	No	Power Failure (System V)
`SIGSYS/SIGUNUSED`	31	Terminate	No	Bad System Called. Unused signal.

4.6 kill

Process cannot send signal directly to another process, must ask kernel to send signal by executing **system call**. Users (including superuser) can send signals to other processes from command line or scripts using **kill**:

```
$ kill 1991
$ kill -9 1991
$ kill -SIGKILL 1991
```

where user sending signal to process with `PID = 1991`. If signal number not given (as in first example), default to send `SIGTERM` **(15)**, terminate signal that can be handled. Program can take elusive action or clean up after itself, rather than die immediately. If this signal ignored, user can usually send `SIGKILL` **(9)** (cannot be ignored), to terminate with extreme prejudice.

Name **kill** -> really bad name, misnomer that survives for historical reasons. Although often used to kill (terminate) processes, command's real function: send any and all signals to processes, even totally benign informative ones.

4.7 killall and pkill

killall: kills all processes with given name, assuming user has sufficient privilege. Uses command name rather than process ID:

```
$ killall bash
$ killall -9 bash
$ killall -SIGKILL bash
```

pkill sends signal to process using selection criteria:

```
$ pkill [-signal] [options] [pattern]
```

For example:

```
$ pkill -u libby foobar
```

will kill all of **libby**'s processes with a name of **foobar**.

Another example:

```
$ pkill -HUP rsyslogd
```

makes **rsyslog** re-read its configuration file.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 5 Package Management Systems - Notes

5.3 Learning Objectives:

- Explain why software package management systems are necessary.
- Understand the function of both binary and source packages.
- Enumerate the main available package management systems.
- Understand why two levels of utilities are needed: one that deals with just bare packages, and one that deals with dependencies among packages.
- Explain how creating your own package enhances the control you have over exactly what goes in software and how it is installed.
- Understand the role of source control systems, and git in particular.

5.4 Software Packaging Concepts

Package management systems supply tools that allow system administrators to automate installing, upgrading, configuring and removing software packages in known, predictable, consistent manner. These systems:

- Gather and compress associated software files into single package (archive), which may require one or more other packages to be installed first.
- Allow for easy software installation or removal
- Can verify file integrity via internal database
- Can authenticate origin of packages
- Facilitate upgrades
- Group packages by logical features
- Manage dependencies between packages

Given package may contain executable files, data files, documentation, installation scripts, configuration files. Also included -> **metadata** attributes such as version numbers, checksums, vendor information, dependencies, descriptions, etc.

Upon installation, all information stored locally into internal database, which can be queried for version status and update information.

5.5 Why Use Packages?

Software package management systems widely seen as one of biggest advancements Linux brought to enterprise IT environments. By keeping track of files, metadata in automated, predictable, reliable way, system administrators can use package management systems to make installation processes scale to thousands of systems without requiring manual work on each individual system. Included features:

- Automation: No need for manual installs and upgrades
- Scalability: Install packages on one system, or 10,000 systems
- Repeatability and predictability
- Security and auditing

5.6 Package Types

Several different types of packages:

- **Binary** packages contain files ready for deployment, including executable files and libraries. Architecture-dependent, must be compiled for each type of machine
- **Source** packages used to generate binary packages. Should always be able to rebuild binary package (eg. by using `rpmbuild --rebuild` on **RPM**-based systems) from source package. One source package can be used to multiple architectures
- **Architecture-independent** packages contain files, scripts that run under script interpreters, + documentation, configuration files
- **Meta-packages**: essentially groups of associated packages that collect everything needed to install relatively large subsystem, eg. desktop environement, office suite, etc.

Binary packages -> ones that system administrators have to deal with most of the time.

On 64-bit systems that can run 32-bit programs, may have two binary packages installed for given program, one with **x86-64** or **amd64** in name, and other with **i386** or **i686** in name.

Source packages helpful in keeping track of changes and source code used to come up with binary packages. Usually not installed on system by default, but can always be retrieved from vendor.

5.7 Available Package Management Systems

Two very common package management systems:

1. **RPM** (**Red Hat Package Manager**): system used by all Red Hat-derived distributions (eg. Red Hat Enterprise Linux, CentOS, Scientific Linux, SUSE and its related community openSUSE distribution)
2. **dpkg** (**Debian Package**): system used by all Debian-derived distributions, including Debian, Ubuntu, Linux Mint.

Other package management systems: **portage/emerge** used by Gentoo, **pacman** used by Arch, specialized ones used by Embedded Linux systems and Android.

Another ancient system: supply packages as **tarballs** without any real management or clean removal strategies. Approach still marks Slackware, one of oldest Linux distributions.

Most of time, either **RPM** or **dpkg**, only ones considered in course.

5.8 Packaging Tool Levels and Varieties

Two levels to packaging systems:

1. **Low Level Utility**: simply installs/removes single package, or list of packages, each one individually/specifically named. Dependencies not fully handled, only warned about:

    - If another package needs to be installed, first installation will fail
    - If package needed by another package, removal will fail

    rpm and **dpkg** utilities play this role for packaging systems that use them.

2. **High Level Utility**: solves dependency problems:

    - If another package/group of packages needs to be installed before software can be installed, such needs will be satisfied
    - If removing package interferes with another installed package, administrator given choice of aborting or removing all affected software

    yum, **dnf**, **zypper** and Package Kit utilities take care of dependency resolution for rpm systems, **apt**, **apt-cache** and other utilities take care of it for dpkg systems.

In course, only discuss command line interface to each packaging systems. Graphical frontends used by each Linux distribution *can* be useful, would like to be less tied to any one interface, have more flexibility.

5.9 Package Sources

Every distribution -> one or more package **repositories** where system utilities go to obtain software, update with new versions. Job of distribution to make sure all packages in repositories play well with each other.

Always other external repositories which can be added to standard distribution-supported list. Sometimes, closely associated with distribution, only rarely produce significant problems (eg. **EPEL** (**E**xtra **P**ackages for **E**nterprise **L**inux), set of version-dependent repositories, fit well with RHEL since source = Fedora + maintainers close to Red Hat)

However, some external repositories not very well constructed or maintained. Eg. when package is updated in main repository, dependent packages may not be updated in external one, can lead to one form of dependency hell.

5.10 Creating Software Packages

Building own custom software packages -> easy to distribute/install own software. Almost every version of Linux has some mechanism for doing this.

Building own package -> allows to control exactly what goes in the software + exactly how it is installed. Can create package so installing it runs scripts that perform all tasks needed to install new software and/or remove old software:

- Creating needed symbolic links
- Creating directories as needed
- Setting permissions
- Anything that can be scripted

No discussion of mechanisms of how to build **.rpm** or **.deb** packages, as that question mostly for developers, rather than administrators.

5.11 Revision Control System

Software projects become more complex to manage as either size of project increases, or number of contributing developers increases.

To organize updates/facilitate cooperation, many different schemes available for source control. Standard features of such programs: ability to keep accurate history (log) of changes, ability to back up to earlier releases, coordinate possibly conflicting updates from more than one developer, etc.

Source Control Systems (or Revision Control Systems, as also commonly called) fill role of coordinating cooperative development.

5.12 Available Source Control Systems

No shortage of available products, both proprietary/open. Brief list of products released under **GPL** license includes:

Product	URL
RCS	https://www.gnu.org/software/rcs
CVS	https://www.nongnu.org/cvs
Subversion	https://subversion.apache.org
git	https://www.kernel.org/pub/software/scm/git
GNU Arch	https://www.gnu.org/software/gnu_arch
Monotone	https://www.monotone.ca
Mercurial	https://mercurial-scm.org

Will only focus on **git**, widely used product which arose from Linux kernel development community. **git** risen to dominant position in use for open source projects in remarkable short time, often used even in closed source environments.

5.13 The Linux Kernel and the Birth of git

Linux kernel development system has special needs -> widely distributed throughout world, with literally thousands of developers involved. All done very publicly, under GPL license.

For long time, no real source revision control system. Then, major kernel developers went over to use of BitKeeper, commercial project which granted restricted use license for Linux kernel development.

However, in very public dispute over licensing restrictions in spring of 2005, free use of BitKeeper became unavailable for Linux

kernel development.

Response: development of git, original author Linus Torvalds. Source code for git here, full documentation here.

5.14 How git Works

Technically, git not source control management system in usual sense. Basic units it works with are not files. Has two important data structures: object database + directory cache.

Object database contains three objects:

- Blobs: chunks of binary data containing file contents
- Trees: sets of blobs including file names/attributes, giving directory structure
- Commits: changesets describing tree snapshots

Directory cache captures state of directory tree.

By liberating controls system from file-by-file-based system, better able to handle changesets involving many files.

git always under rapid development,graphical interfaces also under speedy construction. See kernel.org git repositories webpage. Can easily browse particular changes, source trees.

Sites (eg. Github) now host literally millions of git repositories, both public/private. Host of easy-to-find articles, books, online tutorials etc., on how to to profitably use git.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 6 RPM - Notes

6.2 Introduction

Red Hat Package Manager (**RPM**) used by number of major distributions (and close relatives) to control installation, verification, upgrade, removal of software on Linux systems. Low-level **rpm** program performs all these operations, either on just one package or on list of packages. Operations that would cause problems (eg. removing package that another package depends on, or installing package when system needs other software to be installed first) blocked from completion.

6.3 Learning Objectives:

- Understand how the **RPM** system is organized and what major operations the **rpm** program can accomplish.
- Explain the naming conventions used for both binary and source **rpm** files.
- Query, verify, install, uninstall, upgrade, and freshen packages.
- Grasp why new kernels should be installed, rather than upgraded.
- Use **rpm2cpio** to copy packaged files into a **cpio** archive, as well as to extract the files without installing them.

6.4 RPM

RPM (**R**ed Hat **P**ackage **M**anager) developed by Red Hat (unsurprisingly). All files related to specific task packaged into single **rpm** file, which also contains information about how/where to install/uninstall files. New versions of software lead to new **rpm** files, which then used for updating.

rpm files also contain dependency information. Note: unless given specific **URL** to draw from, **rpm** does not retrieve packages over network, installs only from local machine using absolute/relative paths.

rpm files usually distribution-dependent; installing package on different distribution than it was created for -> difficult, if not impossible.

6.5 Advantages of Using RPM

For system administrators, RPM makes it easy to:

- Determine what package (if any) any file on system is part of
- Determine what version installed
- Install/uninstall (erase) packages without leaving debris behind
- Verify that package was installed correctly. Useful for both troubleshooting/system auditing
- Distinguish documentation files from rest of package, optionally decide not to install them to save space
- Use **ftp** or **HTTP** to install packages over internet

For developers, RPM also offers advantages:

- Software often made available on more than one operating system. With RPM, original full + unmodified source used as basis, but developer can separate out changes needed to build on Linux
- More than one architecture can be built using only one **source package**

6.6 Package File Names

RPM package file names based on fields representing specific information, as documented in RPM standard:

- Standard naming format for binary package:

  `<name>-<version>-<release>.<distro>.<architecture>.rpm`

  `sed-4.2.2-5.el7.x86_64.rpm`

- Standard naming format for source package:

  `<name>-<version>-<release>.<distro>.src.rpm`

  `sed-4.2.2-5.el7.src.rpm`

Note: **distro** field often actually specifies repository that package came from, as given installation may use number of different package repositories, as discussed for **yum**/**zypper** which work above RPM.

6.7 Database Directory

`/var/lib/rpm` -> default system directory that holds RPM database files in form of Berkeley DB hash files. Database files should not be manually modified, updated should be done only through use of **rpm** program.

Alternative database directory can be specified with `--dbpath` option to **rpm** program. Might do this, for example, to examine RPM database copied from another system.

Can use `--rebuilddb` option to rebuild database indices from installed package headers. More of repair, not rebuild from scratch.

6.8 Helper Programs and Modifying Settings

Helper programs and scripts used by RPM reside in `/usr/lib/rpm`. Quite a few, eg. on RHEL 7 system:

```
$ ls /usr/lib/rpm | wc -l
73
```

where **wc** repoting number of lines of output.

Can create `rpmrc` file to specify default settings for **rpm**. By default, **rpm** looks for:

1. `/usr/lib/rpm/rpmrc`
2. `/etc/rpmrc`
3. `~/.rpmrc`

in above order. Note: all these files read; **rpm** does not stop as soon as it finds that one exists. Alternative `rpmrc` file can be specified usinr `--rcfile` option.

6.9 Queries

All **rpm** inquiries include `-q` option, which can be combined with numerous sub-options:

- Which version of a package is installed?

```
$ rpm -q bash
```

- Which package did this file come from?

```
$ rpm -qf /bin/bash
```

- What files were installed by this package?

```
$ rpm -ql bash
```

- Show information about this package.

```
$ rpm -qi bash
```

- Show information about this package from the package file, not the package database.

```
$ rpm -qip foo-1.0.0-1.noarch.rpm
```

- List all installed packages on this system.

```
$ rpm -qa
```

Couple of other useful options: `--require`, `--whatprovides`:

- Return a list of prerequisites for a package:

```
$ rpm -qp --requires foo-1.0.0-1.noarch.rpm
```

- Show what installed package provides a particular requisite package:

```
$ rpm -q --whatprovides libc.so.6
```

6.10 Verifying Packages

`-V` option to **rpm** allows you to verify whether files from particular package consistent with system's RPM database. To verify all packages installed on system:

```
$ rpm -Va
missing /var/run/pluto
....
S.5....T. c /etc/hba.conf
S.5....T. /usr/share/applications/defaults.list
....L.... c /etc/pam.d/fingerprint-auth
....L.... c /etc/pam.d/password-auth
....
.M....... /var/lib/nfs/rpv-pipefs
....
.....UG.. /usr/local/bin
.....UG.. /usr/local/etc
```

showing just few items. (Note: command can take long time, as it examines all files owned by all packages.)

Output generated only when there is problem.

Each characters displayed denotes result of comparison of attribute(s) of file to value of those attribute(s) recorded in database. Single . (period) means test passed, while single ? (question mark) indicates test could not be performed (eg. file permissions prevent reading). Otherwise, character denotes failure of corresponding verification test:

- **S**: file size differs
- **M**: file permissions and/or type differs
- **5**: MD5 checksum differs
- **D**: device major/minor number mismatch
- **L**: symbolic link path mismatch
- **U**: user ownership differs
- **G**: group ownership differs
- **T**: modification time differs
- **P**: capabilities differ

Note: many verification tests do not indicate problem. Eg. many configuration files modified as system evolves.

If specifying one or more package names as argument, examine only that package:

- No output when everything is OK:

```
$ rpm -V bash
```

- Output indicating that a file's size, checksum, and modification time have changed:

```
$ rpm -V talk
S.5....T in.ntalkd.8
```

- Output indicating that a file is missing:

```
$ rpm -V talk
missing /usr/bin/talk
```

6.11 Installing Packages

Installing package as simple as:

```
$ sudo rpm -ivh foo-1.0.0-1.noarch.rpm
```

where `-i` for install, `-v` for verbose, `-h` to print hash marks to show progress.

RPM performs number of tasks when installing package:

- Performs dependency checks: necessary because some packages will not operate properly unless one or more other packages also installed.
- Performs conflict checks: include attempts to install already-installed package or to install older version over newer version.

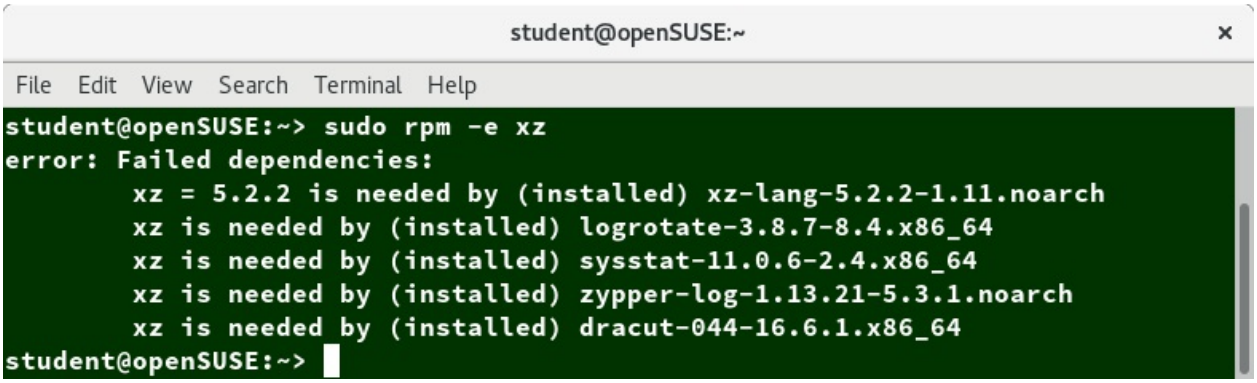
- Executes commands required before installation: developer building package can specify that certain tasks be performed before or after install.
- Deals intelligently with configuration files: when installing configuration file, if file exists and has been changed since previous version of package was installed, RPM saves old version with `suffix .rpmsave`. Allows you to integrate changes made to the configuration file into new version of file. Feature depends on properly created RPM packages.
- Unpacks files from packages, installs them with correct attributes: in addition to installing files in right place, RPM also sets attributes such as permissions, ownership, modification (build) time.
- Executes commands required after installation: performs any post-install tasks required for setup or initialization.
- Updates system RPM database: every time RPM install package, updates information in system database. Uses information when checking for conflicts.

6.12 Uninstalling Packages

`-e` option causes **rpm** to uninstall (erase) package. Normally, `rpm -e` fails with error message if package attempting to install either not actually installed, or required by other packages on system. Successful uninstall produces no output.

```
$ sudo rpm -e system-config-lvm
package system-config-lvm is not installed
```

Example of error due to dependencies:



Can use `--test` option alone with `-e` to determine whether uninstall would succeed or fail, without actually doing uninstall. If operation successful, **rpm** prints no output. Add `vv` option to get more information.

Remember: package argument for erase is package name, not **rpm** file name.

Important (but obvious) note: Never remove (erase/uninstall) **rpm** package itself. Only way to fix this problem to re-install operating system, or booting into rescue environment.

6.14 Upgrading Packages

Upgrading replaces original package (if installed):

```
$ sudo rpm -Uvh bash-4.2.46-30.el7_0.4.x86_64.rpm
```

Can give list of package names, not just one.

When upgrading, already installed package removed after newer version installed. One exception: configuration files from original installation -> kept with `.rpmsave` extension.

If `-U` option used and package not already installed, simply installed and no error.

`-i` option not designed for upgrades. Attempting to install new RPM package over older one fails with error messages, because it tries to overwrite existing system files.

Different versions of same package may be installed if each version of package does not contain same files: kernel packages and library packages from alternative architectures typically only packages that would be commonly installed multiple times.

If want to downgrade with `rpm -U` (to replace current version with earlier version), must add `--oldpackage` option to command line.

6.15 Freshening Packages

```
$ sudo rpm -Fvh *.rpm
```

will attempt to **freshen** all packages in current directory:

1. If older version of package installed, will be upgraded to newer version in directory
2. If version on system is same as one in directory, nothing happens
3. If no version of package installed, package in directory ignored

Freshening useful for applying lot of patches (ie, upgraded packages) at once.

6.16 Upgrading the Kernel

When installing new kernel on system, requires reboot (one of few updates that do) to take effect. Should not do upgrade (`-U`) of kernel: upgrade would remove old currently running kernel.

This in and of itself won't stop system, but if, after reboot, have any problems, will no longer be able to reboot into old kernel, since removed from system. However, if install (`-i`), both kernels coexist and can choose to boot into either one, ie. can revert back to old one if need be.

To install new kernel:

```
$ sudo rpm -ivh kernel-{version}.{arch}.rpm
```

filling in correct version + architecture names.

When doing this, GRUB configuration file automatically updated to include new version. Will be default choice at boot, unless reconfigure system to do something else.

One new kernel version tested, may remove old version if you wish, though not necessary. Unless short on space, recommended to keep one or more older kernels available.

6.17 Using rpm2cpio

Suppose have need to extract files from **rpm** but do not want to actually install package?

rpm2cpio program used to copy files from **rpm** to **cpio** archive + extract files if desired.

Create **cpio** archive with:

```
$ rpm2cpio foobar.rpm > foobar.cpio
```

To list files in **rpm**:

```
$ rpm2cpio foobar.rpm | cpio -t
```

but better w ay is to:

```
$ rpm -qilp foobar.rpm
```

To extract on system:

```
$ rpm2cpio bash-XXXX.rpm | cpio -ivd bin/bash
$ rpm2cpio foobar.rpm | cpio --extract --make-directories
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 7 DPKG - Notes

7.2 Introduction

Debian Package Manager (**DPKG**) used by all Debian-based distributions to control installation, verification, upgrade, removal of softw are on Linux systems. Low -level **dpkg** program can perform all operations, either on just one package, or on list of packages. Operations w hich cause problems (eg. removing package that another package depends on, installing package w hen system needs other softw are to be installed first) blocked from completion.

7.3 Learning Objectives:

- Discuss the DPKG packaging system and its uses.
- Explain the naming conventions used for both binary and source **deb** files.
- Know w hat source packages look like.
- Use querying and verifying operations on packages.
- Install, upgrade, and uninstall Debian packages.

7.4 DPKG Essentials

DPKG (**Debian Package**): packaging system used to install, remove, manage softw are packages under Debian Linux and other derived distributions. Like RPM, not designed to directly retrieve packages in day-today use, but to install and remove them locally.

Package files have `.deb` suffix and DPKG database resides in `/var/lib/dpkg` directory.

like **rpm**, **dpkg** program has only partial view of universe: know s only w hat is installed on system, w hatever given on command line, know s nothing of other available packages (w hether they are in other directory on system or out on Internet). Will also fail if dependency not met, or if trying to remove package other installed packages need.

7.5 Package File Names

Debian package file names based on fields that represent specific information. Standard naming format for binary package:

```
<name>_<version>-<revision_number>_<architecture>.deb
```

as in:

```
logrotate_3.8.7-1_amd64.deb
```

on Debian, and

```
logrotate_3.8.7-1ubuntu1_amd64.deb
```

on Ubuntu.

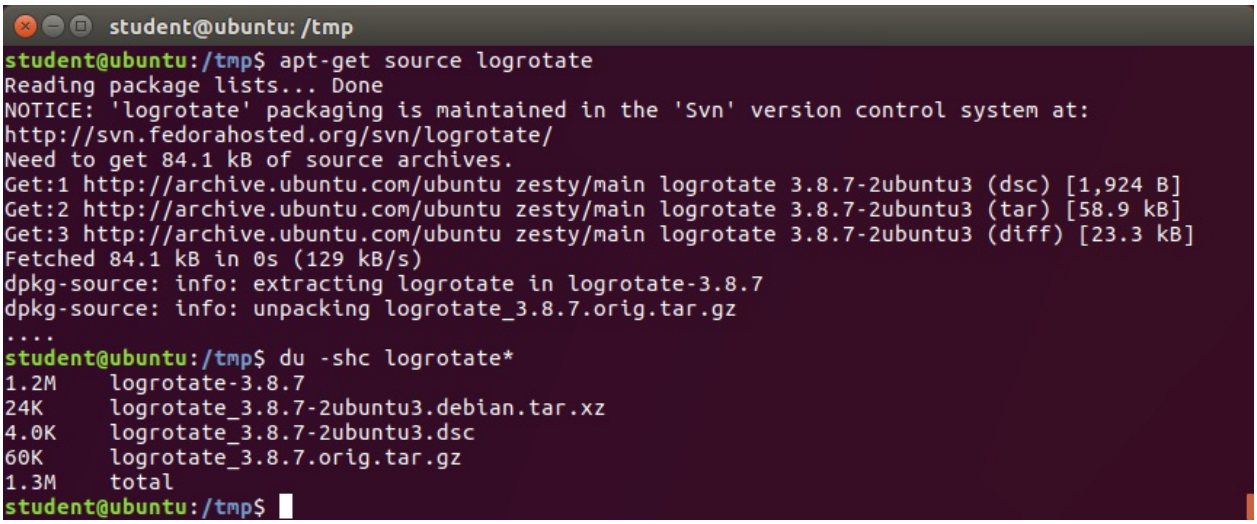
Note: for historical reasons, 64-bit x86 platform called amd64 rather than x86-64, and distributors (eg. Ubuntu) insert their name in package name.

7.6 Source Packages

In Debian packaging system, source package consists of at least three files:

1. Upstream tarball, ending w ith `.tar.gz`. Unmodified source as it comes from package maintainers
2. Description file, ending w ith `.dsc`, containing package name and other metadata, eg. architecture, dependencies
3. Second tarball containing any patches to upstream source + additional files created for package. Ends w ith name `.debian.tar.gz` or `.diff.gz` depending on distribution

Eg. on Ubuntu system, can dow nload source package, then see w hat files dow nloaded or created:



7.7 DPKG Queries

Some examples of queriesL

- List all packages installed:

```
$ dpkg -l
```

One can also specify a package name.

- List files installed in `wget` package:

```
$ dpkg -L wget
```

- Show info about an installed package:

```
$ dpkg -s wget
```

- Show info about a package file:

```
$ dpkg -I webfs_1.21+ds1-8_amd64.deb
```

- List files in a package file:

```
$ dpkg -c webfs_1.21+ds1-8_amd64.deb
```

- Show what package owns `/etc/init/networking.conf`:

```
$ dpkg -S /etc/init/networking.conf
```

- Show the status of a package:

```
$ dpkg -s wget
```

- Verify the installed package's integrity:

```
$ dpkg -V package
```

Without arguments, this will verify all packages on system. See **man** page to interpret output. Note: only recent versions of **dpkg** (1.17+) support this option.

7.8 Installing/Upgrading/Uninstalling Packages

```
$ sudo dpkg -i foobar.deb
```

would be used for either installing/upgrading **foobar** package.

If package not currently installed, then will be installed. If package newer than one currently installed, then will be upgraded.

```
$ sudo dpkg -r package
```

used to remove all of installed package except for configuration files.

```
$ sudo dpkg -P package
```

used to remove all of installed package including configuration files (Note: `-P` stands for **purge**).

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 8 YUM - Notes

8.2 Introduction

yum program provides higher level of intelligent services for using underlying **rpm** program. Can automatically resolve dependencies when installing, updating, removing packages. Accesses external software **repositories**, synchronizing with them, retrieving/installing software as needed.

8.3 Learning Objectives:

- Discuss package installers and their characteristics.
- Explain how **yum** works as a high level package management system.
- Configure **yum** to use repositories.
- Discuss the queries **yum** can be used for.
- Verify, install, remove, and upgrade packages using **yum**.
- Learn about additional commands and how to install new repositories.
- Understand how to use **dnf**, which has replaced **yum** on Fedora.

8.4 Package Installers

Lower-level package utilities (eg. **rpm**, **dpkg**) deal with details of installing specific software package files. managing already installed software.

Higher-level package **management systems** (eg. **yum**, **dnf**, **apt**, **zypper**) work with databases of available software, incorporate tools needed to find, install, update, uninstall software in highly intelligent fashion.

- Can use both local/remote repositories as source to install/update binary as well as source software packages
- Used to automate install, upgrade, removal of software packages
- Resolve dependencies automatically
- Save time because no need to either download packages manually/search out dependency information separately

Software repositories provided by distributions/other independent software providers. Package installers maintain databases of available software derived from catalogs kept by repositories. Unlike low-level package tools, have ability to find/install dependencies automatically -> critical feature.

In this section, discuss **yum** and **dnf**. **zypper** and **apt** discussed in later chapters.

8.5 What is yum?

yum provides frontend to **rpm**. Primary task: fetch packages from multiple remote repositories, resolve dependencies among packages. Used by majority (but not all) of distributions using **rpm**, including RHEL, CentOS, Scientific Linux, Fedora.

yum caches information + databases to speed up performance. To remove some or all cached information, can run command:

```
$ yum clean [ packages | metadata | expire-cache | rpmdb | plugins | all ]
```

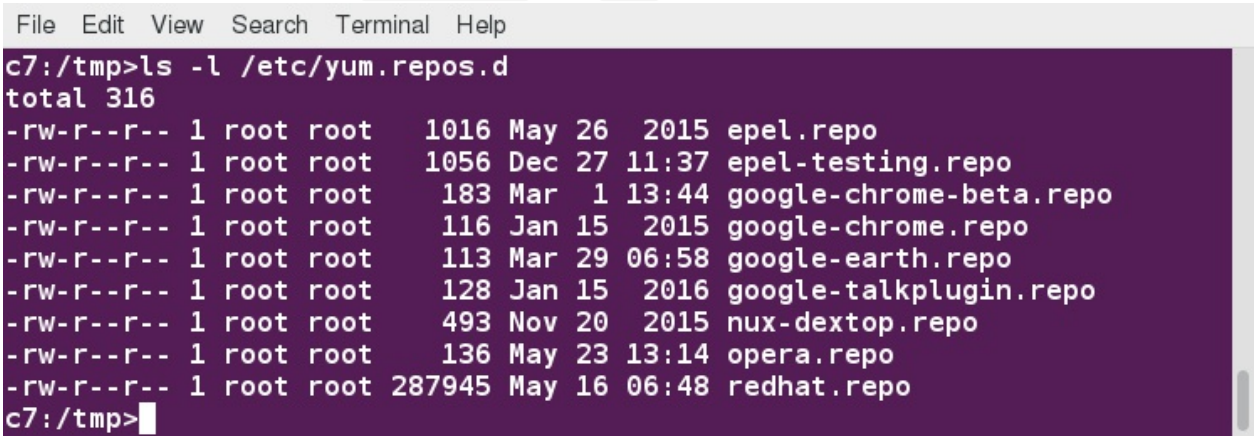
yum has number of modular expressions (plugins) + companion programs that can be found under `/usr/bin/yum*` and

`/usr/sbin/yum*`.

Will concentrate on command line use of **yum**, not consider graphical interfaces distributions provide.

8.6 Configuring yum to Use Repositories

Repository configuration files kept in `/etc/yum.repos.d`, have `.repo` extension. Eg. on one RHEL 7 system:



Note: on RHEL 6 there is no `redhat.repo` file. RHEL 6 + earlier versions handled distribution-supplied repos in somew hat different manner, although RHEL clones like CentOS used conventional repos for main distribution packages.

8.7 Repository Files

Very simple repository file may look like:

```
[repo-name]
    name=Description of the repository
    baseurl=http://somesystem.com/path/to/repo
    enabled=1
```

More complicated examples found in `/etc/yum.repos.d`, w ould be good idea to examine them.

Can toggle the use of particular repository on/off by changing value of enabled to 1/0, or using `--disablerepo=somerepo` and `--enablerepo=somerepo` options w hen using **yum**.

Can (but should not) also turn off integrity checking w ith `gpgcheck` variable.

8.8 Queries

Like **rpm**, **yum** can be used for queries such as searches. How ever, can search not just w hat is present on local system, but also inquire about remote repositories. Examples:

- Search for packages w ith `keyword` in name:

```
$ sudo yum search keyword
$ sudo yum list "*keyword*"
```

These tw o commands give somew hat different information. First one tells more about packages, second one makes it clearer

what is installed, what else is available.

- Display information about a package:

```
$ sudo yum info package
```

Information includes size, version, what repository it came from, source URL, longer description. Wildcards can be given, eg. `yum info "libc*"` for this + most **yum** commands. Note: package need not be installed, unlike queries make with `rpm -q`.

More **yum** examples:

- List all packages, or just those installed, available, or updates that have not yet been installed:

```
$ sudo yum list [ installed | updates | available ]
```

- Show information about package groups installed or available, etc.:

```
$ sudo yum grouplist [group1] [group2]
$ sudo yum groupinfo group1 [group2]
```

- Show packages that contain a certain file name:

```
$ sudo yum provides
```

as in

```
$ sudo yum provides "/logrotate.conf"
```

Note need to use at least one `/` in file name, which can be confusing.

8.9 Verifying Packages

Package verification requires installation of **yum-plugin-verify** package. Might have to do:

```
$ sudo yum install yum-plugin-verify
```

Note: this is **yum plugin**, not executable. Many other plugins available for **yum**, extends possible set of commands and arguments it can take:

- To verify package, giving most information:

```
$ sudo yum verify [package]
```

- To mimic `rpm -V` exactly:

```
$ sudo yum verify-rpm [package]
```

- To list all differences, including configuration files:

```
$ sudo yum verify-all [package]
```

Without arguments, above commands will verify all packages installed on system.

By default, verification commands ignore configuration files which may change through normal + safe usage. Some other options: see **man yum-verify**.

8.10 Installing/Removing/Upgrading Packages

Some examples of commonly performed operations:

- Install one or more packages from repositories, resolving/installing any necessary dependencies:

```
$ sudo yum install package1 [package2]
```

- Install from local **rpm**:

```
$ sudo yum localinstall package-file
```

This is not quite the same as

```
$ rpm -i package-file
```

because it will attempt to resolve dependencies by accessing remote repositories.

- Install specific software **group** from repository, resolving/installing any necessary dependencies for each package in group:

```
$ sudo yum groupinstall group-name
```

or

```
$ sudo yum install @group-name
```

- Remove packages from system:

```
$ sudo yum remove package1 [package2]
```

Must be careful with package removal, as **yum** will not only remove requested packages, but all packages that depend on them! May not be what you want, so never run `yum remove` with `-y` option, which assumes automatic confirmation of removal.

- Update package from repository:

```
$ sudo yum update [package]
```

If not package name given, all packages updated.

During installation (or update), if package has configuration file which is updated, will rename old configuration file with `.rpmsave` extension. If old configuration file will still work with new software, will name new configuration file with `.rpmnew` extension. Can search for these filename extensions (almost always in `/etc` subdirectory tree) to see if you need to do any reconciliation, by doing:

```
$ sudo find /etc -name "*.rpm*"
```

Same behavior the more naked underlying **rpm** utility exhibits, but mentioned here for reference.

8.11 Additional Commands

No shortage of additional capabilities for **yum**, according to what plugins are installed. Can list them all with:

```
$ sudo yum list "yum-plugin"
```

In particular:

- Show list of all enabled repositories:

```
$ sudo yum repolist
```

- Initiate interactive shell in which to run multiple `YUM` commands:

```
$ sudo yum shell [text-file]
```

If `text-file` given, **yum** will read + execute commands from that file instead of from terminal.

More examples of **yum** commands:

- Download package, but do not install them; just store them under the `/var/cache/yum` directory, or another directory specified:

```
$ sudo yum install --downloadonly package
```

or can type `"d"` instead of `"y"` or `"n"` when prompted after issuing install command. Package(s) will be downloaded under `/var/cache/yum` in location depending on repository from which download proceeds, unless `--downloaddir=` option used. Any other necessary packages will also be downloaded to satisfy dependencies.

- Can view history of **yum** commands, and, with correct options, even undo/redo previous commands:

```
$ sudo yum history
```

8.12 dnf

dnf intended to be next generation replacement for **yum**, will underlie **yum** in RHEL 8.

Can gradually learn to use **dnf** on Fedora systems because it accepts subset of **yum** commands that take care of majority of day-to-day tasks + points out at each use of **yum** that has **dnf** equivalent.

To learn more, see: Package Management section in the Fedora System Administrator's Guide.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 9 zypper - Notes

9.2 Introduction

For use on SUSE-based systems, **zypper** program provides higher level of intelligent services for using underlying **rpm** program, plays same role as **yum** on Red Hat-based systems. Can automatically resolve dependencies when installing, updating, removing packages. Accesses external software repositories, synchronizing with them, retrieving/installing software as needed.

9.3 Learning Objectives:

- Explain what **zypper** is.
- Discuss the queries **zypper** can be used for.
- Install, remove, and upgrade packages using **zypper**.
- Learn additional and more advanced **zypper** commands.

9.4 What is zypper?

zypper: command line tool for installing/managing packages in SUSE Linux and openSUSE. Very similar to **yum** in functionality, even in basic command syntax, also works with **rpm** packages.

Retrieves packages from repositories, installs, removes, updates while resolving any dependencies encountered. Equivalent in practice to **yum** and **apt-get** in that it can retrieve packages from repository and also resolve dependencies.

9.5 zypper Queries

Some examples of commonly performed operations involving querying:

- Show a list of available updates:

```
$ zypper list-updates
```

- List available repositories:

```
$ zypper repos
```

- Search repositories for `string`:

```
$ zypper search <string>
```

- List information about package:

```
$ zypper info <package>
```

- Search repositories to ascertain what packages provide a file:

```
$ zypper search --provides <file>
```

- Find which package provides file specified as argument:

```
$ zypper what-provides <file-path>
```

9.6 Installing/Removing/Upgrading

Some examples of commonly performed operations:

- Install or update package(s):

```
$ sudo zypper install package
```

- Do not ask for confirmation when installing or upgrading:

```
$ sudo zypper --non-interactive install <package>
```

This is useful for scripts and is equivalent to running `yum -y`.

- Update all installed packages:

```
$ sudo zypper update
```

Giving package names as an argument will update only those packages and any required dependencies. Do this without asking for confirmation:

```
$ sudo zypper --non-interactive update
```

- Remove a package from the system:

```
$ sudo zypper remove <package>
```

Like with **yum**, have to be careful with removal command, as any package that needs the package being removed would be removed as well.

9.7 Additional zypper Commands

Sometimes, number of **zypper** commands must be run in sequence. To avoid re-reading all databases for each command, can run **zypper** in **shell mode**:

```
$ sudo zupper shell
```

```
> install bash
...
> exit
```

Because **zypper** supports **readline** library, can use all the same command line editing function available in bash shell in zypper shell.

To add new repository:

```
$ sudo zypper addrepo URI alias
```

which is located at the supplied `URI` and will use supplied `alias`.

To remove repository from list:

```
$ sudo zypper removerepo alias
```

using `alias` of repo you want to delete.

To clean up and save space in `/var/cache/zypp`:

```
$ sudo zypper clean [--all]
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 10 APT - Notes

10.2 Introduction

For use on Debian-based systems, **APT** (**A**dvanced **P**ackaging **T**ool) set of programs provides higher level of intelligent services for using underlying **dpkg** program, plays same role as **yum** on Red Hat-based systems. Main utilities: **apt-get**, **apt-cache**. Can automatically resolve dependencies when installing, updating, removing packages. Accesses external software repositories, synchronizing with them, retrieving/installing software as needed.

10.3 Learning Objectives:

- Explain what APT is.
- Perform package queries.
- Clean up system information about packages.
- Install, remove, and upgrade packages using **apt-get**.

10.4 What is APT?

APT not a program in itself: stands for **A**dvanced **P**ackaging **T**ool, which includes number of utilities, eg. **apt-get**, **apt-cache**. These, of course, in turn, invoke lower-level **dpkg** program.

APT system works with Debian packages whose files have `.deb` extension. Many distributions that have descended from Debian (eg. Ubuntu, Linux Mint) which have adopted Debian packaging system with no essential modification. Not uncommon to use repository on more than one Debian-based Linux distribution.

Going to ignore graphical interfaces (on computer), eg. Synaptic, Ubuntu Software Center, or other older frontends to APT, eg. aptitude.

However, excellent Internet-based resources can be found on Debian packages webpage and Ubuntu packages webpage. These databases let you search for packages, examine their contents, and download.

10.5 apt-get

apt-get: main APT command line tool for package management. Can be used to install, manage, upgrade individual packages, or the entire system. Can even upgrade distribution to completely new release, which can be difficult task.

There are even (imperfect) extensions that let **apt-get** work with **rpm** files.

Like **yum** and **zypper**, works with multiple remote repositories.

10.6 Queries

- Search repository for package named `apache2`:

```
$ apt-cache search apache2
```

- Display basic information about `apache2` package:

```
$ apt-cache show apache2
```

- Display more detailed information about `apache2` package:

```
$ apt-cache showpkg apache2
```

- List all dependent packages for `apache2`:

```
$ apt-cache depends apache2
```

- Search repository for file named `apache2.conf`:

```
$ apt-file search apache2.conf
```

- List all files in `apache2` package:

```
$ apt-file list apache2
```

10.7 Cleaning Up

To get rid of any packages that are not needed anymore, such as older Linux kernel versions:

```
$ sudo apt-get autoremove
```

To clean out cache files and any archived package files that have been installed:

```
$ sudo apt-get clean
```

Over time, may be some packages that are no longer needed at all. Can use `autoremove` to take them off system.

Furthermore, can reduce amount of data stored in `/var/cache/apt*` without causing any harm by using `clean` as any `apt-get [update|upgrade]` command will bring in most current information back into system.

10.8 Installing/Removing/Upgrading

apt-get program work horse of installing, removing, upgrading packages:

- Synchronize package index files with their repository sources. Indexes of available packages fetched from location(s) specified in `/etc/apt/sources.list`:

```
$ sudo apt-get update
```

- Install new packages or update an already installed package:

```
$ sudo apt-get install [package]
```

- Remove package from system without removing its configuration files:

```
$ sudo apt-get remove [package]
```

- Remove package from system, as well as its configuration files:

```
$ sudo apt-get --purge remove [package]
```

- Apply all available updates to packages already installed:

```
$ sudo apt-get upgrade
```

- Do a **smart upgrade** that will do a more thorough dependency resolution and remove some obsolete packages and install new dependencies:

```
$ sudo apt-get dist-ugprade
```

This will not update to whole new version of Linus distribution, as is commonly misunderstood.

- Note: must **update** before **upgrade**, unlike with **yum**, where `update` argument does both steps (update repositories and then upgrades packages). Can be confusing to habitual **yum** users on Debian-based systems.

- Get rid of any packages not needed anymore, such as older Linux kernel versions:

```
$ sudo apt-get autoremove
```

- Clean out cache files and any archived package files that have been installed:

```
$ sudo apt-get clean
```

This can save lot of space.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 11 System Monitoring - Notes

11.3 Learning Objectives:

- Understand the concept of inventory and gain familiarity with available system monitoring tools.
- Understand where the system stores log files and examine the most important ones.
- Use the `/proc` and `/sys` pseudo-filesystems.
- Use **sar** to father system activity and performance data and create reports that are readable by humans.

11.4 Available Monitoring Tools

Linux distributions comes with many standard performance/profiling tools already installed. Many familiar from other UNIX-like operating systems, while some developed specifically for Linux.

Most tools make use of mounted **pseudo-filesystems**, especially `/proc` and `/sys`, both which have already been discussed while examining filesystems/kernel configuration. Will look at both.

Also a number of graphical system monitors that hide many details, but will only consider command line tools in course.

Before considering main utilities in detail, can see summary on next few sections, broken down by type: note: some utilities have overlapping domains of coverage. Will revisit tables in following chapters that focus on specific topics.

Summary of main process monitoring utility tools:

Process and Load Monitoring Utilities

Utility	Purpose	Package
top	Process activity, dynamically updated	**procps**
uptime	How long system is running and average load	**procps**
ps	Detailed information about processes	**procps**
pstree	Tree of processes and their connections	**psmisc** (or **pstree**)
mpstat	Multiple processor usage	**sysstat**
iostat	CPU utilization and I/O statistics	**sysstat**
sar	Display and collect information about system activity	**sysstat**
numstat	Information about **NUMA** (Non-Uniform Memory Architecture)	**numactl**
strace	Information about all system calls a process makes	**strace**

Memory Monitoring Utilities

Utility	Purpose	Package
free	Brief summary of memory usage	**procps**

vmstat	Detailed virtual memory statistics and block I/O, dynamically updated	**procps**
pmap	Process memory map	**procps**

I/O Monitoring Utilities

Utility	Purpose	Package
iostat	CPU utilization and I/O statistics	**sysstat**
sar	Display and collect information about system activity	**sysstat**
vmstat	Detailed virtual memory statistics and block I/O, dynamically updated	**procps**

Network Monitoring Utilities

Utility	Purpose	Package
netstat	Detailed networking statistics	**netstat**
iptraf	Gather information on network interfaces	**iptraf**
tcpdump	Detailed analysis of network packets and traffic	**tcpdump**
wireshark	Detailed network traffic analysis	**wireshark**

11.5 System Log Files

System log files -> essential for monitoring/troubleshooting. In Linux, messages appear in various files under `/var/log`. Exact names vary with Linux distribution.

Ultimate control of how messages dealt with -> controlled by **syslogd** (usually **rsyslogd** on modern systems) daemon, common to many UNIX-like operating systems. Newer **systemd**-based systems can use **journalctl** instead,but usually retain **syslogd** and cooperate with it.

Important messages sent not only to logging files, but also to system console window. If not running **X**, or are at virtual terminal, will see them directly there as well. In addition, messages will be copied to `/var/log` messages (or to `/var/log/syslog` on Ubuntu), but if running **X**, have to take some steps to view them.

Can view new messages continuously as new lines appear with:

```
$ sudo tail -f /var/log/messages (or /var/log/syslog)
```

or

```
$ dmesg -w
```

which shows only kernel-related messages.

11.6 Important Log Files in /var/log

Besides looking at log messages in terminal window, can see them using graphical interfaces.

On GNOME desktop, can also access messages by clicking on `System -> Administration -> System Log` or `Applications -> System Tools -> Log File` Viewer in your Desktop menus, and other desktops have similar links you can locate.

Some important log files found under `/var/log`:

File	Purpose
`boot.log`	System boot messages
`dmesg`	Kernel messages saved after boot. To see current contents of kernel message buffer, type **dmesg**.
`messages` or `syslog`	All important system messages
`secure`	Security related messages

In order to keep log files from growing without bound, **logrotate** program run periodically, keeps four previous copies (by default) of log files (optionally compressed). Controlled by `/etc/logrotate.conf`.

11.7 The /proc and /sys Pseudo-filesystems

`/proc` and `/sys` pseudo-filesystems contain lot of information about system. Many entries in these directory trees writable, can be used to change system behavior. Most cases, requires **root** user.

Pseudo-filesystems because totally exist in memory. If look at disk partition when system not running, there will be only empty directory which is used as mount point.

Information displayed is gathered only when looked at. No constant/periodic polling to update entries.

11.8 /proc Basics

`/proc` pseudo-filesystem: long history. Has roots in other UNIX operating system variants. Originally developed to display information about **processes** on system, each of which has own subdirectory in `/proc` with all important process characteristics available.

Over time, grew to contain lot of information about system properties, eg. interrupts, memory, networking, etc. in somewhat anarchistic way. Still extensively used, will often refer to it.

11.9 A survey of /proc

What resides in `/proc` pseudo-filesystem:

```
student@ubuntu: ~
student@ubuntu:~$ ls -F /proc
1/     13/    200/   2288/  250/   33/    40/    5383/  99/          misc
10/    137/   201/   229/   251/   3315/  4047/  54/    990/         modules
100/   14/    202/   23/    2529/  3336/  41/    5410/  acpi/        mounts@
1005/  1457/  203/   230/   2572/  3353/  4156/  5436/  buddyinfo    mpt/
1007/  1465/  204/   231/   26/    3366/  4169/  5438/  bus/         mtrr
1008/  1478/  205/   232/   27/    3394/  418/   581/   cgroups      net@
1009/  15/    206/   233/   274/   34/    419/   583/   cmdline      pagetypeinfo
101/   16/    207/   234/   2756/  3419/  42/    6/     consoles     partitions
1010/  1649/  208/   235/   276/   3430/  420/   601/   cpuinfo      sched_debug
1011/  1780/  209/   236/   28/    3439/  422/   7/     crypto       schedstat
1012/  1782/  21/    2368/  280/   3441/  43/    715/   devices      scsi/
1013/  1798/  210/   237/   281/   35/    44/    717/   diskstats    self@
1014/  18/    211/   2373/  2979/  3512/  45/    718/   dma          slabinfo
102/   1822/  212/   238/   298/   3513/  4595/  721/   driver/      softirqs
1028/  188/   213/   2385/  2989/  3514/  4596/  723/   execdomains  stat
103/   189/   214/   239/   30/    3515/  4599/  728/   fb           swaps
104/   19/    2142/  24/    300/   3532/  46/    731/   filesystems  sys/
105/   190/   215/   240/   3081/  3534/  4601/  733/   fs/          sysrq-trigger
106/   191/   216/   241/   31/    3569/  47/    735/   interrupts   sysvipc/
107/   1916/  217/   242/   32/    3581/  473/   743/   iomem        thread-self@
108/   192/   218/   243/   3228/  3589/  478/   746/   ioports      timer_list
1083/  1922/  219/   2436/  3243/  36/    4792/  750/   irq/         timer_stats
1085/  193/   2196/  244/   3244/  3600/  483/   753/   kallsyms     tty/
109/   1930/  2197/  2443/  3245/  3613/  485/   755/   kcore        uptime
11/    194/   22/    2449/  3246/  3621/  486/   781/   keys         version
110/   1940/  220/   245/   3247/  3655/  492/   798/   key-users    version_signature
1121/  195/   221/   2456/  3251/  37/    50/    8/     kmsg         vmallocinfo
1134/  196/   222/   246/   3255/  370/   502/   828/   kpagecgroup  vmstat
1140/  197/   223/   2461/  3257/  38/    51/    9/     kpagecount   zoneinfo
1141/  198/   224/   247/   3264/  39/    52/    96/    kpageflags
1147/  1986/  225/   248/   3268/  3910/  53/    98/    loadavg
116/   199/   226/   2488/  3272/  393/   5310/  981/   locks
1180/  2/     227/   249/   3274/  399/   5332/  983/   mdstat
12/    20/    228/   25/    3294/  4/     5358/  988/   meminfo
student@ubuntu:~$
```

First, see there is subdirectory for each process on system, w hether sleeping, running, scheduled out. Looking at random one:

```
student@ubuntu: ~
student@ubuntu:~$ ls -F /proc/3589
attr/        coredump_filter  gid_map     mountinfo   oom_score      schedstat  status
autogroup    cpuset           io          mounts      oom_score_adj  sessionid  syscall
auxv         cwd@             limits      mountstats  pagemap        setgroups  task/
cgroup       environ          loginuid    net/        personality    smaps      timers
clear_refs   exe@             map_files/  ns/         projid_map     stack      timerslack_ns
cmdline      fd/              maps        numa_maps   root@          stat       uid_map
comm         fdinfo/          mem         oom_adj     sched          statm      wchan
student@ubuntu:~$
```

Directory full of information about status of process and resources it is using. For example:

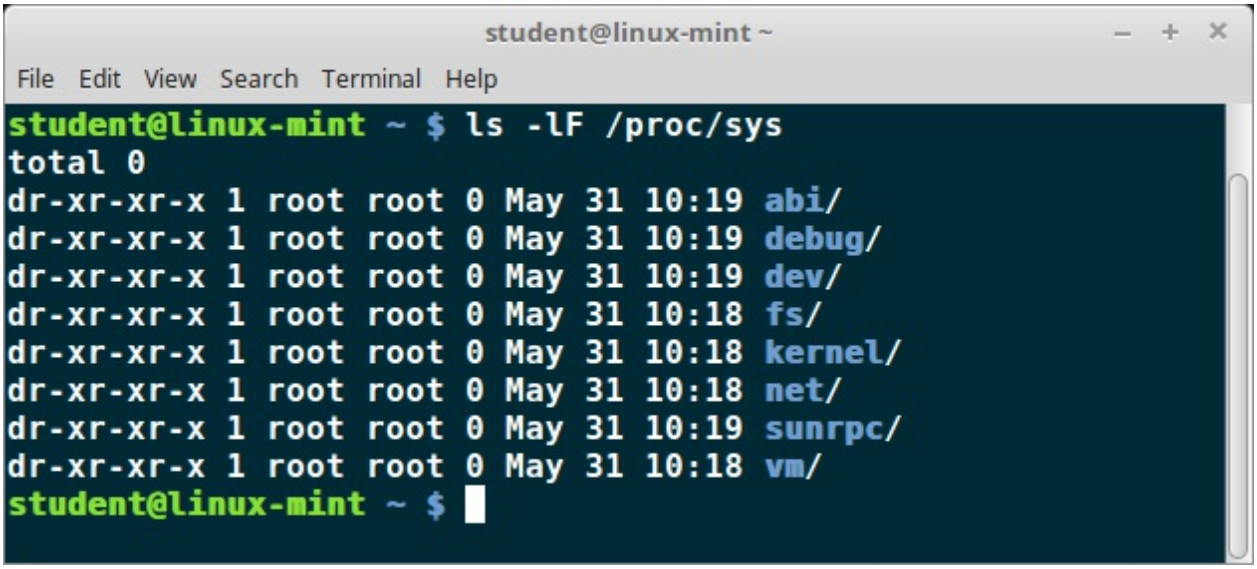
```
student@ubuntu: ~
student@ubuntu:~$ cat /proc/3589/status
Name:   bash
Umask:  0022
State:  S (sleeping)
Tgid:   3589
Ngid:   0
Pid:    3589
PPid:   3581
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
....
Cpus_allowed:   ffffffff,ffffffff,ffffffff,ffffffff
Cpus_allowed_list:      0-127
Mems_allowed:   00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        1328
nonvoluntary_ctxt_switches:     100

student@ubuntu:~$
```

Other entries give system-wide information. Eg. can see **interrupt** statistics in below screenshot. For each interrupt, see what type it is, how many times it has been handled on each CPU, which devices registered to respond to it. Also get global statistics.

```
File  Edit  View  Search  Terminal  Help
x7:/home/coop>cat /proc/interrupts
            CPU0       CPU1       CPU2       CPU3
   0:         88          0          0          0  IR-IO-APIC    2-edge      timer
   1:        566          1       2153          0  IR-IO-APIC    1-edge      i8042
   8:          1          0          0          0  IR-IO-APIC    8-edge      rtc0
   9:      17990         11        552         27  IR-IO-APIC    9-fasteoi   acpi
  12:      64336         21     186157         20  IR-IO-APIC   12-edge      i8042
  16:          0          0          0          0  IR-IO-APIC   16-fasteoi   i801_smbus
 120:          0          0          0          0  DMAR-MSI    0-edge      dmar0
 121:          0          0          0          0  DMAR-MSI    1-edge      dmar1
 122:     217295       1607       4039      59571  IR-PCI-MSI 376832-edge      ahci[0000:00:17.0]
 123:          3          0         46          0  IR-PCI-MSI 514048-edge      snd_hda_intel:card0
 124:      95360         74      49535        192  IR-PCI-MSI 327680-edge      xhci_hcd
 125:     591286          3     272983          2  IR-PCI-MSI 32768-edge      i915
 126:         84         16        149         20  IR-PCI-MSI 520192-edge      enp0s31f6
 127:     225390         29        383         46  IR-PCI-MSI 2097152-edge      iwlwifi
 NMI:         24        120        130        119   Non-maskable interrupts
 LOC:    2255506    2201465    2360863    2239138   Local timer interrupts
 SPU:          0          0          0          0   Spurious interrupts
 PMI:         24        120        130        119   Performance monitoring interrupts
 IWI:          0          0          3          0   IRQ work interrupts
 RTR:         24          3          0          0   APIC ICR read retries
 RES:     185530     146421      95420      45924   Rescheduling interrupts
 CAL:      76456      74989      78143      76063   Function call interrupts
 TLB:      75401      73603      76833      75025   TLB shootdowns
 ERR:          0
 MIS:          0
 PIN:          0          0          0          0   Posted-interrupt notification event
 PIW:          0          0          0          0   Posted-interrupt wakeup event
x7:/home/coop>
```

11.10 /proc/sys

Most tunable system parameters can be found in subdirectory tree rooted at `/proc/sys`:



Each subdirectory contains information + knobs that can be tuned (w ith care):

- `abi/` : Contains files w ith application binary information; rarely used
- `debug/` : Debugging parameters; for now , just some control of exception reporting
- `dev/` : Device parameters, including subdirectories for **cdrom** , **scsi**, **raid**, **parport**
- `fs/` : Filesystem parameters, including quote, files handles used, and maximums, inode and directory information, etc.
- `kernel/` : Kernel parameters. Many important entries here.
- `net/` : Netw ork parameters. Subdirectories for **ipv4**, **netfilter**, etc.
- `vm/` : Virtual memory parameters. Many important entries here.

View ing/changing parameters can be done w ith simple commands. Eg. maximum number of threads allow ed on system seen by looking at:

```
$ ls -l /proc/sys/kernel/threads-max
$ cat /proc/sys/kernel/threads-max
129498
```

Can then modify value, verify change w as effected:

```
$ sudo bash -c 'echo 100000 > /proc/sys/kernel/threads-max'
$ cat /proc/sys/kernel/threads-max
100000
```

Remember from discussion of **sysctl**, same effect accomplished by:

```
$ sudo sysctl kernel.threads-max=100000
```

View ing value can be done as normal user, changing requires superuser privilege.

11.11 /sys Basics

`/sys` pseudo-filesystem: integral part of **Unified Device Model**. Conceptually, based on **device tree**, one can w alk through it and see buses, devices, etc. Also now contains information w hich may or may not be strictly related to devices, such as kernel modules.

Has more tightly defined structure than `/proc`. Most entries contain only one line of text (although there are exceptions) unlike precursor which has many multi-line entries whose exact contents may change between kernel versions. Thus, interface hopefully more stable.

There are system properties which have display entries in both `/proc` and `/sys`. For compatibility with widely used system utilities, older forms only gradually being whittled down.

11.12 A Survey of /sys

Support for **sysfs** virtual filesystem built into all modern kernels, should be mounted under `/sys`. However, unified device model does not require mounting **sysfs** in order to function.

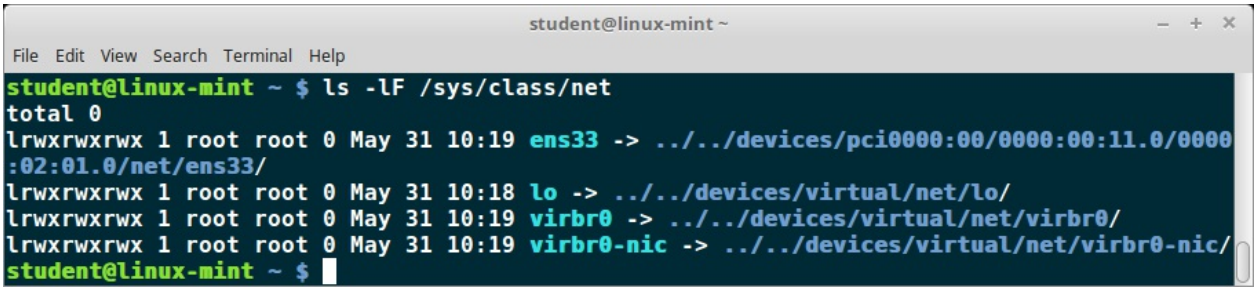
Taking look at 3.18 kernel (warning; exact layout of this filesystem tends to mutate). Top level directory command yields:

```
$ ls -F /sys
block/ bus/ class/ dev/ devices/ firmware/ fs/ kernel/ module/ power/
```

which displays basic device hierarchy. Device model **sysfs** implementation also includes information not strictly related to hardware.

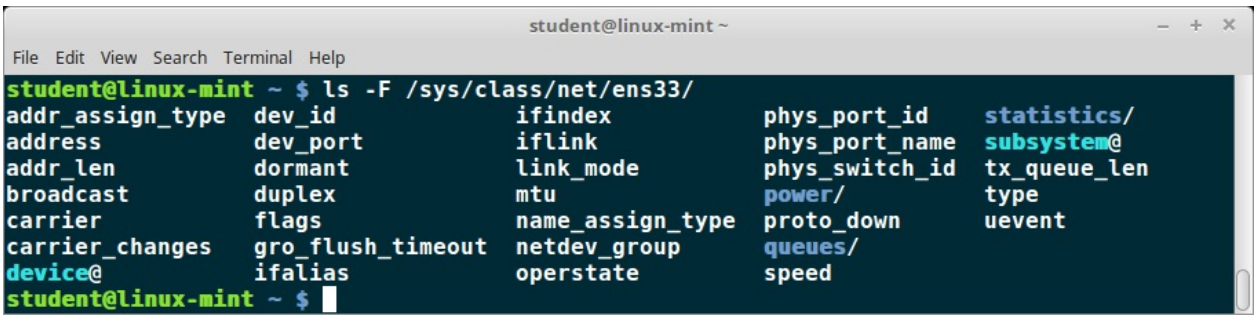
Network devices examined with:

```
$ ls -lF /sys/class/net
```



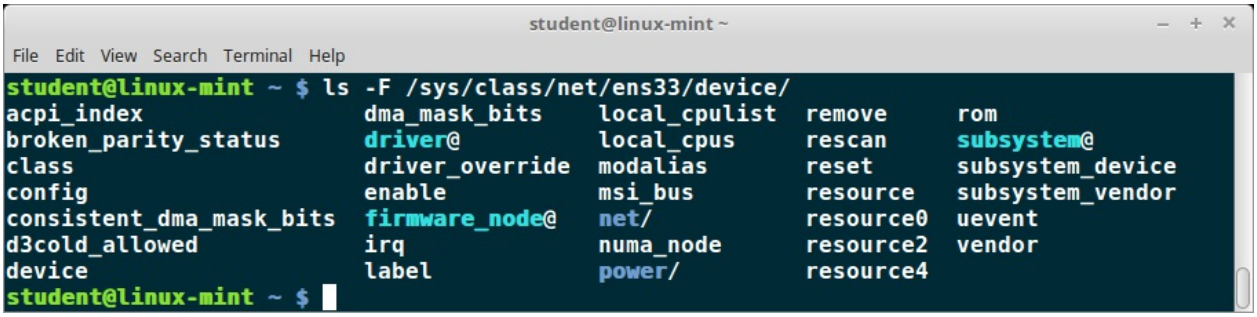
Below, can see what looking at Ethernet card gives.

Intention with **sysfs** to have one text value per line, although not expected to be rigorously enforced.



Underlying device and driver for first network interface can be traced through `device` and (to be seen shortly) `driver` symbolic links. Below shows what can be seen when looking at directory corresponding to first Ethernet card.

To see full spectrum of information available with **sysfs**, will just have to examine.



11.13 sar

sar: **S**ystems **A**ctivity **R**eporter. All-purpose tool for gathering system activity + performance data, creating reports readable by humans.

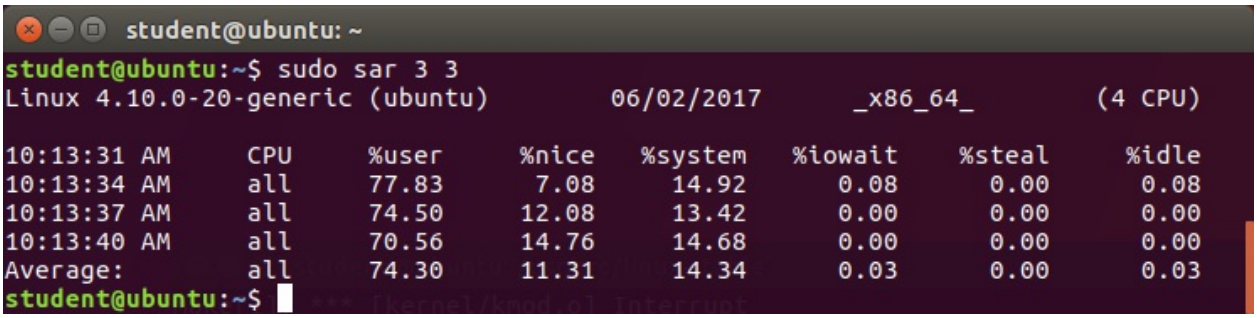
On Linux systems, backend to **sar** is **sadc** (system activity data collector) which actually accumulates statistics. Stores information in `/var/log/sa` directory, with daily frequency by default, but which can be adjusted. Data collection can be started from command line, regular periodic collection usually started as **cron** job stored in `/etc/cron.d/sysstat`.

sar then reads in this data (either from default locations or by use of file specified with `-f` option), then produces report.

sar invoked via:

```
$ sar [ options ] [ interval ] [ count ]
```

where report repeated after interval seconds a total of count times (defaults to 1). With no options, gives report on CPU usage.



List of major **sar** options, or modes, each one of which has its own sub-options:

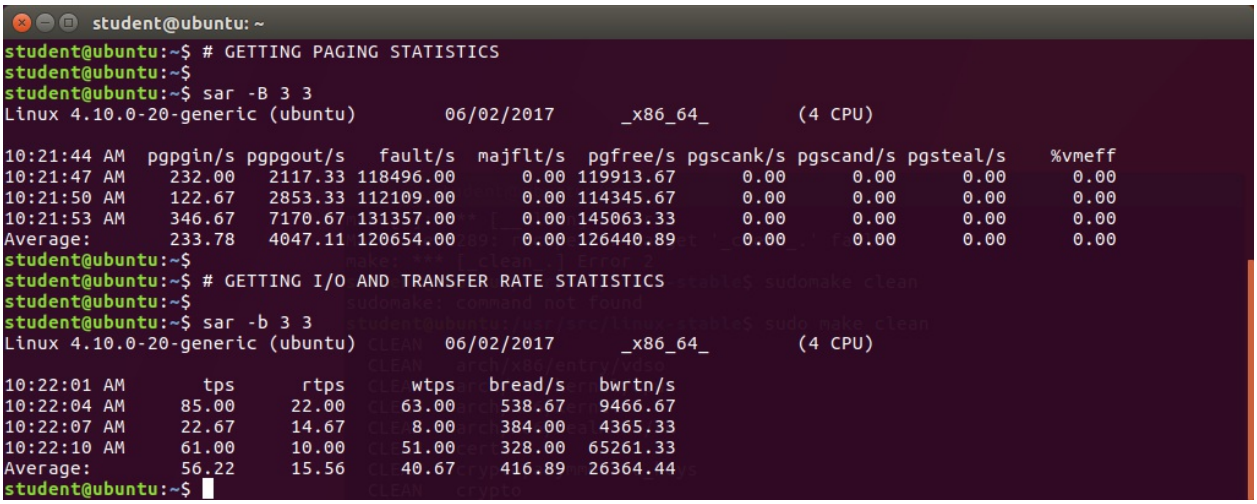
sar Options

Option	Meaning
`-A`	Almost all Information
`-b`	I/O and transfer rate statistics (similar to **iostat**)
`-B`	Paging statistics including page faults
`-x`	Block device activity (similar to **iostat -x**)
`-n`	Network statistics
`-P`	Per CPU statistics (as in `sar -P ALL 3`)

`-q`	Queue lengths (run queue, processes, and threads)
`-r`	Swap and memory utilization statistics
`-R`	Memory statistics
`-u`	CPU utilization (default)
`-v`	Statistics about inodes and files and files handles
`-w`	Context switching statistics
`-W`	Swapping statistics, pages in and out per second
`-f`	Extract information from specified file, created by the `-o` option
`-o`	Save readings in the file specified, to be read in later with `-f` option

For example, below can take look at getting paging statistics, and then I/O and transfer rate statistics.

ksar program -> **java**-based utility for generating nice graphs for **sar** data. Can be downloaded from https://sourceforge.net/projects/ksar/.



##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 12 Process Monitoring - Notes

12.2 Introduction

Keeping track running (and sleeping) processes -> essential system administration task. **ps** program -> main tool for doing so in UNIX-based operating systems for decades.

HOw ever, because utility has long + complicated history of being used differently in more than one operating system variety, has large assortment of options that can be applied w ith often confusing combinations. Another trust tool provided by **top**, w hich interactively monitors the system's state.

12.3 Learning Objectives:

- Use **ps** to view characteristics and statistics associated w ith processes.
- Identify different **ps** output fields and customize the **ps** output.
- Use **pstree** to get a visual description of the process ancestry and multi-threaded applications.
- Use **top** to view system loads interactively.

12.4 Monitoring Tools

In this section, w ill concentrate on process monitoring. Linux administrators make use of many utilities for process monitoring, eg. **ps**, **pstree**, **top**. All have long histories in UNIX-like operating systems.

Review of some of the main tools for process monitoring:

Process and Load Monitoring Utilities

Utility	Purpose	Package
top	Process activity, dynamically updated	**procps**
uptime	How long system is running and average load	**procps**
ps	Detailed information about processes	**procps**
pstree	Tree of processes and their connections	**psmisc** (or **pstree**)
mpstat	Multiple processor usage	**sysstat**
iostat	CPU utilization and I/O statistics	**sysstat**
sar	Display and collect information about system activity	**sysstat**
numstat	Information about **NUMA** (Non-Uniform Memory Architecture)	**numactl**
strace	Information about all system calls a process makes	**strace**

12.5 Viewing Process States with ps

ps -> workhorse for displaying characteristics and statistics associated with proceeses, all of which garnered from `/proc` directory associated with process.

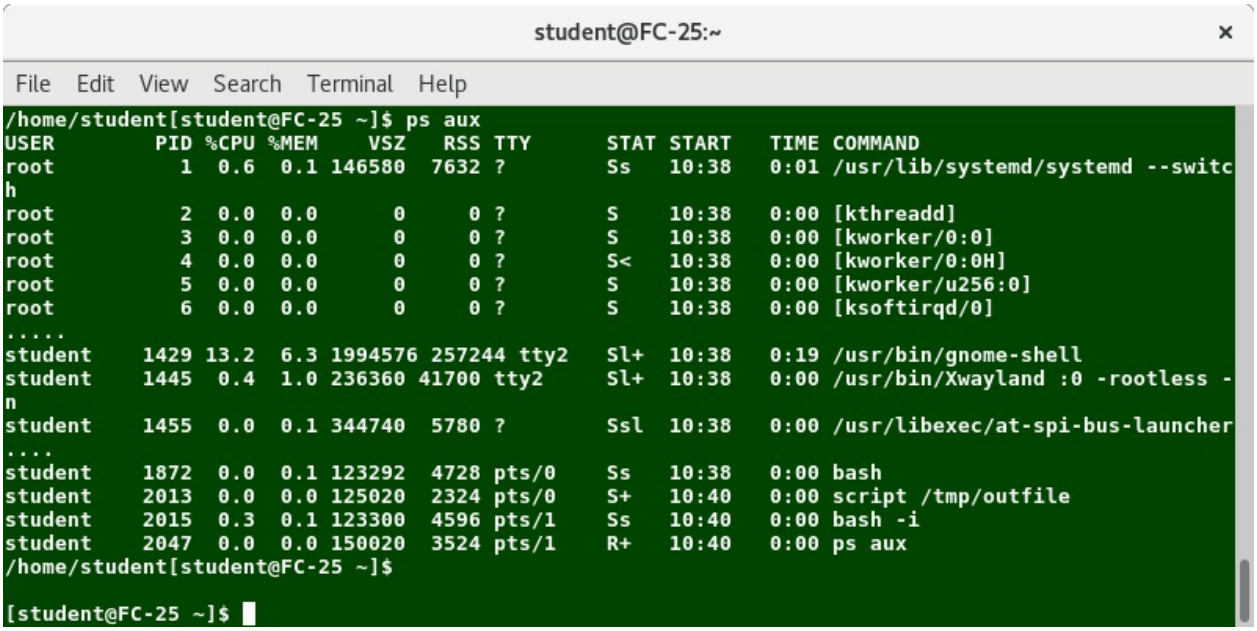
Command utility existed in all UNIX-like operating system variant. Diversity reflected in complicated potpourri of options that Linux version of **ps** accepts. Falls into three categories:

1. UNIX options, which must be preceded by -, and what may be grouped.
2. BSD options, which must *not* be preceded by -, and which may be grouped.
3. GNU long options, each of which must be preceded by --.

Having all these possible options can make life rather confusing. Most system administrators tend to use one or two standard combinations for daily use.

12.6 BSD Option Format for ps

Can see typical usage with BSD option format below, where `aux` option shows all processes. Commands surrounded by square brackets (as in `[ksoftirqd/0]`) -> threads that exist totally within kernel. If there is one for each CPU, command followed by `integer` specifying CPU it is running on.



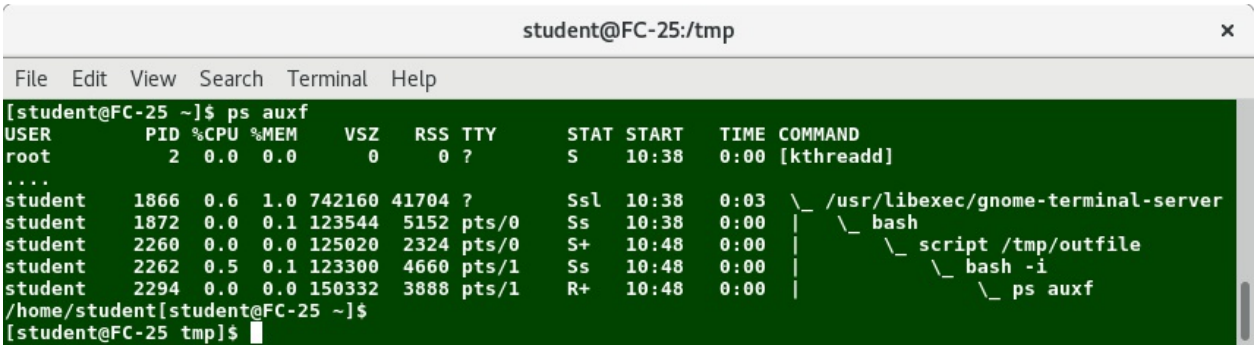
12.7 ps Output Fields

Most fields in preceding example self-explanatory. Of others:

- `VSZ`: process' virtual memory size in KB
- `RSS`: resident set size; non-swapped physical memory task is using in KB.
- `STAT`: describes state of process. in example, only see **S** for sleeping, or **R** for running. Additional character in state (where it exists):
  - `<`: high priority (not *nice*)
  - `N`: low priority (*nice*)
  - `L`: having pages locked in memory
  - `s`: session leader
  - `l`: multi-threaded
  - `+`: being in foreground process group

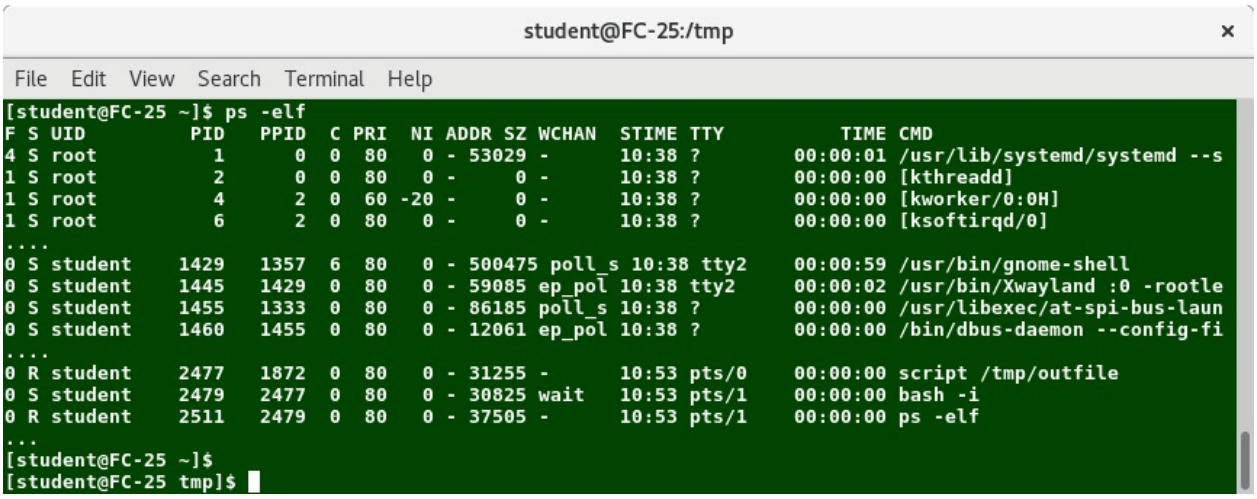
Adding `f` option will show how processes connect by ancestry:

```
$ ps auxf
```



12.8 UNIX Option Format for ps

Can see typical usage with UNIX option format below. Note: now showing **Parent Process ID** (`PPID`) and the niceness (`NI`). May observe that many processes show `PPID=2`, in this screenshot (taken from RHEL 7, using systemd) an internal kernel process `kthreadd` (designed to adopt children when parent process dies). In older kernels/systems, would see `PPID=1` for **sbin/init**, but really same thing going on.



Some common selection options in UNIX format:

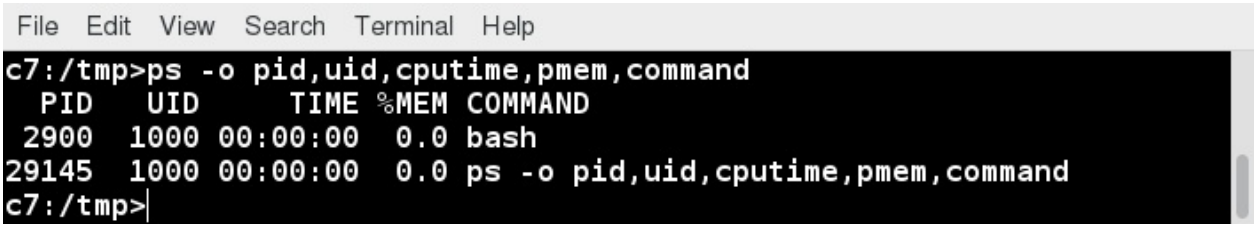
- `-A` or `-e`: Select all processes
- `N`: Negate selection (means do the opposite)
- `C`: Select by command name
- `G`: Select by real group ID (also supports names)
- `U`: Select by real user ID (also supports names)

12.9 Customizing the ps Output

If you use `-o` option, followed by comma-separated list of field identifiers, can print out customized list of **ps** fields:

- `pid`: Process ID number
- `uid`: User ID number
- `cmd`: Command with all arguments
- `cputime`: Cumulative CPU time
- `pmem`: Ratio of process's resident set size to physical memory on machine, expressed as percentage

Can see example below. Can consult **ps man** page for many other output options.



12.10 Using pstree

pstree gives visual description of process ancestry and multi-threaded applications:

```
$ pstree -aAp 2408

bash,2408
|-emacs,24998 pmonitor.tex
|  |-{emacs},25002
|  '-{emacs},25002
|-evince,18036 LFS201-SLIDES.pdf
|  |-{evince},18040
|  |-{evince},18046
|  '-{evince},18047
```

Consult **man** page for **pstree** for explanation of many options. In above, have chosen just to show information for `pid-2408`.

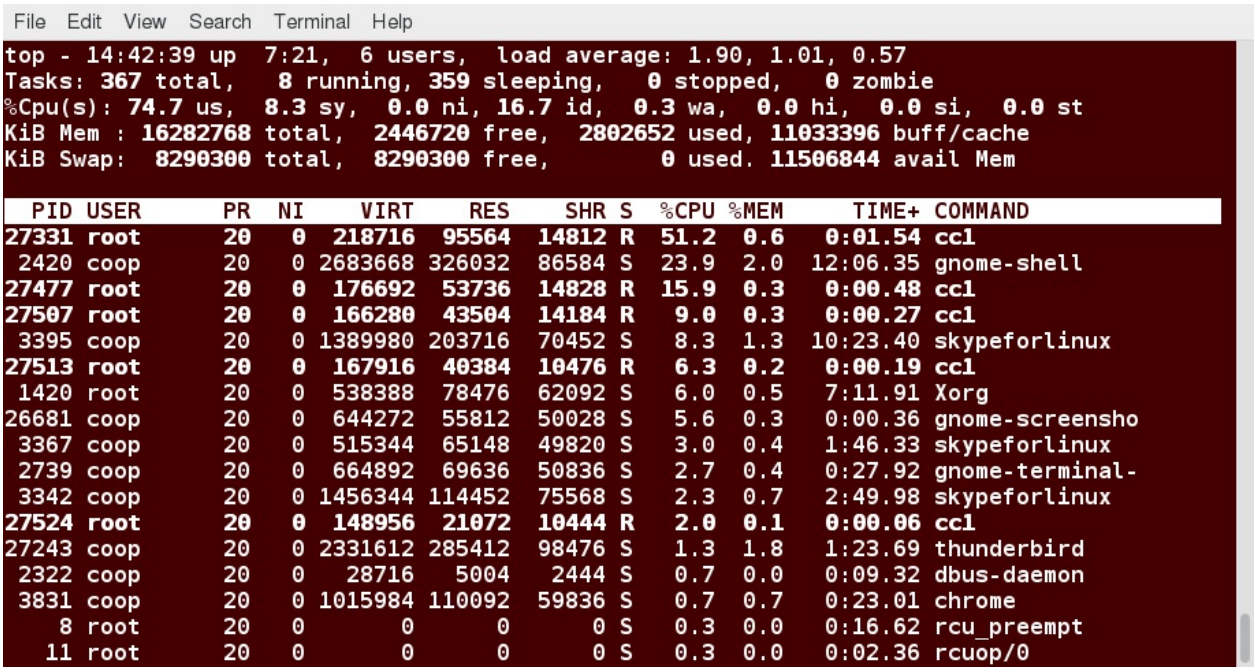
Note: one of its child processes (**evince**, `pid=18036`) has three children of its own. Another way to see that:

```
$ ls -l /proc/18036/task

total 0
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18036
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18040
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18046
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18047
```

12.11 Viewing System Loads with top

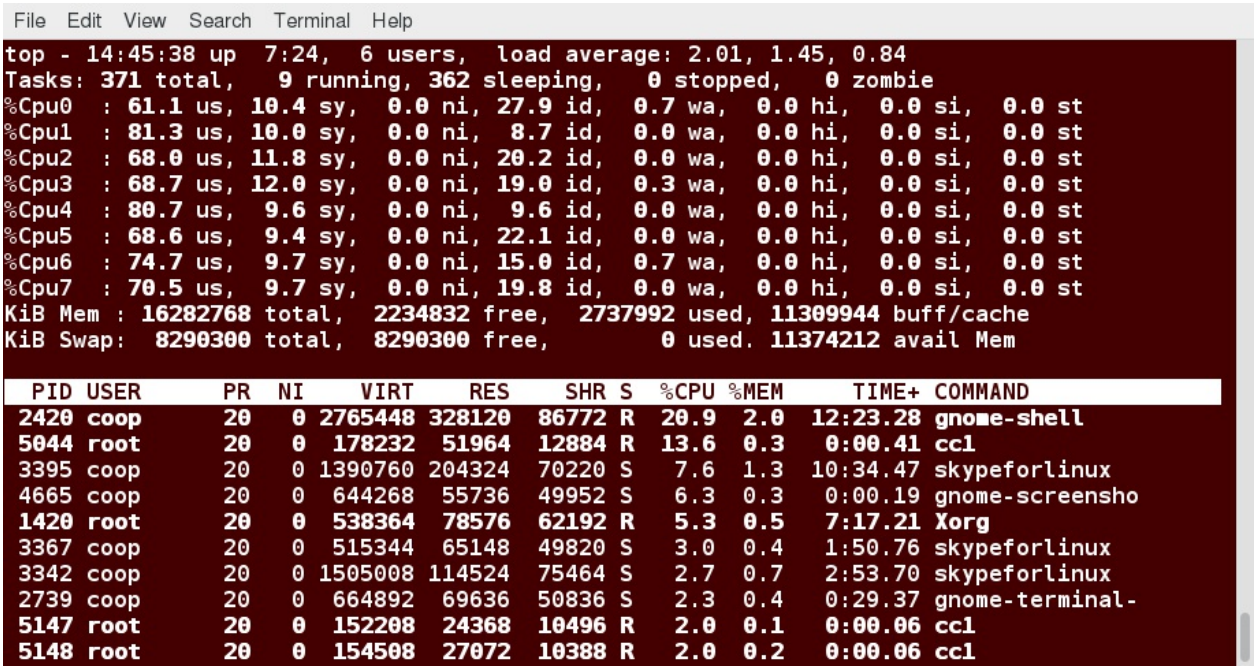
When want to know what system spending time on, first tool often used is **top**. Below shows what can be seen when using **top** without arguments.

BY default, **top** refreshes every 3.0 seconds.



12.12 top Options

top: ancient utility and has tons of options, as w ell as interactive commands triggered w hen certain keys pressed. Eg. if hit `1`, each CPU show n separately, if hit `i` only active processes show n. Can see w hat doing both gives you below.



Have lot of control over how processes sorted, w hich fields displayed. Many others besides defaults. Eg. hitting `h` or `?` gives brief list of interactive commands, `q` quits.

Can **kill** task by hitting `k`, or **renice** (change its priority) w ith `r`.

man top w ill give extensive documentation on configuration possibilities, options, interactive possibilities.

Note: there are popular alternatives to standard **top** program. Some have more visual interfaces and/or additional information, such as **htop**, **ntop**, **atop**. Most Linux distributions have graphical system monitor (eg. **gnome-system-monitor** or

ksysguard), which has **top**-like display window that can be shown.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 13 Memory: Monitoring Usage and Tuning - Notes

13.2 Introduction

Over time, systems more demanding of memory resources while RAM prices decreased and performance improved. Yet, bottlenecks in overall system performance often memory-related. CPUs and I/O subsystem can be waiting for data to be retrieved from/written to memory. Many tools for monitoring, debugging, tuning system's behavior with regard to its memory.

13.3 Learning Objectives:

- List the primary (inter-related) considerations and tasks involved in memory tuning.
- Use entries in `/proc/sys/vm` and decipher `/proc/meminfo`.
- Use **vmstat** to display information about memory, paging, I/O, processor activity, and processes' memory consumption.
- Understand how the **OOM-killer** decides when to take action and selects which processes should be exterminated to open up some memory.

13.4 Memory Tuning Considerations

Tuning memory sub-system can be complex process. Have to take note that memory usage and I/O throughput are intrinsically related, since in most cases most memory being used to cache contents of files on disk.

Thus, changing memory parameters -> large effect on I/O performance. Changing I/O parameters -> equally large converse effect on virtual memory sub-system.

When tweaking parameters in `/proc/sys/vm`, usual best practice to adjust one thing at a time and look for effects. Primary (inter-related) tasks:

- Controlling flush parameters; ie, how many pages allowed to be **dirty** and how often they flushed out to disk
- Controlling swap behavior; ie, how much pages that reflect file contents allowed to remain in memory, as opposed to those that need to be swapped out as they have no other backing store
- Controlling how much memoery **overcommission** is allowed, since many programs never need full amount of memory they request, particularly because of **copy on write** (**COW**) techniques

Memory tuning often subtle. What works in one system situation or load may be far from optimal in other circumstances.

13.5 Memory Monitoring Tools

Some important basic tools for monitoring and tuning memory in Linux:

Memory Monitoring Utilities

Utility	Purpose	Package
free	Brief summary of memory usage	**procps**
vmstat	Detailed virtual memory statistics and block I/O, dynamically updated	**procps**

pmap	Process memory map	procps

Simplest tool to use is **free**:

```
c7:/tmp> free -m
        total  used  free   shared  buff/cache  available
Mem:    15895  2946  6085   3580    6863        8984
Swap:   20023  0     20023
```

13.6 /proc/sys/vm

`/proc/sys/vm` directory contains many tunable knobs to control **Virtual Memory** system. Exactly what appears in directory depends somewhat on kernel version. Almost all entries writable (by **root**).

Note: values can be changed either by directly writing to entry, or using **sysctl** utility. Values can be set at boot time by modifying `/etc/sysctl.conf`.

Can find full documentation for `/proc/sys/vm` directory in kernel source (or kernel documentation package on distribution) usually under `Documentation/sysctl/vm.txt`.

`proc/sys/vm` **Entries**

Entry	Purpose
`admin_reserve_kbytes`	Amount of free memory reserved for privileged users
`block_dump`	Enables block I/O debugging
`compact_memory`	Turns on or off **memory compaction** (essentially defragmentation) when configured into the kernel
`dirty_background_bytes`	**Dirty** memory threshold that triggers writing uncommitted pages to disk
`dirty_background_ratio`	Percentage of total pages at which kernel will start writing dirty data out to disk
`dirty_bytes`	The amount of dirty memory a process needs to initiate writing on its own
`dirty_expire_centisecs`	When dirty data is old enough to be written out in hundredths of a second
`dirty_ratio`	Percentage of pages at which a process writing will start writing out dirty data on its own
`dirty_writeback_centisecs`	Internal in which periodic writeback daemons wake up to flush. If set to zero, no automatic periodic writeback
`drop_caches`	Echo 1 to free page cache, 2 to free dentry and inode caches, 3 to free all. note only **clean** cached pages are dropped; do **sync** first to flush dirty pages
`extfrag_threshold`	Controls when kernel should compact memory
`hugepages_treat_as_movable`	Used to toggle how huge pages are treated
`hugetlb_shm_group`	Sets a group ID that can be used for System V huge pages
`laptop_mode`	Can control a number of features to save power on laptops
`legacy_va_layout`	Use old layout (2.4 kernel) for how memory mappings are displayed

`lowmem_reserve_ratio`	Controls how much low memory is reserved for pages that can only be there; ie, pages which can go in high memory instead will do so. Only important on 32-bit systems with high memory
`max_map_count`	Maximum number of memory mapped areas a process may have. Default is 64 K
`min_free_kbytes`	Minimum free memory that must be reserved in each zone
`mmap_min_addr`	How much address space a user process cannot memory map. Used for security purposes, to avoid bugs where accidental kernel null dereferences can overwrite the first pages used in an application
`nr_hugepages`	Minimum size of hugepage pool
`nr_pdflush_hugepages`	Maximum size of the hugepage pool = `nr_hugepages*nr_overcommit_hugepages`
`nr_pdflush_threads`	Current number of **pdflush** threads; not writeable
`oom_dump_tasks`	If enabled, dump information produced when **oom-killer** cuts in
`oom-kill-allocating_task`	If set, **oom-killer** kills task that triggered out-of-memory situation, rather than trying to select best one
`overcommit_kbytes`	One can set either `overcommit_ratio` or this entry, not both
`overcommit_memory`	If 0, kernel estimates how much free memory is left when allocations are made. If 1, permits all allocations until memoery actually does run out. If 2, prevents any overcommission
`overcommit_ratio`	If overcommit_memory = 2 memory commission can reach swap plus this percentage of **RAM**
`page-cluster`	Number of pages that can be written to swap at once, as a power of two. Default if 3 (which means 8 pages)
`panic_on_oom`	Enable system to crash on an out of memory situation
`percpu_pagelist_fraction`	Fraction of pages allocated for each cpu in each zone for hot_pluggable CPU machines
`scan_unevictable_pages`	If written to, system will scan and try to move pages to try and make them reclaimable
`stat_interval`	How often vm statistics are updated (default 1 second) by **vmstat**
`swappiness`	How aggressively should the kernel swap
`user_reserve_kbytes`	If `overcommit_memory` is set to 2 this sets how low the user can draw memory resources
`vfs_cache_pressure`	How aggressively the kernel should reclaim memory used for inode and dentry cache. Default is 100; if 0 this memoery is never reclaimed due to memory pressure

13.7 vmstat

vmstat: multi-purpose tool that displays information about memory, paging, I/O, processor activity and processes. Has many options. General form of command:

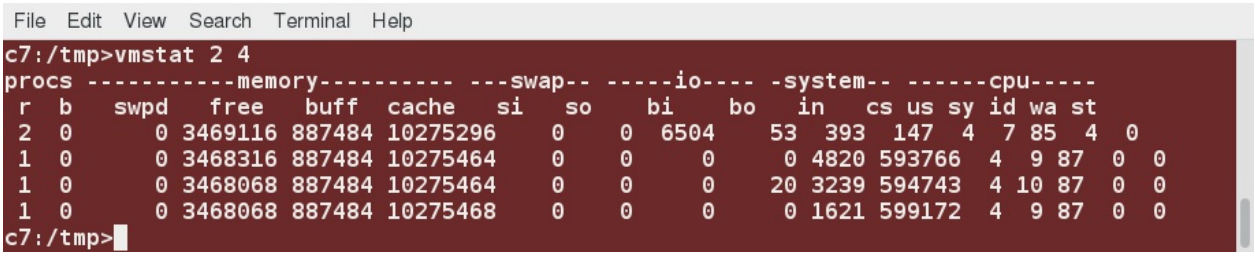
```
$ vmstat [options] [delay] [count]
```

If **delay** given in seconds, report repeated at interval count times. If **count** not given, **vmstat** will keep reporting statistics

forever, until killed by signal, such as `Ctrl-C`.

If no other arguments given, can see what **vmstat** displays, where first line shows averages since last reboot, while succeeding lines show activity during specified interval.

```
$ vmstat 2 4
```



Fields shown are:

vmstat Fields

Field	Subfield	Meaning
Processes	r	Number of processes waiting to be scheduled in
Processes	b	Number of processes in uninterruptible sleep
memory	swpd	Virtual memory used (KB)
memory	free	Free (idle) memory (KB)
memory	buff	Buffer memory (KB)
memory	cache	Cached memory (KB)
swap	si	Memory swapped in (KB)
swap	so	Memory swapped out (KB)
I/O	bi	Blocks read from devices (block/sec)
I/O	bo	Blocks written to devices (block/sec)
system	in	Interrupts/second
system	cs	Context switches/second
CPU	us	CPU time running user code (percentage)
CPU	sy	CPU time running kernel (system) code (percentage)
CPU	id	CPU time idle (percentage)
CPU	wa	Time waiting for I/O (percentage)
CPU	st	Time "stolen" from virtual machine (percentage)

If option `-S m` given, memory statistics will be in MB instead of KB.

With `-a` option, **vmstat** displays information about **active** and **inactive** memory, where **active** memory pages are those which

have been recently used. May be **clean** (disk contents are up to date) or **dirty** (need to be flushed to disk eventually), By contrast, **inactive memory** pages have not been recently used and are more likely to be clean and are released sooner under memory pressure:

```
$ vmstat -a 2 4
```

Memory can move back and forth between active and inactive lists, as they get newly referenced, or go a long time between uses.

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>vmstat -SM -a 2 4
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free  inact active   si   so    bi    bo   in   cs us sy id wa st
 2  0      0   6611   5972   2911    0    0  6450    52  392  311  4  7 85  4  0
 2  0      0   6611   5972   2911    0    0     0   186 1602 601224  4  9 87  0  0
 1  0      0   6612   5970   2911    0    0     0     0 1800 593070  4  9 86  0  0
 1  0      0   6612   5970   2912    0    0     0     2 1615 587838  4  9 87  0  0
c7:/tmp>
```

To get table of memory statistics and certain event counters, use `-s` option:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>vmstat  -s
     16282936 K total memory
      1644676 K used memory
      2983884 K active memory
      6170976 K inactive memory
      6711528 K free memory
       888112 K buffer memory
      7038620 K swap cache
      8290300 K total swap
            0 K used swap
      8290300 K free swap
       397791 non-nice user cpu ticks
          595 nice user cpu ticks
       645895 system cpu ticks
      7848947 idle cpu ticks
       325912 IO-wait cpu ticks
            0 IRQ cpu ticks
          499 softirq cpu ticks
            0 stolen cpu ticks
    591242339 pages paged in
      4817760 pages paged out
            0 pages swapped in
            0 pages swapped out
     36056386 interrupts
   3204205246 CPU context switches
   1496401448 boot time
      1318407 forks
c7:/tmp>
```

To get table of disk statistics, use `-d` option:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>vmstat -d
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors      ms  total merged sectors      ms    cur    sec
sda   556655   7077 824315972 45801817   3778  11692  111736  346798      0   3771
sdb   902791   6144 352365208 4576974 287799 194865 9550552 1030517      0    501
dm-0   23238      0 26917042 1242033   2645      0   21136  184567      0    722
dm-1   29232      0 39888592 1630477   1293      0   10320   55072      0    159
dm-2   52769      0 90611938 3974865   4047      0   32352   38436      0    399
dm-3   13530      0 19852730  352195    698      0    5560    9288      0    100
dm-4  256511      0 522426640 26326775    390      0    3096   62784      0   2122
dm-5      60      0     912    2547      0      0       0       0      0      2
dm-6      19      0     152    1500      0      0       0       0      0      1
dm-7      60      0     912    4117      0      0       0       0      0      4
dm-8   99246      0 124611034 5390402   4912      0   39272  261699      0    484
dm-9      60      0     912    4618      0      0       0       0      0      4
dm-10  89687      0 39028146  389764 136223      0 2783432  293439      0    114
dm-11 257274      0 19785464  308130  30015      0  241152  313049      0     42
dm-12 161343      0 266631144 2953670  14217      0 2541816  292337      0    262
loop0 2907517      0 5815826  124532      0      0       0       0      0      6
c7:/tmp>
```

vmstat Disk Fields

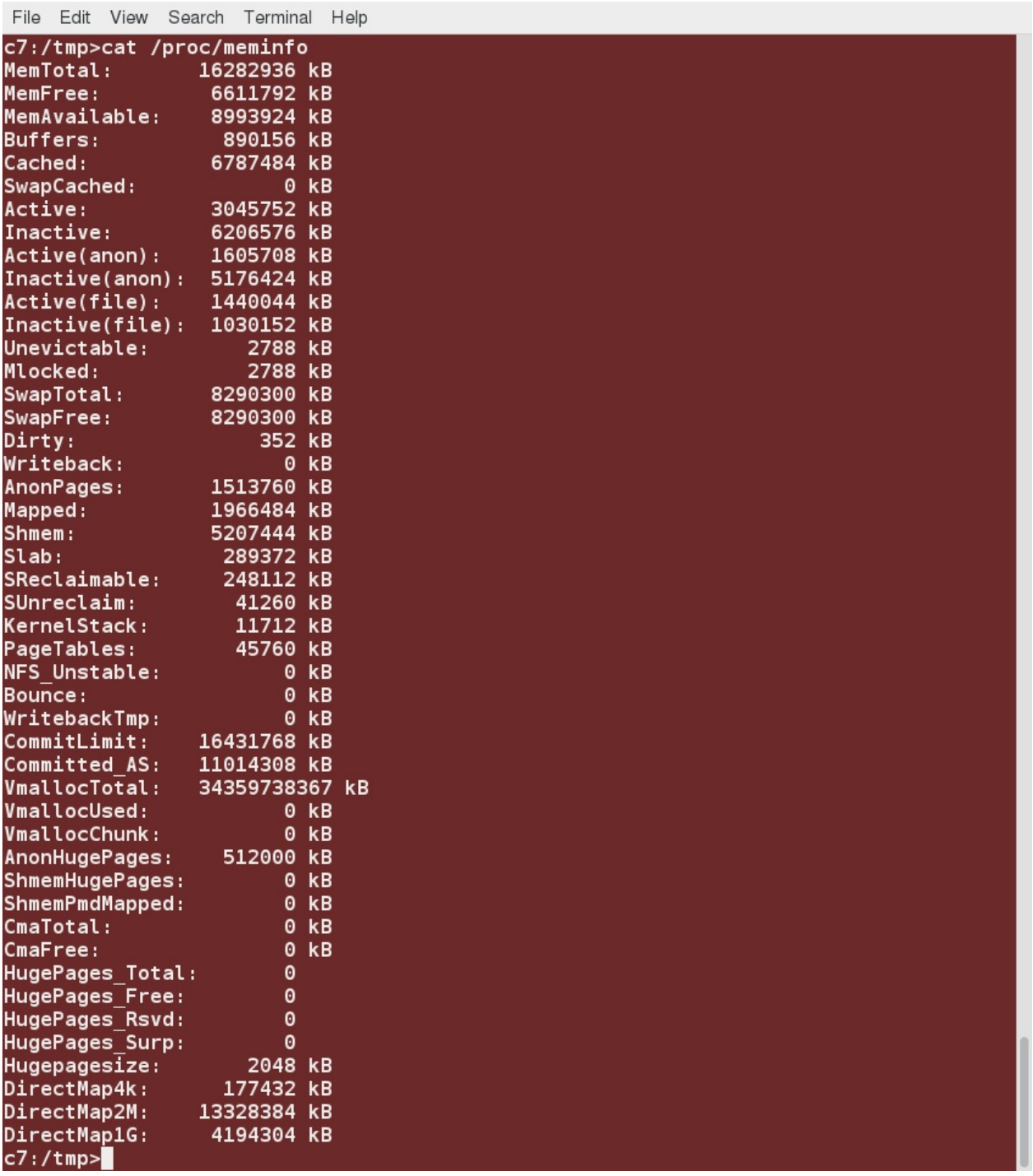
Field	Subfield	Meaning
reads	total	Total reads completed successfully
reads	merged	Grouped reads (resulting in one I/O)
reads	ms	Milliseconds spend reading
w rites	total	total w rites completed successfully
w rites	merged	Grouped w rites (resulting in one I/O)
w rites	ms	Milliseconds spent w riting
I/O	cur	I/O in progress
I/O	sec	seconds spent for I/O

If w ant to just get some quick statistics on only one partition, use `-p` option:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>vmstat -p /dev/sdb1 2 4
sdb1          reads   read sectors  writes    requested writes
              358324   26917482     192161    3988152
              358324   26917482     192161    3988152
              358324   26917482     192161    3988152
              358324   26917482     192161    3988152
c7:/tmp>
```

13.8 /proc/meminfo

As noted earlier, relatively lengthy summary of memory statistics in `/proc/meminfo`:



Worthw hile to go through listing and understand most of the entries:

`/proc/meminfo` Entries

Entry	Meaning
`MemTotal`	Total usable RAM (physical minus some kernel reserved memory)
`MemFree`	Free memory in both low and high zones
`Buffers`	Memory used for temporary block I/O storage
`Cached`	Page cache memory, mostly for file I/O

`SwapCached`	Memory that was swapped back in but is still in the swap file
`Active`	Recently used memory, not to be claimed first
`Inactive`	Memory not recently used, more eligible for reclamation
`Active (anon)`	Active memory for **anonymous** pages
`Inactive (anon)`	Inactive memory for **anonymous** pages
`Active (file)`	Active memory for **file**-backed pages
`Inactive (file)`	Inactive memory for **file**-backed pages
`Unevictable`	Pages which can not be swapped out of memory or released
`Mlocked`	Pages which are locked in memory
`SwapTotal`	Total swap space available
`SwapFree`	Swap space not being used
`Dirty`	Memory which needs to be written back to disk
`Writeback`	Memory actively being written back to disk
`AnonPages`	Non-file back pages in cache
`Mapped`	Memory mapped pages, such as libraries
`Shmem`	Pages used for shared memory
`Slab`	Memory used in slabs
`SReclaimable`	Cached memory in slabs that can be reclaimed
`SUnreclaim`	Memory in slabs that can't be reclaimed
`KernelStack`	Memory used in kernel stack
`PageTables`	Memory being used by page table structures
`Bounce`	Memory used for block device bounce buffers
`WritebackTmp`	Memory used by **FUSE** filesystems for writeback buffers
`CommitLimit`	Total memory available to be used, including overcommission
`Committed_AS`	Total memory presently allocated, whether or not it is used
`VmallocTotal`	Total memory available in kernel for **vmalloc** allocations
`VmallocUsed`	Memory actually used by **vmalloc** allocations
`VmallocChunk`	Largest possible contiguous **vmalloc** area
`HugePages_Total`	Total size of the huge page pool
`HugePages_Free`	Huge pages that are not yet allocated
`HugePage_Rsvd`	Huge pages that have been reserved, but not yet used
`HugePages_Surp`	Huge pages that are surplus, used for overcommission

`Hugepagesize`	Size of a huge page

Note: exact entries seen may depend on exact kernel version being run.

13.9 OOM Killer

Simplest way to deal with memory pressure -> permit memory allocations to succeed as long as free memory is available, fail when all memory exhausted.

Second simplest way -> use **swap** space on disk to push some resident memory out of core. In this case, total available memory (at least in theory) is actual RAM plus size of **swap** space. Hard part of this is to figure out which pages of memory to swap out when pressure demands. In this approach, once swap space filled, requests for new memory must fail.

Linux, however, goes one better. Permits system to overcommit memory, so that it can grant memory requests that exceed size of RAM plus **swap**. Might seem foolhardy, but many (if not most) processes do not use all requested memory.

Example 1: program that allocates 1 MB buffer, and then uses only few pages of memory. Example 2: every time child process forked, receives copy of entire memory space of parent. Because Linux uses COW (copy on write) technique, unless one of the processes modifies memory, no actual copy needs to be made. However, kernel has to assume that copy might need to be done.

Thus, kernel permits overcommission of memory, but only for pages dedicated to user processes. Pages used within kernel not swappable, and always allocated at request time.

Can modify, and even turn off this overcommission by setting value of `/proc/sys/vm/overcommit_memory`:

- 0: (default) Permit overcommission, but refuse obvious overcommits, and give root users somewhat more memory allocation than normal users
- 1: All memory requests are allowed to overcommit
- 2: Turn off overcommission. Memory requests will fail when the total memory commit reaches the size of the **swap** space plus a configurable percentage (50 by default) of RAM. This factor is modified changing `/proc/sys/vm/overcommit_ratio`.

If available memory exhausted, Linux invokes **OOM**-killer (**O**ut **O**f **M**emory) to decide which process(es) to exterminate to open up memory.

No precise science, algorithm must be heuristic, cannot satisfy everyone. In minds of many developers, purpose of OOM-killer to permit graceful shutdown, rather than be part of normal operations.

An amusing take on this by Andries Brouwer (https://lwn.net/Articles/104185/):

> An aircraft company discovered that it was cheaper to fly its planes with less fuel on board. The planes would be lighter and use less fuel and money was saved. On rare occasions however the amount of fuel was insufficient, and the plane would crash. This problem was solved by the engineers of the company by the development of a special OOF (out-of-fuel) mechanism. In emergency cases a passenger was selected and thrown out of the plane. (When necessary, the procedure was repeated.) A large body of theory was developed and many publications were devoted to the problem of properly selecting the victim to be ejected. Should the victim be chosen at random? Or should one choose the heaviest person? Or the oldest? Should passengers pay in order not to be ejected, so that the victim would be the poorest on board? And if for example the heaviest person was chosen, should there be a special exception in case that was the pilot? Should first class passengers be exempted? Now that the OOF mechanism existed, it would be activated every now and then, and eject passengers even when there was no fuel shortage. The engineers are still studying precisely how

> this malfunction is caused.

In order to make decisions of w ho gets sacrificed to keep system alive, value called **badness** computed (can be read from `/proc/[pid]/oom_score`) for each process on system and order of killing determined by this value.

Tw o entries in same directory can be used to promote/demote likelihood of extermination. Value of `oom_adj` : number of bits points should be adjusted by. Normal users can only increase badness. Decrease (negative value for `oom_adj`) can only be specified by superuser. Value of `oom_adj_score` directly adjusts point value. Note: use of `oom_adj` deprecated.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 14 I/O Monitoring and Tuning - Notes

14.3 Learning Objectives:

- Understand the importance of monitoring I/O activity and when it constitutes system performance bottlenecks.
- Use **iostat** to monitor system I/O device activity.
- Use **iotop** to display a constantly updated table of current I/O usage.
- Use **ionice** to set both the **I/O scheduling class** and the **priority** for a given process.

14.4 I/O Monitoring and Disk Bottlenecks

Disk performance problems -> strongly coupled to other factors, eg. insufficient memory, inadequate network hardware/tuning. Disentanglement difficult.

Rule: system considered as **I/O-bound** when CPU found sitting idle waiting for I/O to complete, or network waiting to clear buffers.

However, can be misled. What appears to be insufficient memory can result from too slow I/O. If memory buffers being used for reading/writing fill up, may appear that memory is problem, when real problem is that buffers are not filling up or emptying out fast enough. Similarly, network transfers may be waiting for I/O to complete, causing network throughput to suffer.

Real-time monitoring + tracing -> both necessary tools for locating/mitigating disk bottlenecks. However, rare/non-repeating problems can make this difficult to accomplish.

Many relevant variables, I/O tuning complex. Will also consider **I/O scheduling** later.

14.5 iostat

iostat: basic workhorse utility for monitoring I/O device activity on system. Can generate reports with lot of information, with precise content controlled by options.

Can see below what simply typing **iostat** shows:

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>iostat
Linux 4.11.3 (c7)       06/02/2017      _x86_64_        (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          1.27    0.00    0.94    0.25    0.00   97.53

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.60        32.32         0.25     509530       3908
sdb              11.76       283.35       176.27    4467320    2779108
dm-0              0.19        30.42         0.24     479617       3812
dm-1              0.07         0.33         0.00       5196         12
dm-2              0.02         0.12         0.00       1921         12
dm-3              0.03         0.15         0.00       2417         12
dm-4              0.02         0.12         0.00       1892         48
....
dm-12             1.59       118.88        74.54    1874200    1175172
loop0             0.99         1.01         0.00      16002          0

coop@c7:/tmp
c7:/tmp>
```

After brief summary of CPU utilization, I/O statistics given: **tps** (I/O transactions per second; logical requests can be **merged** into one actual request), clocks read/written per unit time, where blocks are generally sectors of 512 bytes; total blocks read/written.

Information broken out by disk partition (and if LVM is being used also by **dm**, or device mapper, logical partitions).

14.6 iostat Options

Somewhat different display generated by giving `-k` option, results in showing KB instead of blocks. Can also use `-m` to get results in MB.

```
$ iostat -k
```

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>iostat -k
Linux 4.11.3 (c7)       06/02/2017      _x86_64_        (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          1.30    0.00    0.94    0.27    0.00   97.48

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.82       258.31         0.25    4211894       4064
sdb              11.96       274.27       174.60    4472104    2846868
dm-0              0.19        30.67         0.24     500021       3944
dm-1              0.07         0.32         0.00       5196         12
dm-2              0.02         0.12         0.00       1921         12
dm-3              0.03         0.15         0.00       2417         12
dm-4              0.24       225.93         0.00    3683852         72
....
dm-12             1.55       114.94        72.17    1874200    1176720
loop0             0.96         0.98         0.00      16002          0
c7:/tmp>
c7:/tmp>
```

Another useful option: `-N`, to show device name (or `-d` for somewhat different format), as shown below:

```
$ iostat -N
```

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>iostat -N
Linux 4.11.3 (c7)       06/02/2017      _x86_64_        (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.29    0.00    0.94    0.27    0.00   97.49

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.81       256.52         0.25    4219286       4108
sdb              12.05       272.67       174.42    4484900    2868920
VG2-pictures      0.19        30.85         0.24     507413       3988
VG2-dead          0.07         0.32         0.00       5196         12
VG2-iso_images    0.02         0.12         0.00       1921         12
VG2-audio         0.03         0.15         0.00       2417         12
VG2-virtual       0.24       223.97         0.00    3683852         72
VG2-w7back        0.00         0.03         0.00        456          0
VG2-P             0.00         0.00         0.00         76          0
VG2-isabelle      0.00         0.03         0.00        456          0
VG2-dead2         0.21         0.88         0.00      14489         12
VG2-PLAY          0.00         0.03         0.00        456          0
VG-local          7.02        35.91        65.06     590629    1070180
VG-src            1.27         5.85         0.44      96264       7224
VG-vms            1.54       113.95        71.56    1874200    1177052
loop0             0.95         0.97         0.00      16002          0
c7:/tmp>
c7:/tmp>
```

14.7 iostat Extended Options

Much more detailed report obtained by using `-x` option (for extended).

```
$ iostat -xk
```

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>iostat -xk
Linux 4.11.3 (c7)       06/02/2017      _x86_64_        (8 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.30    0.00    0.94    0.27    0.00   97.49

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     0.02    0.72    0.08   251.52     0.25   630.23     0.05   63.63   58.61  109.68  25.59   2.04
sdb               0.09     4.66    5.79    6.41   267.04   174.83    72.47     0.03    2.41    0.49    4.14   0.43   0.52
dm-0              0.00     0.00    0.13    0.06    30.87     0.24   330.18     0.02  128.02  126.65  130.89  97.60   1.84
dm-1              0.00     0.00    0.07    0.00     0.31     0.00     9.36     0.00    7.93    7.64   73.20   4.19   0.03
dm-2              0.00     0.00    0.02    0.00     0.11     0.00    14.01     0.00   15.46   14.99   41.40  14.12   0.02
dm-3              0.00     0.00    0.03    0.00     0.14     0.00    11.09     0.00   36.09   36.28   19.80  12.95   0.03
dm-4              0.00     0.00    0.23    0.00   218.98     0.00  1897.46     0.02   95.29   94.75  199.20   8.82   0.20
dm-12             0.00     0.00    1.12    0.38   111.41    70.02   240.59     0.02   11.01    0.86   40.76   0.44   0.07
loop0             0.00     0.00    0.93    0.00     0.95     0.00     2.05     0.00    0.01    0.01    0.00   0.01   0.00
c7:/tmp>
c7:/tmp>
```

Fields seen above have following meanings:

Extended iostat Fields

Field	Meaning
Device	Device or partition name

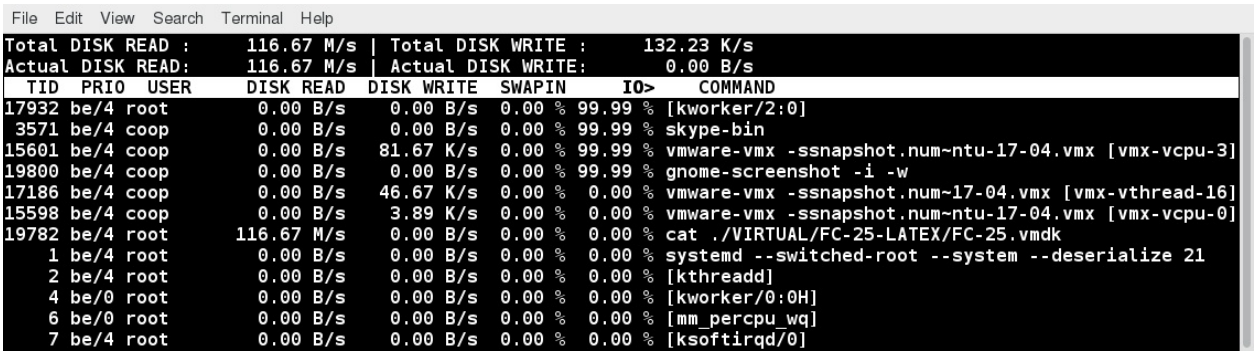
rrqm/s	Number of read requests merged per second, queued to device
w rqm/s	Number of w rite requests merged per second, queued to device
r/s	number of read requests per second, issued to device
w /s	number of w rite requests per second, issued to device
rkB/s	KB read from the device per second
w kB/s	KB w ritten to the device per second
avgrq-sz	Average request size in 512 byte sectors per second
avgqu-sz	Average queue length of requests issued to the device
aw ait	Average time (in msecs) I/O requests betw een w hen a request is issued and w hen it is completed: queue time plus service time
svctm	Average service time (in msecs) for I/O requests
%util	Percentage of CPU time during the device serviced requests

Note: if utilization percentages approaches 100, system saturated, or I/O bound.

14.8 iotop

iotop -> another very useful utility, must be run as root. Displays table of current I/O usage, updated periodically, like **top**. Can see below w hat typing `sudo iotop` w ith no options show s.

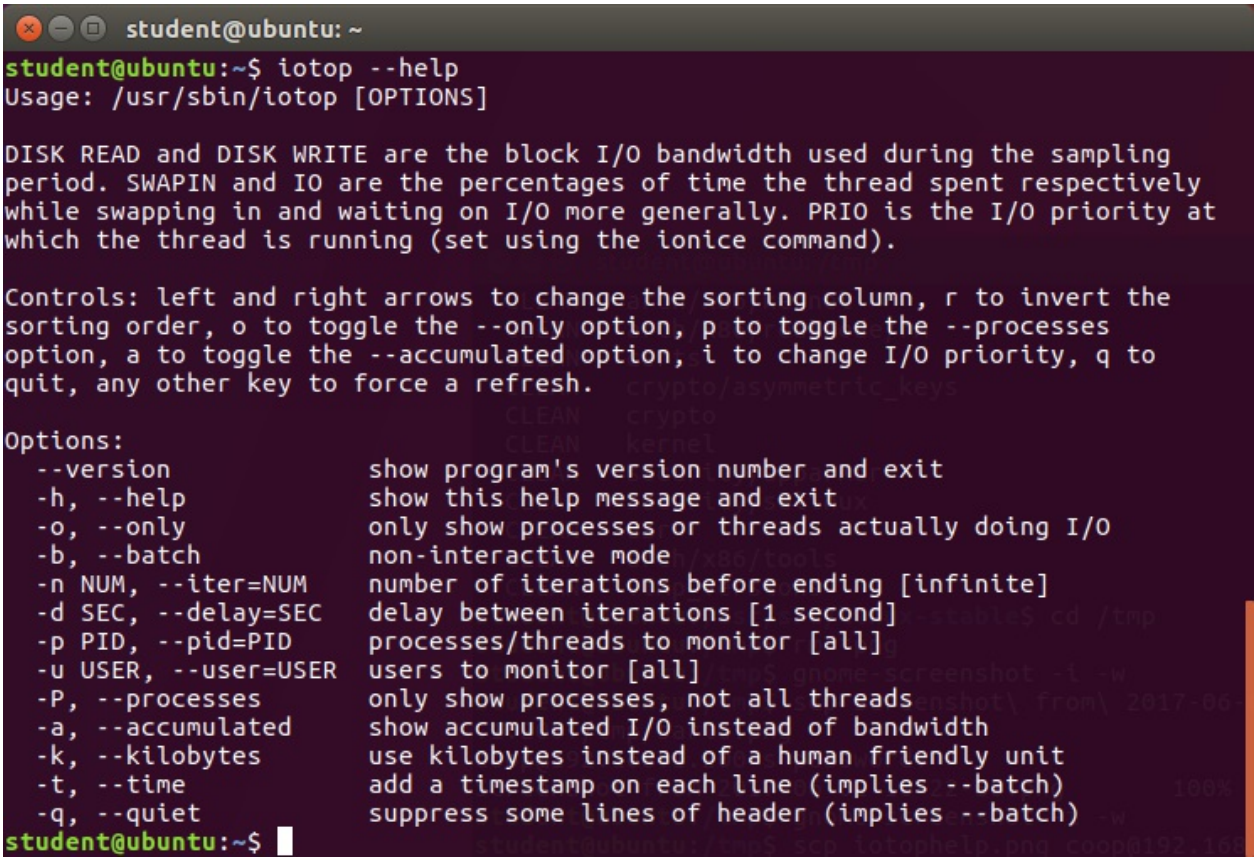
Note: `be` and `rt` entries in `PRIO` field explained in **ionice** section, stand for **best effort** and **real time**.



Available options show n by using `--help` option.

```
$ iotop --help
```

Using `-o` option can be useful to avoid clutter.



14.9 Using ionice to Set I/O Priorities

ionice utility sets both I/O **scheduling class** and **priority** for given process. Takes the form:

```
$ ionice [-c class] [-n priority] [-p pid] [COMMAND [ARGS] ]
```

If **pid** given with `-p` argument results apply to requested process, otherwise it is process that will be started by `COMMAND` with possible arguments. If no arguments given, **ionice** returns scheduling class and priority of current shell process:

```
$ ionice
idle: prio 7
```

`-c` parameter specifies I/O scheduling class, which can have following 3 values:

I/O Scheduling Class

I/O Scheduling Class	`-c` value	Meaning
None or Unknown	0	Default value
Real Time	1	Get first access to disk, can starve other processes. Priority defines how big a time slice each process gets
Best Effort	2	All programs serviced in round-robin fashion, according to priority settings. The Default

Idle	3	No access to disk I/O unless no other program has asked for it for a defined period

Best Effort and **Real Time** classes take `-n` argument which gives **priority**, which can range from 0 to 7, with 0 being highest priority:

```
$ ionice -c 2 -n 3 -p 30078
```

Note: **ionice** works only when using **CFQ** I/O Scheduler (will talk about in next chapter).

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 15 I/O Scheduling - Notes

15.2 Introduction

System performance often depends very heavily on optimizing I/O scheduling strategy. Many (often competing) factors influence behavior, including minimizing hardware access times, avoiding wear/tear on storage media, ensuring data integrity, granting timely access to applications that need to do I/O, being able to prioritize important tasks. Linux offers variety of **I/O Schedulers** to choose from, each of which has tunable parameters + number of utilities for reporting on/analyzing I/O performance.

15.3 Learning Objectives:

- Explain the importance of I/O scheduling and describe the conflicting requirements that need to be satisfied.
- Delineate and contrast the options available under Linux.
- Understand how the **CFQ** (**C**ompletely **F**air **Q**ueue) and **Deadline** algorithms work.

15.4 I/O Scheduling

I/O scheduler provides interface between generic block layer and low-level physical device drivers. Both VM (Virtual Memory) and VFS (Virtual File System) layers submit I/O requests to block devices. Job of I/O scheduling layer to prioritize + order requests before they are given to block devices.

Any I/O scheduling algorithm has to satisfy certain (sometimes conflicting) requirements:

- Hardware access times should be minimized; ie, requests should be order according to physical location on disk. Leads to **elevator** scheme where requests inserted in pending queue in physical order
- Requests should be merged to extent possible to get as big a contiguous region as possible, which also minimizes disk access time
- Requests should be satisfied with as low a latency as feasible. In some cases, determinism (in sense of deadlines) important
- Write operations usually wait to migrate from caches to disk without stalling process. Read operations almost always require process to wait for completion before proceeding further. Favoring reads over writes leads to better parallelism and system responsiveness
- Processes should share I/O bandwidth in fair, consciously prioritized fashion. Even if it means some overall performance slowdown of I/O layer, process throughput should not suffer inordinately

15.5 I/O Scheduler Choices

Since demands can be conflicting, different I/O schedulers may be appropriate for different workloads, eg, large database server vs. desktop system. Different hardware may mandate different strategy. To provide flexibility, Linux kernel has object oriented scheme, where pointers to various needed functions supplied in a data structure, the particular one of which can be selected at boot on kernel command line:

```
linux ... elevator=[cfq|deadline|noop]
```

At least one of I/O scheduling algorithms must be compiled into kernel. Current choices:

- Completely Fair Queueing (CFQ)
- Deadline Scheduling

- noop (A simple scheme)

Default choice is compile configuration option. Modern distributions choose either CFQ or Deadline.

15.6 I/O Scheduling and SSD Devices

Gradual introduction of **SSD** (**S**olid **S**tate **D**rive) devices, which use flash memory to emulate hard disks, has important implications for I/O scheduling.

SSDs do not require elevator scheme + benefit from **wear leveling** to spread I/O over the devices which have limited write/erase cycles.

Can examine `/sys/block/<device>/queue/rotational` to see whether or not device is **SSD** or not:

```
$ cat /sys/block/sda/queue/rotational
1
$ cat /sys/block/sdb/queue/rotational
0
```

15.7 Tunables and Switching the I/O Scheduler at Runtime

Each of I/O schedulers exposes parameters which can be used to tune behavior at run time. Parameters accessed through pseudo-filesystem mounted at `/sys`.

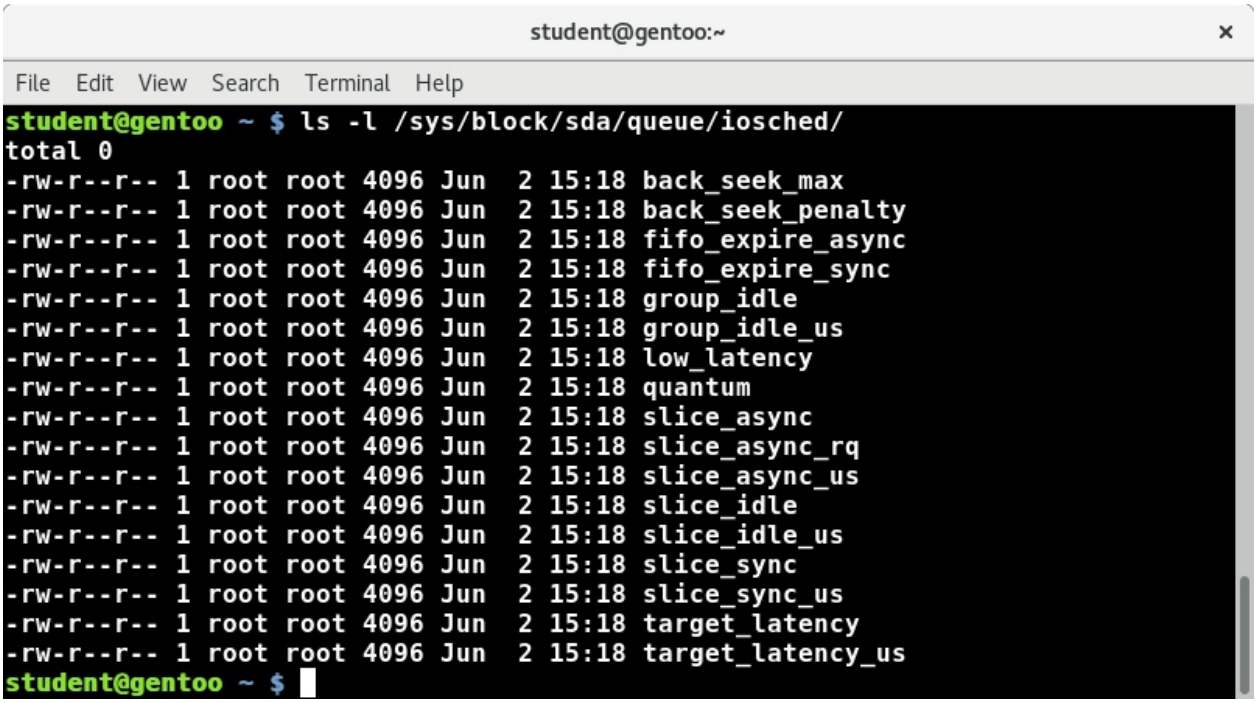
In addition, possible to use different I/O schedulers for different devices. Choice can be made easily through command line. For example:

```
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
$ echo noop > /sys/block/sda/queue/scheduler
$ cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

Actual tunables vary according to particular I/O scheduler, can be found under `/sys/block/<device>/queue/iosched`.

Can see actual tunables for disk using **CFQ** below.

Will discuss some parameters shortly.



15.9 CFQ (Completely Fair Queue) Scheduler

CFQ (**C**ompletely **F**air **Q**ueue) method has goal of equal spreading of I/O bandw idth among all processes submitting requests.

Theoretically, each process has ow n I/O queue, w hich w orks together w ith dispatch queue w hich receives actual requests on w ay to device. In actual practice, number of queues fixed (at 64) and hash process based on process ID used to select queue w hen request submitted.

Dequeuing of requests done **round robin** style on all queues, each one of w hich w orks in **FIFO** (**F**irst **I**n **F**irst **O**ut) order. Thus, w ork spread out. To avoid excessive seeking operations, entire round selected, and then sorted into dispatch queue before actual I/O requests issued to device.

15.10 CFQ Tunables

In examples below, parameter `HZ` -> kernel-configured quantity, corresponds to number of **jiffies** per second, w hich kernel uses as coarse measure of time. Time unit `HZ/2` = 0.5 seconds, `5 * HZ` = 5 seconds, etc.

- `quantum` : Maximum queue length in one round of service (Default = 4)
- `queued` : Minimum request allocation per queue (Default = 8)
- `fifo_expire_sync` : `FIFO` timeout for sync requests (Default = `HZ/2`)
- `fifo_expire_async` : `FIFO` timeout for async requests (Default = `5 * HZ`)
- `fifo_batch_expire` : Rate at w hich the `FIFO` 's expire (Default = `HZ/8`)
- `back_seek_max` : Maximum backw ards seek, in KB (Default = 16K)
- `back_seek_penalty` : Penalty for a backw ards seek (Default = 2)

15.11 Deadline Scheduler

Deadline `I/O` scheduler aggressively reorders requests w ith simultaneous goals of improving overall performance + preventing large latencies for individual requests, ie, limiting starvation.

Kernel associates **deadline** w ith each and every request. Read requests = higher priority then w rite requests.

Five separate `I/O` queues maintained:

- Two sorted lists maintained, one for reading, one for writing, arranged by starting block
- Two **FIFO** lists maintained, one for reading, one for writing. Lists sorted by submission time
- Fifth queue contains requests to be shoveled to device driver itself. Called dispatch queue

Art of the algorithm: how requests are peeled off from first four queues and placed on fifth (dispatch queue).

15.12 Deadline Tunables

Available tunables for **Deadline** scheduler:

- `read_expire`: How long (in milliseconds) read request is guaranteed to occur within (Default = `HZ/2 = 500`)
- `write_expire`: How long (in milliseconds) write request is guaranteed to occur within (Default = `5 * HZ = 5000`)
- `writes_starved`: How many requests we should give preference to reads over writes (Default = `2`)
- `fifo_batch`: How many requests should be moved from sorted scheduler list to dispatch queue, when deadlines have expired (Default = `16`)
- `front_merges`: Back merges more common than front merges as contiguous request usually continues to next block. Setting this parameter to 0 disables front merges and can give boost if you know they are unlikely to be needed (Default = `1`)

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 16 Title - Notes

16.3 Learning Objectives:

- Explain the basic filesystem organization.
- Understand the role of the VFS.
- Know what filesystems are available in Linux and which ones can be used on your actual system.
- Grasp why journalling filesystems represent significant advances.
- Discuss the use of special filesystems in Linux.

16.4 Filesystem Basics

Application programs -> read/write files, rather than dealing with physical locations on actual hardware (where files are stored).

Files (and their names) -> abstraction camouflaging physical I/O layer. Directly writing to disk in **raw** fashion (ignoring **filesystem** layer)-> *very dangerous*, only done by low-level operating system software, never by user application.

Local filesystems generally reside within disk partition (which can be physical partition on disk, or logical partition controlled by **Logical Volume Manager** (LVM)). Filesystems can also be part of network nature, true physical embodiment completely hidden to local system across network.

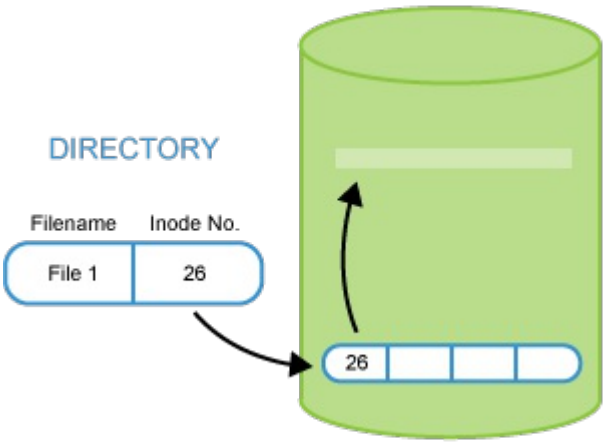
16.5 Inodes

Inode: data structure on disk that describes and stores file attributes, including location. Every file associated with own inode. Information stored in inode:

- Permissions
- User and group ownership
- Size
- Timestamps (nanoseconds)
  - Last access time
  - Last modification time
  - Change time

Note: File names **not** stored in file's inode, but instead stored in **directory** file.

All I/O activity concerning file usually also involves file's inode, as information must be updated.



Data storage in an inode vs. data storage in a directory file

16.6 Hard and Soft Links

Directory file: particular type of file used to associate file names and inodes. Two ways to associate (or link) file name with inode:

- **Hard** links point to an inode
- **Soft** (or **symbolic**) links point to a file name which has an associated inode

Link: each association of directory file contents and inode. Additional links can be created using **ln**.

Because possible (+ quite common) for two or more directory entries to point to same inode (hard links), a file can be known by multiple names, each of which has its own place in directory structure. However, can only have one inode, no matter which name being used.

When process refers to pathname, kernel searched directories to find corresponding inode number. After name converted to inode number, inode loaded into memory + used by subsequent requests.

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>touch file
c7:/tmp>ln -s file file-soft
c7:/tmp>ln    file file-hard
c7:/tmp>ls -li file*
976710 -rw-rw-r-- 2 coop coop 0 Nov 19 09:33 file
976710 -rw-rw-r-- 2 coop coop 0 Nov 19 09:33 file-hard
976739 lrwxrwxrwx 1 coop coop 4 Nov 19 09:34 file-soft -> file
c7:/tmp>
```

16.7 Filesystem Tree organization

All Linux systems use inverted tree hierarchy branching off the root (`/`) directory. While entire tree may be contained in one local filesystem in one partition, usually there are multiple partitions (or network filesystems) joined together at **mount points**. Can also include removable media, eg. USB drives, optical drives, etc.

In addition, certain **virtual pseudo filesystems** (useful abstractions which exist only in memory) will be mounted within tree. Include `/proc` , `/sys` , `/dev` , maybe `/tmp` and `/run` .

Each element mounted within tree *may* have own filesystem variety. But, to applications and operating system, all appears in one unified tree structure.

16.8 Virtual File System (VFS)

Linux implements Virtual File System (VFS), as do all modern operating systems. When application needs to access file, interacts with VFS abstraction layer, which then translates all I/O system calls (reading, writing, etc.) into specific code relevant to particular filesystem.

Thus, applications do not need to consider neither specific actual filesystem or physical media/hardware on which it resides, and network filesystems (such as NFS) can be handled transparently.

This permits Linux to work with more filesystem varieties than any other operating system. Democratic attribute has been large factor in success.

Most filesystems have full read/write access, while few have only read access, perhaps experimental write access. Some filesystem types, especially non-UNIX based ones, may require more manipulation in order to be represented in VFS.

Variants such as **vfat** do not have distinct read/write/execute permissions for owner/group/world fields. VFS has to make assumption about how to specify distinct permissions for the three types of user, such behavior can be influenced by mounting operations. There are non-kernel filesystem implementations, eg. read/write ntfs_3g, which are reliable but have weaker performance than in-kernel filesystems.

16.9 Available Filesystems

Linux supports many filesystem varieties, most with full read/write access:

- **ext4**: Linux native filesystem (and earlier ext2 and ext3)
- **XFS**: high-performance filesystem originally created by SGI
- **JFS**: high-performance filesystem originally created by IBM
- Windows-nativesL **FAT12**, **FAT16**, **FAT32**, **VFAT**, **NTFS**
- **Pseudo-filesystem**s resident only in memoery, including **proc**, **sysfs**, **devfs**, **debugfs**
- Network filesystems such as **NFS**, **coda**, **afs**
- etc.

Democratic flexibility large factor in success. Most filesystems have full read/write access, while few have read only access.

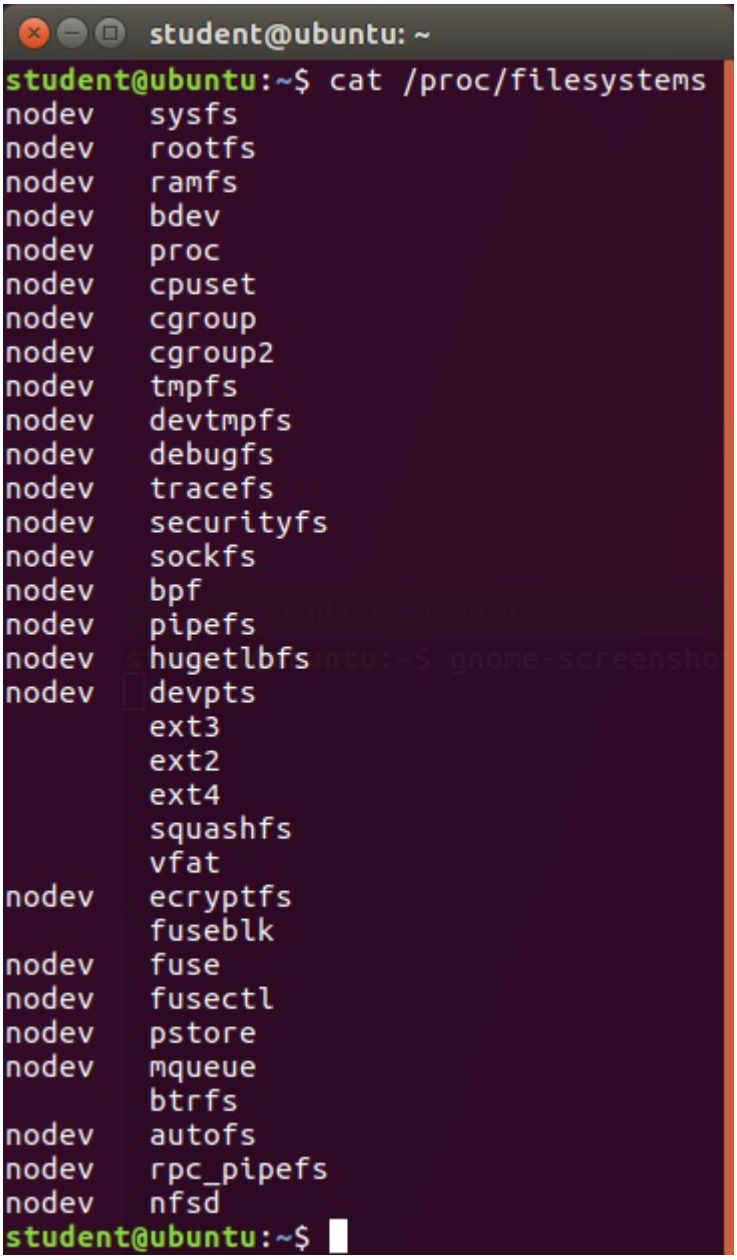
Commonly used filesystems include **ext4**, **xfs**, **btrfs**, **squashfs**, **nfs**, **vfat**. A list of currently supported filesystems at `/proc/filesystems`.

16.10 Current Filesystem Types

Can see list of filesystem types currently registered + understood by current running Linux kernel by doing:

```
$ cat /proc/filesystems
```

Note: additional filesystem types may have their code loaded only when system tries to access a partition that uses them.



16.12 Journaling Filesystems

Number of newer, high performance filesystems include full **journalling** capability.

Journalling filesystems recover from system crashes/ungrateful shutdowns with little or no corruption + very rapidly. Comes at price of having some more operations to do, but additional enhancements can more than offset price.

In journalling filesystem, operations grouped into **transactions**. Transaction must be completed without error, **atomically**. Otherwise, filesystem not changed. Log file maintained of transactions. When error occurs, usually only last transaction needs to be examined.

Following journalling filesystems freely available under Linux:

- **ext3**: extension of earlier non-journalling ext2 filesystem
- **ext4**: vastly enhanced outgrowth of ext3. Features include extents, 48-bit block numbers, and up to 16TB size. Most linux distributions have used ext4 as default filesystem for quite a few years
- **reiserfs**: first journalling implementation used in Linus, but lost its leadership and further development was abandoned
- **JFS**: originally product of IBM, was ported from IBM's AIX operating system

- **XFS**: originally product of SGI, was ported from SGI's IRIX operating systems. RHEL 7 has adopted XFS as default filesystem
  btrfs: newest of journalling filesystems, still under rapid development

16.13 Special Filesystems

Linux widely employs use of **special filesystems** for certain tasks. Particularly useful for accessing various kernel data structures + tuning kernel behavior, or for implementing particular functions. Note: some of these special filesystems have no mount point. This means user applications don't interact with them, but kernel uses them, taking advantage of VFS layers + code.

Special Filesystems

Filesystem	Mount Point	Purpose
rootfs	None	During kernel load, provides an empty root directory
hugetlbfs	Anywhere	Provides an extended page access (2 or 4 MB on **X86**)
bdev	None	Used for block devices
proc	`/proc`	Pseudo filesystem access to many kernel structures and subsystems
sockfs	None	Used by **BSD Sockets**
tmpfs	Anywhere	RAM disk with swapping, re-sizing
shm	None	Used by System V IPC Shared Memory
pipefs	None	Used for pipes
binfmt_misc	Anywhere	Used by various executable formats
devpts	`/dev/pts`	Used by **Unix98** pseudo-terminals
usbfs	`/proc/bus/usb`	Used by **USB** sub-system for dynamical devices
sysfs	`/sys` (or elsewhere)	Used as a **device tree**
debugfs	`/sys/kernel/debug` (or elsewhere)	Used for simple debugging file access

##

Back to top

Chapter 17 Disk Partitioning - Notes

17.3 Learning Objectives:

- Describe and contrast the most common types of hard disks and data buses.
- Explain disk geometry and other partitioning concepts.
- Understand how disk devices are named and how to identify their associated device nodes.
- Distinguish among and select different partitioning strategies.
- Use utilities such as **fdisk**, **blkid**, and **lsblk**.
- Back up and restore partition tables.

17.4 Common Disk Types

Number of different hard disk types, each characterized by type of **data bus** they are attached to, and other factors such as speed, capacity, how well multiple drives work simultaneously, etc.:

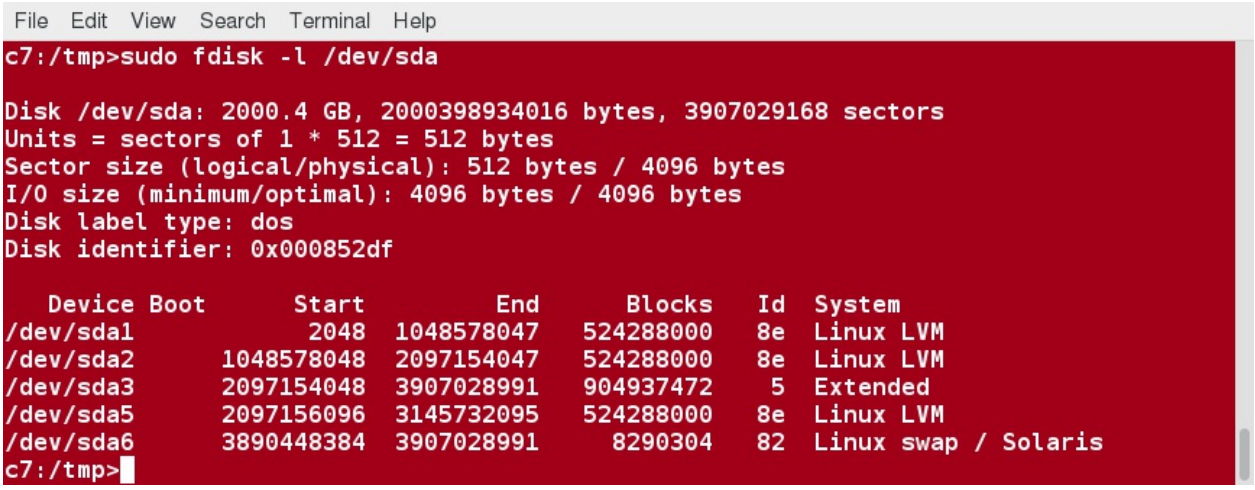
- **SATA** (Serial Advanced Technology Attachment): designed to replace old IDE drives. Offer smaller cable size (7 pins), native hot swapping, faster/more efficient data transfer. Seen as SCSI devices.
- **SCSI** (Small Computer Systems Interface): SCSI disks range from narrow (8 bit bus) to wide (16 bit bus), with transfer rate between 5MB per second (narrow, standard SCSI) and 160MB per second (Ultra-wide SCSI-3). Most PCs use single-ended or differential SCSI drives. Unfortunately, the two types not compatible with each other. Fortunately, the two types of devices may coexist on same controller. Single-ended devices may host up to 7 devices, and use maximum cable length of about 6 meters. Differential controllers may host up to 15 devices, and have maximum cable length of about 12 meters.
- **SAS** (Serial Attached SCSI): use newer point-to-point protocol, have better performance than SATA disks.
- **USB** (Universal Serial Bus): include flash drives and floppies. Seem as SCSI devices.
- **SSD** (Solid State Drives): modern SSD drives have come down in price, have no moving parts, use less power than drives with rotational media, have faster transfer speeds. Internal SSDs installed with same form factor and in same enclosures as conventional drives. SSDs still cost a bit more, but price decreasing. Common to have both SSDs and rotational drives in same machines, with frequently accessed + performance critical data transfers taking place on SSDs.
- **IDE and EIDE** (Integrated Drive Electronics, Enhanced IDE): obsolete.

17.5 Disk Geometry

Disk Geometry: concept with long history for rotational media. One talks of **heads**, **cylinders**, **tracks**, **sectors**. Below shows how to view geometry with **fdisk**:

```
$ sudo fdisk -l /dev/sda
```

Note use of `-l` option, which simply lists partition table without entering interactive mode.



Rotational disks composed of one or more platters, each read by one or more **heads**. Heads read circular **track** off platter as disk spins.

Circular tracks divided into data blocks called **sector**, typically 512 bytes in size. **Cylinder**: group which consists of the same track on all platters.

Physical structure image has been less and less relevant as internal electronics on drive actually obscure much of it. SSDs have no moving parts or anything like above ingredients.

Currently, disks starting to be manufactured with sectors larger than 512 bytes; 4KB is becoming available. While larger sector sizes can lead to faster transfer speeds, operating system support not yet mature in dealing with larger sizes.

17.6 Partition Organization

Disks divided into **partitions**. In geometrical terms, these consist of physically contiguous groups of sectors or cylinders. Partition: physically contiguous region on disk. Two partitioning layouts in use:

- MBR (Master Boot Record)
- GPT (GUID Partition Table)

MBR dates back to early days of MSDOS. When using MBR, disk may have up to four **primary** partitions. One of the primary partitions can be designated as an extended partition, which can be subdivided further into logical partitions with 15 partitions possible.

When using MBR scheme, if we have a SCSI, eg. `/dev/sda`, then `/dev/sda1`: first primary partition, and `dev/sda2`: second primary partition. If we created an extended partition `dev/sda3`, could be divided into logical partitions. All partitions greater than four -> logical partitions (meaning contained within extended partition). Can only be one extended partition, but it can be divided into numerous logical partitions.

Note: Linux doesn't require partitions to begin or end on cylinder boundaries, but other operating systems might complain if they don't. For this reason, widely deployed Linux partitioning utilities try to play nice and end on boundaries. Obviously, partitions should not overlap either.

GPT: on all modern systems, based on UEFI (Unified Extensible Firmware Interface). By default, may have up to 128 primary partitions. When using GPT scheme, no need for extended partitions. Partitions can be up to 2^{33} TB in size (with MBR, limit is just 2TB).

17.7 Why Partition?

Multiple reasons why it makes sense to divide system data into multiple partitions:

- Separation of user and application data from operating system files
- Sharing between operating systems and/or machines
- Security enhancement by imposing different quotas and permissions for different system parts
- Size concerns; keeping variable and volatile storage isolated from stable
- Performance enhancement of putting most frequently used data on faster storage media
- Swap space can be isolated from data and also used for hibernation storage

Deciding what to partition and how to separate partitions: cause for thought. Reasons to have distinct partitions: increased granularity of security, quote, settings, size restrictions. Could have distinct partitions to allow for data protection.

Common partition layout contains a **boot** partition, a root filesystem `/`, a swap partition, and a partition for `/home` directory tree.

Note: more difficult to resize partition after the fact, than during install/creation time. Plan accordingly.

17.8 MBR Partition Table

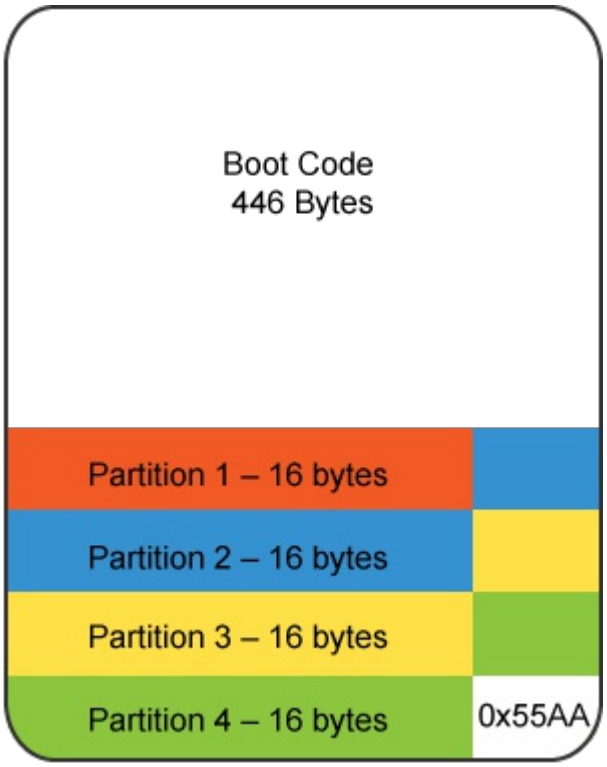
Disk partition table contained with disk's **M**aster **B**oot **R**ecord (**MBR**), is the 64 bytes following the 446 byte boot record. One partition on a disk may be marked active. When system boots, that partition is where MBR looks for items to load.

Note: can only be one extended partition, but that partition may contain number of logical partitions.

Structure of MBR defined by operating system-independent convention. First 446 bytes: reserved for the program code, typically hold part of a boot loader program. Next 64 bytes: provide space for partition table of up to four entries. Operating system needs this table for handling the hard disk.

On Linux systems, beginning and ending address in CHS ignored.

Note for curious: there are 2 more bytes at the end of the MBR known as the magic number, signature word, or end of sector marker, which always have the value `0x55AA`.



Disk Partition Table

Each entry in partition table -> 16 bytes long, describes one of the four possible primary partitions. Information for each:

- Active bit
- Beginning address in cylinder/head/sectors (**CHS**) format (ignored by Linux)
- Partition type code, indicating: **xfs**, **LVM**, **ntfs**, **ext4**, **swap**, etc.
- Ending address in CHS (also ignored by Linux)
- Start sector, counting linearly from 0
- Number of sectors in partition

Linux only uses last two fields for addressing, using the linear block addressing (**LBA**) method.

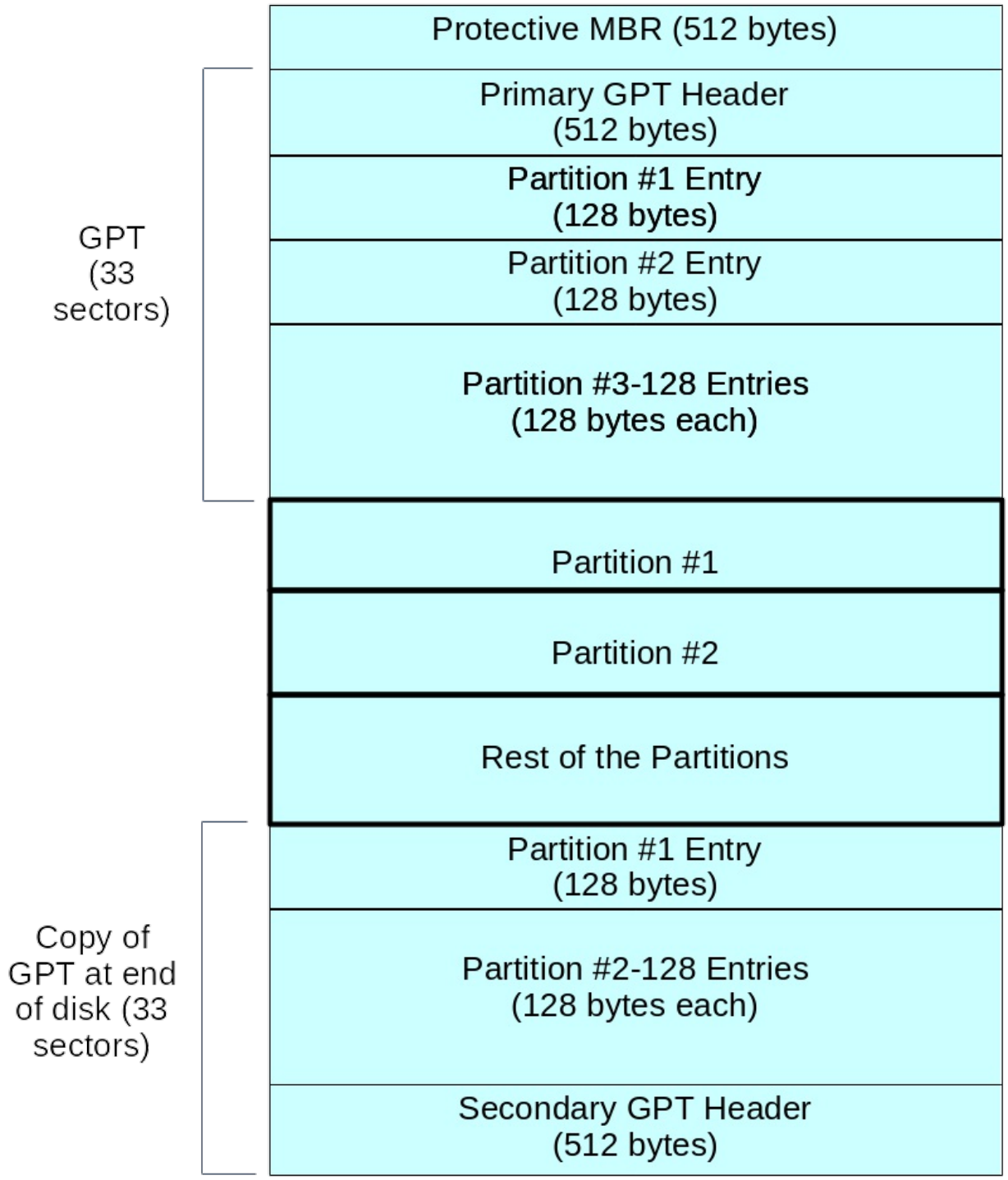
17.9 GPT Partition Table

Modern hardware comes with GPT support; MBR support will gradually fade away.

The Protective MBR is for backwards compatibility, so UEFI systems can be booted the old way,

There are two copies of the GPT header, at the beginning + at the end of the disk, describing metadata:

- List of usable blocks on disk
- Number of partitions
- Size of partition entries. Each partition entry has minimum size of 128 bytes



blkid utility (to be discussed later) show information about partitions.

On modern UEFI/GPT system:

```
ROOT@x7:/root>blkid /dev/sda6
/dev/sda6: LABEL="CENTOS7" UUID="77461ee7-c34a-4c5f-b0bc-29f4feecc743" TYPE="ext4" PARTUUID="1f361af4-81e6-4a81-82"
ROOT@x7:/root>
```

On legacy MBR system:

```
c7:/teaching/LFCW/LFS301>sudo blkid /dev/sdb1
/dev/sdb1: LABEL="RHEL7" UUID="471dfeba-3ec7-4529-8069-2afe50762c57" TYPE="ext4"
```

Note: both examples give unique `UUID`, which describes filesystem on partition, not the partition itself. Changes if filesystem reformatted.

GPT partition also gives `PARTUUID` which describes partition and stays the same even if filesystem reformatted. If hardware supports it, possible to migrate MBR system to GPT, but not hard to *brick* machine while doing so.

Thus, usually benefits not worth the risk.

17.10 Naming Disk Devices and Nodes

Linux kernel interacts at low level with disks through device nodes normally found in `/dev` directory. Normally, device nodes accessed only through infrastructure or kernel's Virtual File System. Raw access through device nodes -> extremely efficient way to destroy filesystem. For example, you do this when formatting a partition:

```
$ sudo mkfs.ext4 /dev/sda9
```

Device nodes for SCSI and SATA disks follow simple `xxy[z]` naming convention, where `xx` is device type (usually `sd`), `y` is the letter for drive number (`a`, `b`, `c`, etc.), and `z` is partition number:

- First hard disk is `/dev/sda`
- Second hard disk is `/dev/sdb`
- Etc.

Partitions also enumerated:

- `/dev/sdb1`: first partition on second disk
- `/dev/sdc4`: fourth partition on third disk

In above, `sd` means SCSI or SATA disk. Back when IDE disks could be found, they would have been `/dev/hda3`, `/dev/hdb` etc.

Doing `ls -l /dev` will show current available disk device nodes.

17.11 More on SCSI Device Names

For SCSI devices, need to elaborate a little more on what first, second hard disk etc. means. Determined by controller number/ID number combination.

Drive designation (`a`, `b`, `c`, etc.) primarily based on ID number of SCSI device, rather than position on bus itself.

Eg. if two SCSI controllers with target ID number 1 and 3 on controller 0, and target ID number 2 and 5 on controller 1 (with ID 2 as last drive):

- ID 1: `/dev/sda`
- ID 3: `/dev/sdb`
- ID 2 (on controller 1): `/dev/sdc`
- ID 5: `/dev/sdd`

17.12 blkid

blkid: utility to locate block devices and report on attributes. Works with **libblkid** library. Can take as an argument a particular device of list of devices. Image below shows use of **blkid** with arguments:

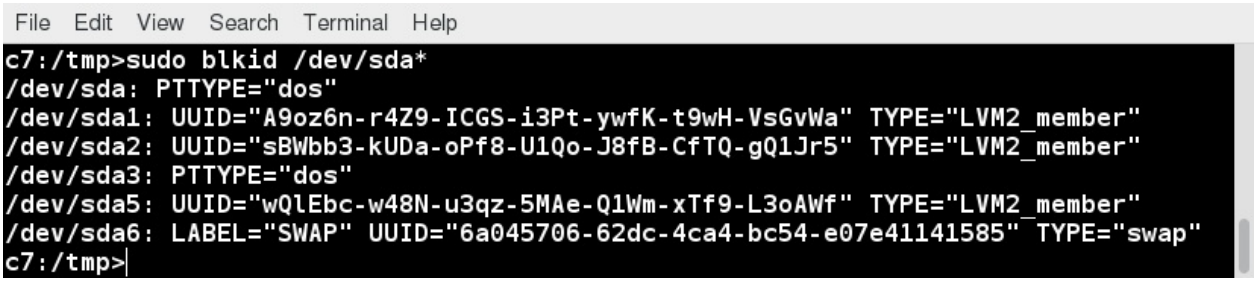
```
$ sudo blkid /dev/sda*
```

Can determine type of content (eg. filesystem, swap) a block device holds, and also attributes (tokens, `NAME=value` pairs) from content metadata (eg. `LABEL` or `UUID` fields).

Will only work on devices which contain data that is finger-printable: eg. empty partition will not generate block-identifying **UUID**.

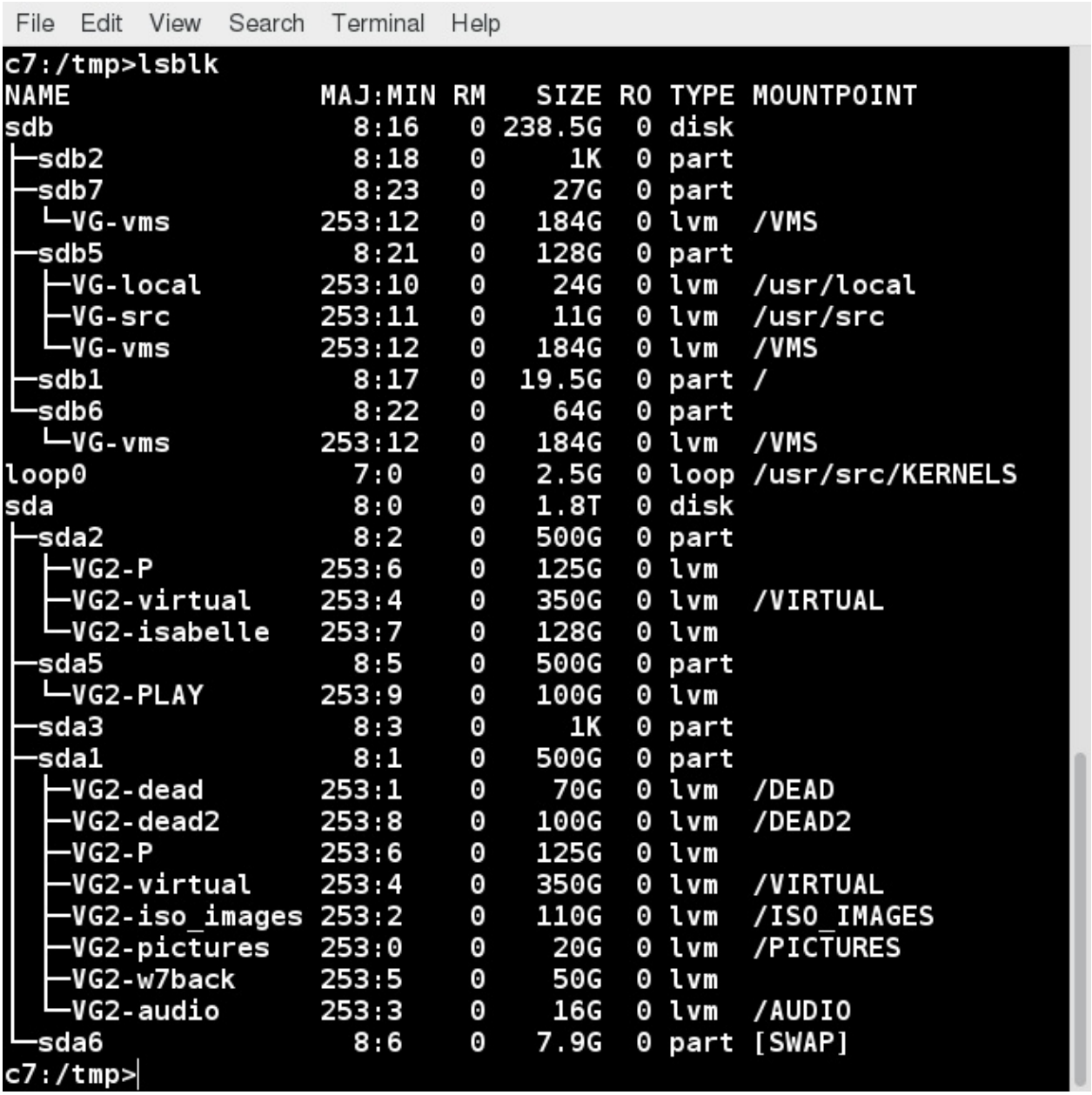
Has two main forms of operations: either searching for a device with specific `NAME=value` pair, or displaying `NAME=value` pairs for one or more devices. Without arguments, will report on all devices. Quite a few options designating how to specify devices and what attributes to report on. Other sample commands:

```
$ sudo blkid
$ sudo blkid -L root
```



17.13 lsblk

lsblk: related utility which presents block device information in tree format, as shown below:



17.14 Sizing Up Partitions

Most Linux systems should use ***minimum*** of two partitions:

- **/ (root)**: used for the entire logical filesystem
  - In practice, most installations will have more than one filesystem on more than one partition, which are joined together at **mount points**
  - Difficult with most filesystem types to resize the root partition. Using **LVM** (discussed later), can make this easier.
  - While certainly possible to run Linux with just the root partition, most systems use more partitions to allow for easier backups, more efficient use of disk drives, and better security.
- **Swap**: used as extension of physical memory. Pages of memory which are not file-backed can be moved to disk until needed again.
  - Usual recommendation: swap size should be equal to physical memory in size. Sometimes, twice that recommended. However, correct choice depends on related issues of system use scenarios, hardware capabilities. Examples of thinking on this subject can be found at https://help.ubuntu.com/community/SwapFaq and https://www.suse.com/support/kb/doc/?id=7010157.
  - System may have multiple swap partitions and/or swap files.
  - On a single disk system, try to center the swap partition. On multiple disk systems, try to spread swap over disks.

- Adding more and more swap will necessarily not help because at a certain point it becomes useless. One will need to either add more memory or re-evaluate system setup.

Swap used as **virtual memory**: any time pages from processes are moved out of physical memory, generally stored on swap device. Recent Ubuntu distributions are now placing swap in a file rather than a partition by default.

17.15 Backing Up and Restoring Partition Tables

Partitioning and re-partitioning disks are dangerous operations. Need to know how to back up/restore partition tables in order to restore the situation if something goes wrong.

Backing up can be done easily with **dd**:

```
$ sudo dd if=/dev/sda of=mbrbackup bs-512 count=1
```

which will back up **MBR** on first disk, including 64-bit partition table which is part of it.

MBR can then be restored, if necessary, by doing:

```
$ sudo dd if=mbrbackup of=/dev/sda bs=512 count=1
```

Note: above commands only copy primary partition table. Do not deal with any partition tables stored in other partitions (for extended partitions, etc.).

Note: should *always* assume that changing disk partition table might eliminate all data in all filesystems on disk (It should not, but be cautious!). Always prudent to make backup of all data (that is not already backed up) before doing any of this type of work.

In particular, must be careful in using **dd**: simple typing error or misused option could *destroy* entire disk. Hence, do backups!!!

For GPT systems, best to use **sgdisk** tool:

```
x7:/tmp>sudo sgdisk --backup=/tmp/sda_backup /dev/sda
The operation has completed successfully.
x7:/tmp>sudo file sda_backup
sda_backup: x86 boot sectorl partition 1: ID=0xee, starthead 0, startsector 1, 1000215215 sectors, extended partition tabl
code offset 0x63
```

Note if run on a pure MBR system:

```
x7:/tmp>sudo sgdisk --backup=/tmp/sda_backup /dev/sda
**********************************************
Found Invalid GPT and valid MBR; converting MBR to GPT format in memory.
**********************************************
The operation has completed successfully.
x7:/tmp>sudo file sda_backup
/tmp/sda_backup: x86 boot sectorl partition 1: ID=0xee, starthead 0, startsector 1, 3907029167 sectors, code offset 0xb8
```

17.16 Partition Table Editors

Number of utilities which can be used to manage partition tables:

- **fdisk**: menu-driven partition table editor. Most standard and one of the most flexible partition table editors. As with any other partition table editor, make sure to write down current partition table settings or make copy of current settings before making changes
- **sfdisk**: non-interactive Linux-based partition editor program, making it useful for scripting. Use **sfdisk** tool with care!
- **parted**: **GNU** partition manipulation program. Can create, remove, resize, move partitions (including certain filesystems). GUI interface to **parted** command is **gparted**
- **gparted**: widely used graphical interface to **parted**
- **gdisk**: used for GPT systems, but can also operate on MBR systems
- **sgdisk**: script or command line interface

Many Linux distributions have live/installation version which can be run off either CD-ROM or USB stick. These media usually include a copy of **gparted**, so can easily be used as graphical partitioning tool on disks which are not actually being used while partitioning program is run.

17.16 Using fdisk

fdisk always included in any Linux installation, so good idea to learn how to use it. Must be root to run **fdisk**. Can be somewhat complex to handle, caution is advised.

fdisk interface: simple and text-driven menu. After starting on particular disk:

```
$ sudo fdisk /dev/sdb
```

main (one letter) commands are:

- `m`: Display the menu
- `p`: List the partition table
- `n`: Create a new partition
- `d`: Delete a partition
- `t`: Change a partition type
- `w`: Write the new partition table information and exit
- `q`: Quit without making changes

Fortunately, no actual changes are made until you write the partition table to the disk by entering `w`. Therefore important to verify partition table is correct (with `p`) before writing to disk with `w`. If something wrong, can jump out safely with `q`.

System will not use new partition table until reboot. However, can use following command:

```
$ sudo partprobe -s
```

to try and read in revised partition table. However, this doesn't always work reliably and it is best to reboot before doing things like formatting new partitions, etc. as mixing up partitions can be catastrophic.

Any anytime, can do:

```
$ cat /proc/partitions
```

to examine what partitions the operating system is currently aware of.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 18 Filesystem Features: Attributes, Creating, Checking, Mounting - Notes

18.3 Learning Objectives:

- Explain concepts such as inodes, directory files, and extended attributes.
- Create and format filesystems.
- Check and fix errors on filesystems.
- Mount and unmount filesystems.
- Use network file systems.
- Understand how to automount filesystems and make sure they mount at boot.

18.4 Extended Attributes and lsattr/chattr

Extended Attributes associate metadata not interpreted directly by filesystem with files. Four namespaces exist:

- user
- trusted
- security
- system

System namespace used for **Access Control Lists** (**ACL**s) and security namespace used by SELinux.

Flag values stored in file inode, maybe modified and set only by root user. Viewed with **lsattr** and set with **chattr**. Flags may be set for files:

- `i`: Immutable. File with immutable attribute cannot be modified (not even my root). Cannot be deleted or renamed. No hard link can be created to it, and no data can be written to file. Only superuser can set or clear this attribute.
- `a`: Append-only. FIle with append-only attribute set can only be opened in append mode for writing. Only superuser can set ot clear this attribute.
- `d`: No-dump. File with no-dump attribute set is ignored when the **dump** program is run. Useful for swap and cache files that you don't want to waste time backing up.
- `A`: No **atime** update. File with no-atime-update attribute set will not modify its **atime** (access time) record when file accessed but not otherwise modified. Can increase performance on some systems because it reduces amount of disk I/O on system.

Note: there are other flags that can be set; typing `man chattr` will show whole list. Format for **chattr**:

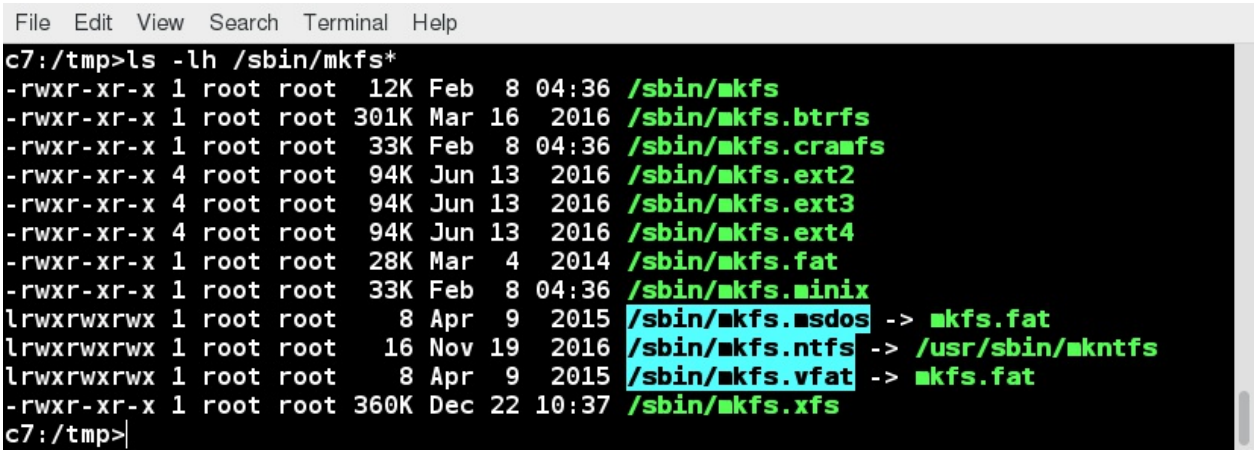
```
$ chattr [+|-|=mode] filename
```

lsattr used to display attributes for file:

```
$ lsattr filename
```

18.5 Creating and Formatting Filesystems

Every filesystem type has utility for **formatting** (making) filesystem on partition. Generic name for these utilities: **mkfs**. However, this is just frontend for filesystem-specific programs, each of which may have particular options.



Thus, following two commands entirely equivalent:

```
$ sudo mkfs -t ext4 /dev/sda10
$ sudo mkfs.ext4 /dev/sda10
```

General format for **mkfs**:

```
mkfs [-t fstype] [options] [device-file]
```

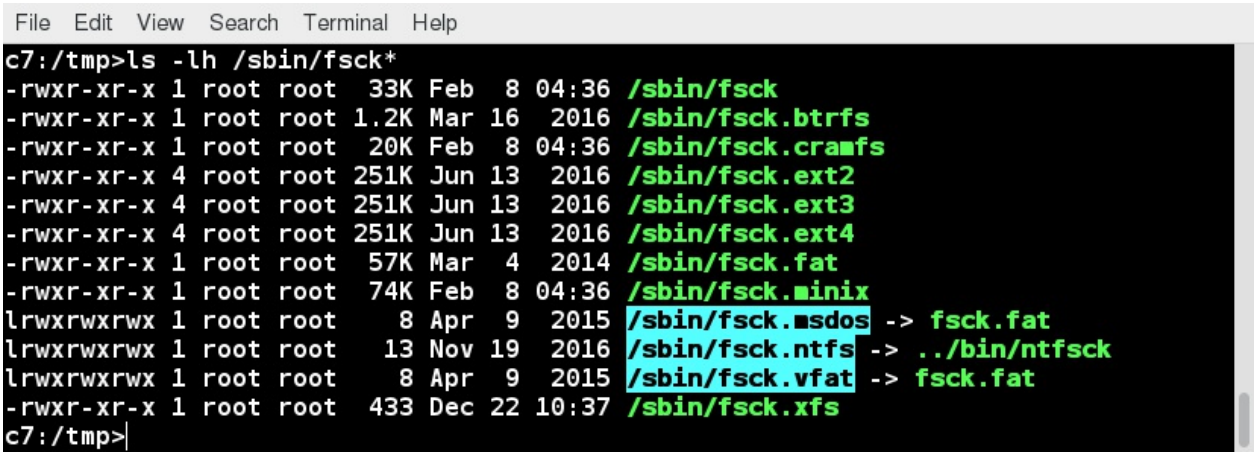
where `[device-file]` is usually device name like `/dev/sda3` of `/dev/vg/lvm1`.

Each filesystem has own particular options that can be set when formatting. Eg. when creating **ext4** filesystem, journalling settings = one thing to keep in mind. Include defining journal file size, whether or not to use external journal file.

Should look at **man** page for each of **mkfs.*** programs to see details.

18.6 Checking an Repairing Filesystems

Every filesystem type has utility designed to check for errors (and hopefully fix any that are found). Generic name for these utilities: **fsck**. However, just frontend for filesystem-specific programs.



Thus, following two commands entirely equivalent:

```
$ sudo fsck -t ext4 /dev/sda10
$ sudo fsck.ext4 /dev/sda10
```

If filesystem is of type understood by operating system, can almost always just do:

```
$ sudo fsck /dev/sda10
```

and system will figure out type by examining first few bytes on partition.

fsck run automatically after set number of mounts, or set interval since last time it was run, or after abnormal shutdown. Should only be run on unmounted filesystems. Can force a check of all ounted filesystems at boot by doing:

```
$ sudo touch /forcefsck
$ sudo reboot
```

The file `/forcefsck` will disappear after successful check. One reason this is valuable trick: can do **fsck** on root filesystem, which is hard to do on running system.

General format for **fsck**:

```
fsck [-t fstype] [options] [device-file]
```

where `[device-file]` is usually device name like `/dev/sda3` or `/dev/vg/lvm1`. Usually, do not need to specify filesystem type, as **fsck** can figure it out by examining superblocks at start of partition.

Can control whether any errors found should be fixed one by one manually with `-r` option, or automatically, as best possible, by using `-a` option, etc. In addition, each filesystem type may have own particular options that can be set when checking.

Note: **journalling** filesystems much faster to check than older generation filesystems for two reasons:

- Rarely need to scan entire partition for errors, as everything but very last transaction logged and confirmed, so takes almost no time to check
- Even if whole filesystem checked, newer filesystems designed with **fast fsck** in mind. Older filesystems did not think much about this when designed as sizes were much smaller

Should look at **man** page for each of **fsck.*** programs to see details.

18.7 Mounting Filesystems

All accessible files in Linux organized into one large hierarchical tree structure with head of tree being root directory (`/`). However, common to have more than one partition (each of which can have own filesystem type) joined together in same filesystem tree. Partitions can also be on different physical devices, even on network.

mount program allows attaching at any point in tree structure. **umount** allows detaching them.

Mount point is directory where filesystem attached. Must exist before **mount** can use it. **mkdir** can be used to create expty directory. If pre-existing directory used + contains files prior to being used as mount point, files will be hidden after mounting. These files are *not* deleted and will again be visible when filesystem unmounted.

By default, only superuser can mount/unmount filesystems.

Each filesystem mounted under specific directory:

```
$ sudo mount -t ext4 /dev/sdb4 home
```

- Mounts **ext4** filesystem
- Usually not necessary to specify type with `-t` option
- Filesystem is located on specific partition of hard drive (`/dev/sdb4`)
- Filesystem mounted at position `/home` in current directory tree
- Any files residing in original `/home` directory hidden until partition unmounted

18.10 mount

General form for mount:

```
mount [options] <source> <directory>
```

Note: in this example, filesystem mounted by using device node it resides on. However, also possible to mount using label or **UUID**. Thus, following all equivalent:

```
$ sudo mount /dev/sda2  /home
$ sudo mount LABEL=home /home
$ sudo mount    -L home /home
$ sudo mount UUID=26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
$ sudo mount   -U 26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
```

Labels assigned by filesystem type specific utilities, such as **e2label**, and **UUIDs** assigned when partitions created as containers for filesystem, formatted with **mkfs**.

While any of these three methods for specifying device can be used, modern systems deprecate using device node form because names can change according to how the system is booted, which hard drives found first, etc. Labels = improvement, but on rare occasions, could have two partitions that wind up with same label. **UUIDs**, however, should always be unique + created when partitions created.

18.11 mount Options

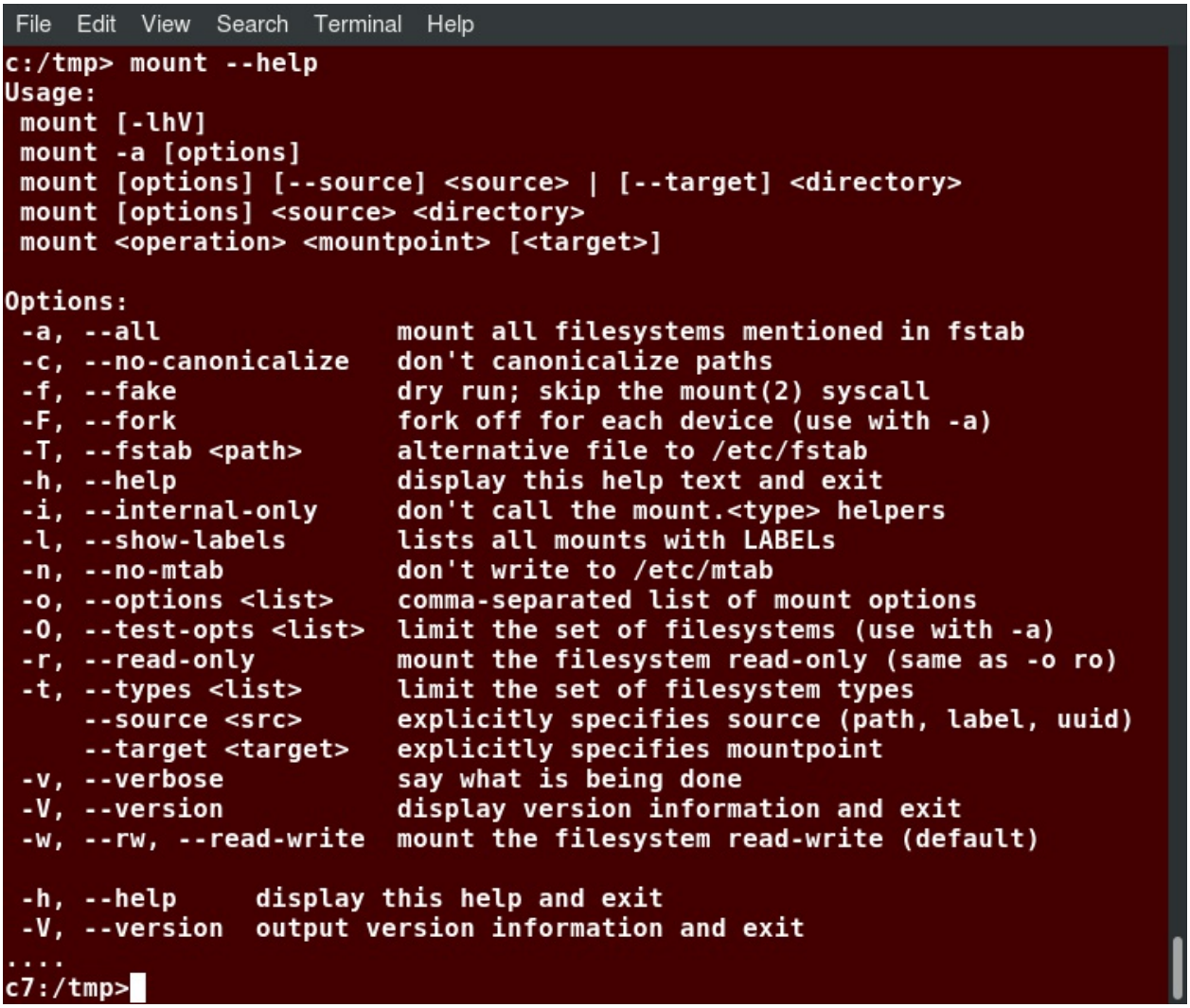
mount takes many options, some generic like `-a` (mount all filesystems mentioned in `/etc/fstab`) and many filesystem specific. Has very long **man** page. Common example:

```
$ sudo mount -o remount,ro /myfs
```

which remounts filesystem with **read-only** attribute.

An exampe of hot to get quick summary of mount options:

```
$ mount --help
```



18.12 umount

Filesystems can be unmounted:

```
$ umount [device-file | mount-point]
```

Below, some examples of how to unmount filesystem:

- Unmount `/home` filesystem:

```
$ sudo umount /home
```

- Unmount the `/dev/sda3` device:

```
$ sudo umount /dev/sda3
```

Note: command to unmount filesystem is **umount** (*not* **unmount**!).

Like **mount**, **umount** has many options, many of which specific to filesystem type. **man** pages = best source for specific option

information.

Most common error encountered when unmounting filesystem: trying to do this on filesystem currently in use, ie. current applications using files or other entries in filesystem.

Can be as simple as having terminal window open in directory on mounted filesystem. Just using **cd** in that window, or killing it, will get rid of `device is busy` error and allow unmounting.

However, if other processes inducing this error, must kill those processes before unmounting filesystem. Can use **fuser** to find out which users using filesystem and kill them (be careful with this, may also want to warn users first). Can also use **lsof** ("list open files") to try and see which files being used and blocking unmounting.

18.13 Network Shares (NFS)

Common to mount remote filesystems through network shares so they appear as if they were on local machine. Probably most common method used historically: **NFS** (**N**etwork **F**ile **S**ystem).

NFS originally developed by Sun Microsystems in 1989, has been continuously updated. Modern systems use **NFSv4**, which has been continuously updated since 2000.

Other network filesystems include **AFS** (**A**ndrew **F**ile **S**ystem), and **SMB** (**S**erver **M**essage **B**ook), also termed **CIFS** (**C**ommon **I**nternet **F**ile **S**ystem).

Because network filesystem may be unavailable at any time, either because it is not present on network share, or the network is unavailable, systems have to be prepared for this possibility.

Thus, in such circumstances, system should be instructed not to get hung, or blocked, while waiting longer than specified period. Can be specified in **mount** command:

```
$ sudo mount -t nfs myserver.com:/shdir /mnt/shdir
```

or in `/etc/fstab`. Put following line in `/etc/fstab` to mount on boot or with `mount -a`:

```
myserver.com:/shdir /mnt/shdir nfs rsize=8192,wsize=8192,timeo=14,intr 0 0
```

System may try to mount **NFS** filesystem before network is up. `netdev` and `noauto` options can be used. For more information, check **man nfs**, examine **mount** options.

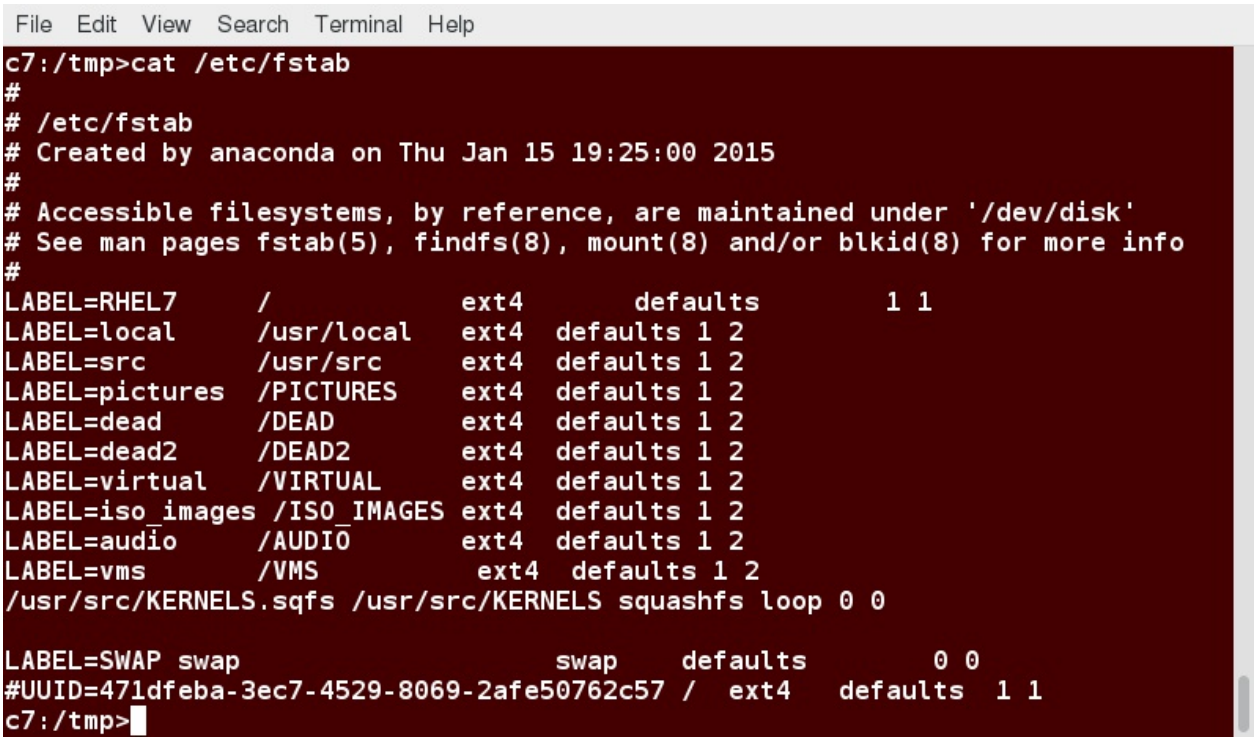
Can also be solved using **autofs** or **automount**.

Mount has large amount of options, some of which specific to **nfs**. See **man** pages for both **nfs** and **mount** for details.

18.14 Mounting at Boot

During system initialization, the command `mount -a` executed. Mounts all filesystems listed in `/etc/fstab` configuration file. Entries can refer to both local + remote network-mounted filesystems. Below shows example of how to mount all filesystems listed in `/etc/fstab` configuration file during system boot.

File shows what filesystems may be automatically mounted at boot, and where they may be found on the local machine or network. Can specify who may mount them and with what permissions, and other relevant options. Some lines refer to special pseudo-filesystems such as **proc**, **sys**, **devpts**.



Each record in `/etc/fstab` file contains w hite space separated fields of information about filesystem to be mounted:

- Device node, label, or UUID

  For filesystems w hich do not have device node, such as **tmpfs**, **proc**, and **sysfs**, this field is just placeholder. Sometimes w ill see the w ord *none* in that column, or used on command line.

- Mount point

  Can also be placeholder, like for **swap**, w hich is not mounted anyw here.

- Filesystem type (ie. **ext4**, **xfs**, **btrfs**, **vfat**)

- A comma-separated list of options

- **dump** frequency (or a 0)

  Used by rarely used `dump -w` command.

- **fsck** pass number (or 0, meaning do not check state at boot)

mount and **umount** utilities can use information in `/etc/fstab`. In such case, could type:

```
$ sudo mount /usr/src
```

instead of

```
$ sudo mount LABEL=src /usr/src
```

in above example.

18.15 Automatic Filesystem Mounting

Linux systems have long had ability to mount filesystem only when filesystem needed. Historically, done using **autofs**. This utility requires installation of **autofs** package using appropriate package manager + configuration of files in `/etc`.

While **autofs** very flaxible + well understood, systemd-based systems (including all recent enterprise Linux distributions) come with **automount** facilities built into **systemd** framework. Configuring this = simple as adding a line in `/etc/fstab` specifying proper device, mount point, mount options:

```
LABEL=Sam128 /SAM ext4 noauto,x-systemd.automount,x-systemd.device-timeout=10,x-systemd.idle-timeout=30 0 0
```

and then, either rebooting or issuing command:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart local-fs.target
```

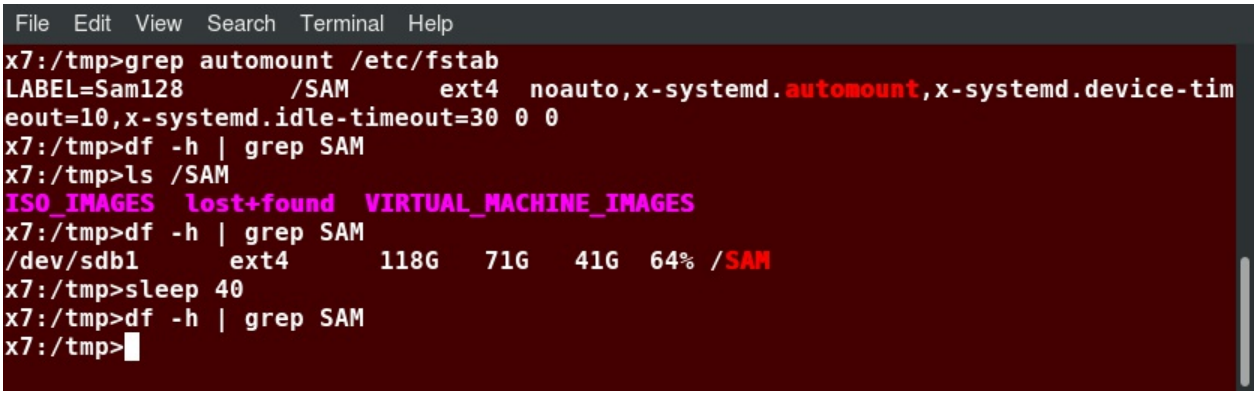
Next, will give example and explain options.

18.16 automount Example

Example provided next mounts USB pen drive that is always plugged into system, only when it is used. Options in `/etc/fstab`:

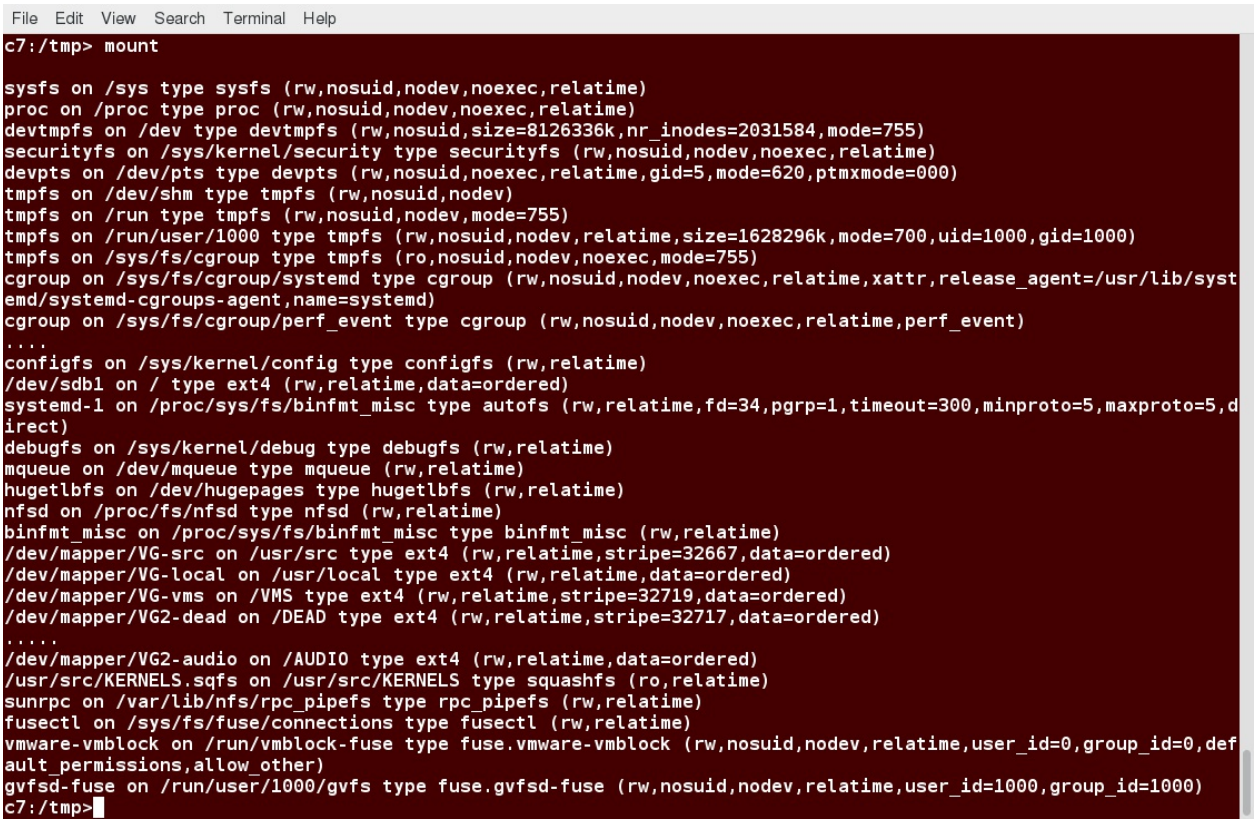
- `noauto`: do not mount at boot. Here, `auto` does not refer to **automount**
- `x-systemd.automount`: use the **systemd automount** facility
- `x-systemd.automount.device-timeout=10`: if this device is not available, say it is a network device accessible through NFS, timeout after 10 seconds instead of getting hung
- `x-systemd.automount.idle-timeout=30`: if device is not used for 30 seconds, unmount it

Note: device **may** be mounted during boot, but then should be unmounted after timeout specified. Below shows how device is only available one it is used.



18.17 Listing Currently Mounted Filesystems

List of currently mounted filesystems can be seen by typing:



##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 19 Filesystem Features: Swap, Quotas, Usage - Notes

19.2 Introduction

Linux uses tobust **swap space** implementation through w hich virtual memory system permits apparent use of more memory than is physically available. Filesystem **quotas** can be used to administer user account usage of disk space. Utilities such as **df** and **du** enable easy monitoring of filesystem usage and capabilities.

19.3 Learning Objectives:

- Explain the concepts of sw ap and quotas.
- Use the utilities that help manage quotas: **quotacheck**, **quotaon**, **quotaoff**, **edquota**, and **quota**.
- Use the utilities of **df** and **du**.

19.4 Using swap

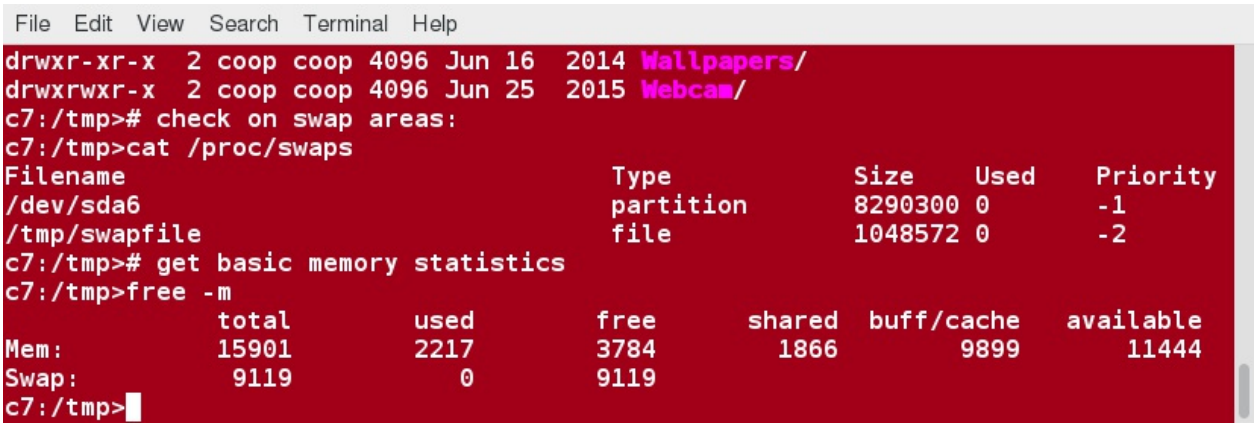
Linux employs **virtual memory** system, in w hich operating system can function as if it had more memory than it really does. This kind of memory **overcommission** functions in tw o w ays:

- Many programs do not actually use all the memory they are given permission to use. Sometimes, because child processes inherit copy of parent's memory regions utilizing **COW** (**C**opy **O**n **W**rite) technique, in w hich child only obtains unique copy (on page-by-page basis) w hen there is change
- When memory pressure becomes important, active memory regions may be sw apped out to disk, to be recalled only w hen needed again

Such sw apping usually done to one or more dedicated partitions or files. Linus permits multiple sw ap areas, so needs can be adjusted dynamically. Each area has a priority, and low er priority areas are not used until higher priority areas are filled.

In most situations, recommended sw ap size = total RAM on system. Can see w hat system currently using for sw ap areas by looking at `/proc/swaps` and getting basic memory statistics w ith **free**:

```
$ cat /proc/swaps
$ free -m
```



Only commands involving swap:

- **mkswap**: format a swap partition or file
- **swapon**: activate a swap partition or file
- **swapoff**: deactivate a swap partition or file

At any given time, most memory in use for caching file contents to prevent actually going to the disk more than necessary, or in a sub-optimal order or timing. Such pages of memory never swapped out as backing store is file themselves, so writing out a swap would be pointless. Instead, **dirty** pages (memory containing updated file contents that no longer reflect the stored data) flushed out to disk.

Also worth pointing out that in Linux, memory used by kernel itself, as opposed to application memory, *never* **swapped** out, in distinction to some other operating systems.

19.5 Filesystem Quotas

Linux can use + enforce quotas on filesystems. Disk quotas allow administrators to control maximum space particular users (or groups) are allowed. Considerable flexibility allowed, and quotas can be assigned on per filesystem basis. Protection provided again subset of users exhausting collective resources.

These utilities help manage quotas:

- **quotacheck**: generates + updates quota accounting files
- **quotaon**: enables quota accounting
- **quotaoff**: disables quota accounting
- **edquota**: used to editing user or group quotas
- **quota**: reports on usage and limits

Quota operations require existence of the files `aquota.user` and `aquota.group` in root directory of filesystem using quotas.

Quotas may be enabled or disabled on per-filesystem basis. In addition, Linux supports use of quotas based on user/group IDs.

Different filesystem types may have additional quota-related utilities, such as **xfs_quota**.

19.6 Setting up Quotas

To create filesystem quota, must first make sure to have mounted filesystem with the user and/or group quota mount options. Without these, nothing else will work. Basic steps:

- Mount filesystem with user and/or group quota options:
  - Add `usrquota` and/or `grpquota` options to filesystems entry in `/etc/fstab`
  - Remount filesystem (or mount it if new)

- Run **quotacheck** on filesystem to set up quotas
- Enable quotas on filesystem
- Set quotas with **edquota** program

First, need to put the right options in `/etc/fstab`:

```
/dev/sda5 /home ext4 defaults,usrquota 1 2
```

where we have assumed `/home` is on a dedicated partition.

Then, test with following commands:

```
$ sudo mount -o remount /home
$ sudo quotacheck -vu /home
$ sudo quotaon -vu /home
$ sudo edquota someusername
```

May also want to set up **grace periods** with **edquota**. Mount options that should be used in `/etc/fstab` file are `usrquota` for user quotas and `grpquota` for group quotas.

19.7 quotacheck

quotacheck utility creates + updates quota accounting files (`aquota.user` and `aquota.group`) for filesystem.

To update user files for all filesystems in `/etc/fstab` with user quota options:

```
$ sudo quotacheck -ua
```

To update group files for all filesystems in `/etc/fstab` with group quota options:

```
$ sudo quotacheck -ga
```

To update user file for particular filesystem:

```
$ sudo quotacheck -u [somefilesystem]
```

To update group file for particular filesystem:

```
$ sudo quotacheck -g [somefilesystem]
```

Use the `-v` to get more verbose output.

quotacheck generally only run when quotas initially turned on (or need to be updated). Program may also be run when **fsck** reports errors in filesystem when system is starting up.

19.8 Turning Quotas On and Off

quotaon used to turn filesystem quotas on. **quotasoff** used to turn them off. Used as in:

```
$ sudo quotaon  [flags] [filesystem]
$ sudo quotaoff [flags] [filesystem]
```

where flags can be:

```
-a, --all               turn quotas off for all filesystems
-f, --off               turn quotas off
-u, --user              operate on user quotas
-g, --group             operate on group quotas
-P, --project           operate on project quotas
-p, --print-state       print whether quotas are on or off
-x, --xfs-command=cmd   perform XFS quota command
-F, --format=formatname  operate on specific quota format
-v, --verbose           print more messages
-h, --help              display this help text and exit
-V, --version           display version information and exit
```

Note: **quotaon** and **quotaoff** programs really one and the same, operate accordingly to which name they are called with.

For example:

```
$ sudo quotaon -av
/dev/sda6 [/]: group quotas turned on
/dev/sda5 [/home]: user quotas turned on

$ sudo quotaoff -av
/dev/sda6 [/]: group quotas turned off
/dev/sda5 [/home]: user quotas turned off

$ sudo quotaon -avu
/dev/sda5 [/home]: user quotas turned on

$ sudo quotaoff -avu
/dev/sda5 [/home]: user quotas turned off

$ sudo quotaon -avg
/dev/sda6 [/]: group quotas turned on

$ sudo quotaoff -avg
/dev/sda6 [/]: group quotas turned off
```

Note: quota operations will fail if files `aquota.user` or `aquota.group` do not exist.

19.9 Examining Quotas

quota utility used to generate reports on quotas:

- `quota` (or `quota -u`) returns current user quota
- `quota -g` returns current group quota
- Superuser may look at quotas for any user or group by specifying user or group name

For example:

```
$ sudo quota george
Disk quotas for user george (uid 1000):
   Filesystem   blocks quota limit grace files quota limit grace
```

```
     /dev/sda5   837572   500  1000        5804     0     0

$ sudo quota gracie
Disk quotas for user gracie (uid 1001):
   Filesystem   blocks quota limit grace files quota limit grace
    /dev/sda5    83757  5000 10000        5804     0     0
```

19.10 Setting Quotas

Typing **edquota** brings up quota editor. For specified user or group, temporary file created with text representation of current disk quotas for that user or group.

Then, editor invoked for that file, and quotas may then be modified. Once you leave editor, temporary file read and binary quota files adopt changes.

Soft and hard limits -> only fields which can be edited in quota. Other fields informational only.

Examples of how to use **edquota**:

- `edquota -u [username]` edits limits for `username`
- `edquota -g [groupname]` edits limits for `groupname`
- `edquota -u -p [userproto] [username]` copies `userproto`'s user quota values to `username`
- `edquota -g -p [groupproto] [groupname]` copies `groupproto`'s group quota values to `groupname`
- `edquota -t` to set grace periods

Third and fourth commands useful for including in scripts which might be used to create new accounts and set quotas for them.

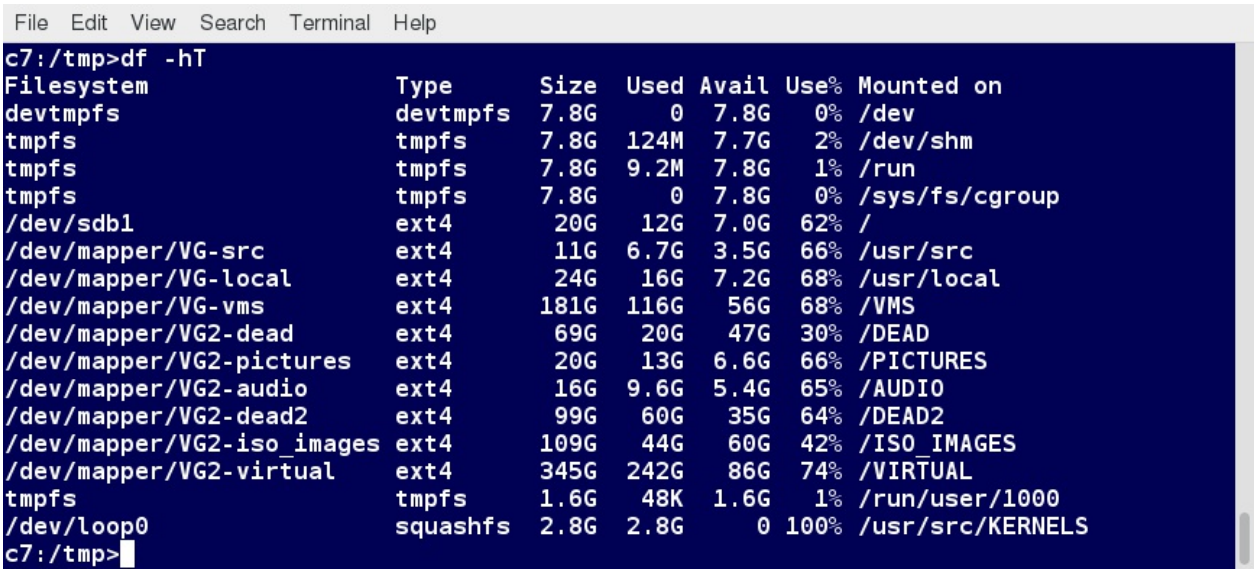
Quotas for users/groups may be set for disk blocks and/or inodes. In addition, soft/hard limits may be set, as well as grace periods. Soft limits may be exceeded for a grace period. Hard limits may never be exceeded.

Grace period set on per-filesystem basis.

```
$ sudo edquota gracie
$ sudo edquota -t
```

19.11 df: Filesystem Usage

df (**d**isk **f**ree) utility examines filesystem capacity and usage. Below, `-h` option means "human-readable" (ie. in KB, MD, GB, not bytes) and `-T` shows filesystem type. Using `-i` option would show inode information instead of bytes.



19.12 du: Disk Usage

du (**d**isk **u**sage) used to evaluate both disk capacity and usage.

To display disk usage for current directory:

```
$ du
```

To list all files, not just directories:

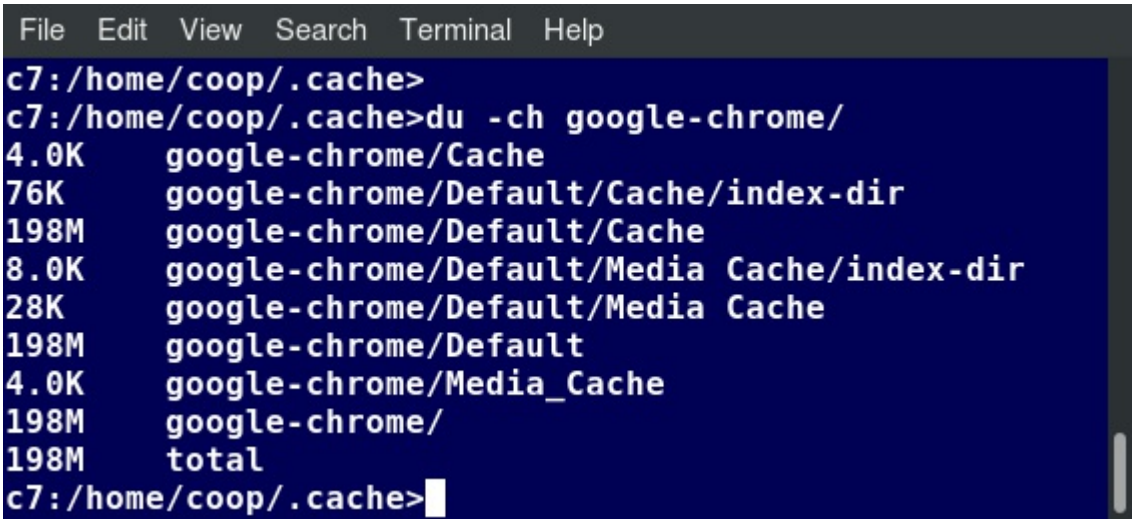
```
$ du -a
```

To list in human-readable format:

```
$ du -h
```

To display disk usage for specific directory:

```
$ du -h somedir
```



##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 20 The Ext2/Ext3/Ext4 Filesystems - Notes

20.2 Introduction

ext family of filesystems native to Linux since its earliest days, have been the most w idely used of choices available. Until very recently, **ext4** most frequent default choice of Linux distributions, due to its excellent combination of performance, integrity, stability.

20.3 Learning Objectives:

- Describe the main features of the **ext4** filesystem and how it is laid out on disk.
- Explain the concepts of **block groups**, **superblock**, **data blocks and inodes**.
- Use the **dumpe2fs** and **tune2fs** utlities.
- List the **ext4** filesystem enhancements.

20.4 Ext4 history and Basics

ext2 filesystem: Linux's original native variety, is available on all Linux systems, but rarely used today.

ext3 filesystem: first **journalling** extension of **ext2**. Had same on-disk layout as **ext2** except for presence of journal file.

ext4 filesystem: first appeared in kernel version 2.6.19, experimental designation removed in version 2.6.28. Included significant enhancements, eg. use of **extents** for large files, instead of lists of file blocks.

extent: simply a group of contiguous blocks. Can improve large file performance + reduce fragmentation by using them.

ext4 -> default filesystem on mos enterprise Linux distributions, although RHEL 7 adopted **XFS** as default.

20.5 Ext4 Filesystem Features

Close correspondence of features betw een VFS and ext2/3/4 filesystems, as each designed w ith other in mind.

Block size is selected w hen filesystem built. Can be 512, 1024, 2048, or 4096 bytes. By default, 4096 usually used for hard disk systems, unless partition very small. Unless very many small files, larger block sizes can be more efficient in terms of minimizing hard disk access.

Linux kernel memory management system requires only an integral number of blocks must fit into page of memory. Thus, cannot have 8 KB blocks on x86 platform w here pages of memory 4 KB in size.

Number of **inodes** on filesystem can also be tuned, w hich may save disk space.

Feature called **inode reservation** consists of reserving several inodes w hen directory is created, expecting them to be used in the future. Improves performance, because, w hen new files created, directory already has inodes allocated.

If pathname of symbolic link less than 60 characters, so-called *fast symbolic link* created, stored directly in inode, obviating need to read data block.

20.6 Ext4 Filesystem Enhancements

ext4 filesystem:

- Backwards compatible with **ext3** and **ext2**
- Increases maximum filesystem size to 1 EB (from 16 TB), and maximum file to 16 TB (from 2 TB). These limits arise from 48-bit addressing that is used; full 64-bit addressing may be in future, but not really needed at this point
- Increases without limit the number of subdirectories, which was limited to 32K in ext3
- Splits large files into the largest possible extents instead of using indirect block mapping. This can improve large file performance and reduce fragmentation
- Uses **multiblock allocation** which can allocate space all at once, instead of one block at a time. In addition, delayed allocation can also increase performance
- Can **pre-allocate** disk space for a file. Allocated space is usually both guaranteed and contiguous
- Uses **allocate-on-flush**, a performance technique which delays block allocation until data is written to disk
- Uses fast **fsck** which can make the speed of a filesystem check go up by an order of a magnitude or more
- Uses **checksums** for journal which improves reliability. Can also safely avoid a disk I/O wait during journalling, resulting in slight performance boost
- Uses improved timestamps measured in nanoseconds

20.7 Ext4 Layout

All fields written to disk in **little-endian** order, except the journal.

Disk blocks partitioned into **block groups**, each of which contains inode and data blocks stored adjacently, thereby lowering access time.

Layout of standard block group is simple. For block group 0, first 1024 byte unused (to allow for boot sectors, etc.). Superblock will start at first block, except for block group 0. This is followed by group descriptors and number of **GDT** (**G**roup **D**escriptor **T**ables) blocks. These followed by data block bitmap, inode bitmap, inode table, and data blocks.

Data blocks **pre-allocated** to files before actually used. Thus, when file's size increased, adjacent space already reserved and file fragmentation reduced.

Filesystem **superblock** contains bit-fields which are used to ascertain whether or not filesystem requires checking when first mounted during system initialization. Status can be:

- **clean**: filesystem cleanly unmounted previously
- **dirty**: usually means mounted
- **unknown**: not cleanly unmounted, such as when there is a system crash

In latter two cases, **fsck** will be run to check filesystem and fix any problems before it is mounted.

Other fields in superblock contain information about when filesystem was last checked, both in data + number of mounts, and automatic checking triggered when tunable limits exceeded for these quantities.

First block on filesystem: so-called **boot block**, reserved for partition boot sector.

20.8 Superblock Information

Superblock contains global information about filesystem:

- Mount count and maximum mount count (Mount count is incremented every time disk is successfully mounted, and its value is number of times this has been done since last **fsck**.) One can also force filesystem check after 180 days by default, or another time that can be changed by **tune2fs** utility
- Block size (Cannot be greater than page of memory, set by **mkfs**.)
- Blocks per group
- Free block count

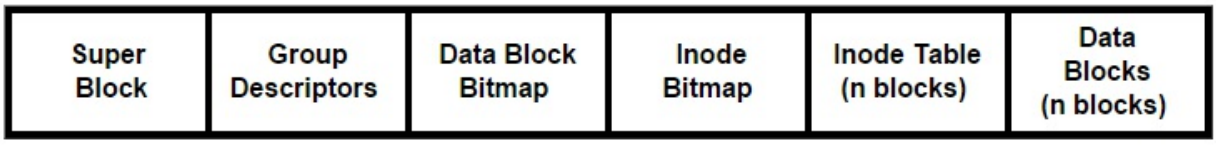
- Free inode count
- Operating System ID

As discussed earlier, superblock redundantly stored in several block groups.

20.9 Block Groups

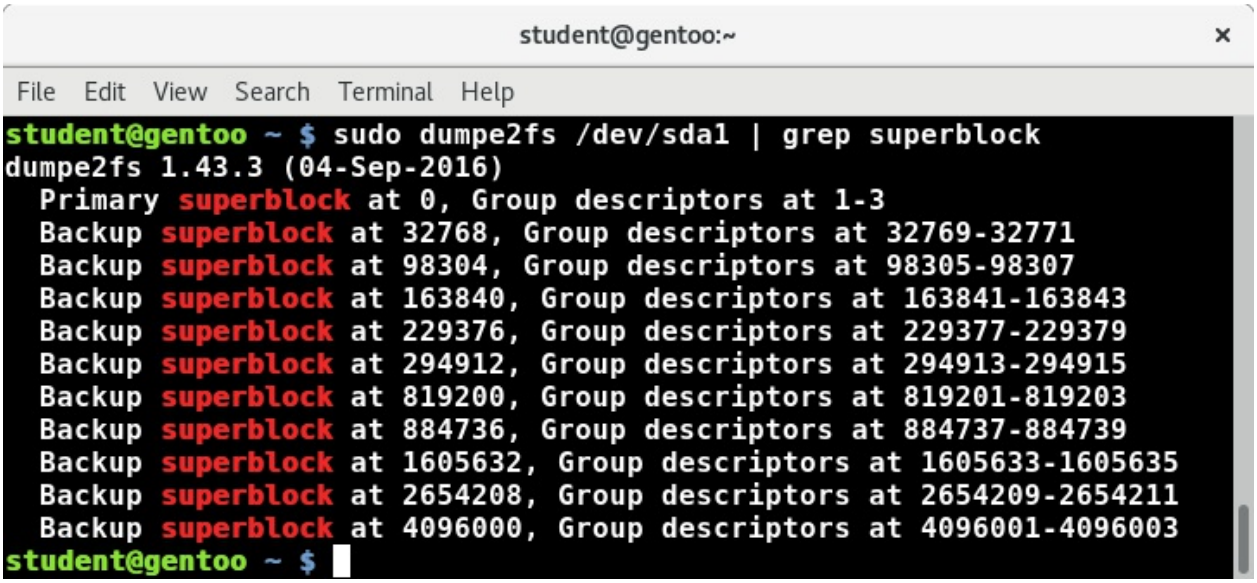
After boot block, there is a series of **block groups**, each of which is same size. Layout of each block group given below:

ext2/3/4 Filesystem Layout



First and second block groups same in every block group, comprise **Superblock** and **Group Descriptors**. Under normal circumstances, only those in first block group used by kernel; duplicate copies only referenced when filesystem being checked. If everything OK, kernel merely copies them over from the first block group. If there is problem with master copies, goes to next and so on until healthy one found and filesystem structure is rebuilt. This redundancy makes it very difficult to thoroughly fry **ext2/3/4** filesystem, as long as filesystem checks run periodically.

In early incarnations of **ext** filesystem family, each block group contained group descriptors for every block group + copy of superblock. As optimization today, not all block groups have copy of superblock + group descriptors. Too see what you have, could examine it as shown below (putting in appropriate device node) to see precise locations. Happens when filesystem created with `sparse-super` option, which is default.



Number of block groups constrained by fact that the **block bitmap**, which identifies used and free blocks within group, has to fit in single block. Thus, if block 4096 bytes in size, a block group can contain no more than 32 K blocks, or 128 MB. If we take pargest possible block group size, 10 GB partition would thus have to have at least 80 block groups.

Block allocator tries to keep each file's blocks within same block group to reduce seek times.

20.10 Data Blocks and Inodes

Data block and **inode** bitmaps are blocks whose bits contain 0 for each free blocks or inode, and 1 for each used one. Only one

of these bitmaps per block group.

Inode table contains as many consecutive blocks as necessary to cover number of inodes in block group. Each inode requires 128 bytes. Thus, 4 KB block can contain 32 inodes.

Note: there is space reserved for some operating system dependent information; different OSs could possibly mount **ext2/3/4** filesystem, much as Linux can mount many kinds of non-native filesystems.

Note: inode number number itself *not* stored on disk in this structure. Calue can be quickly calculated from block group number and offse w ithin inode table.

ext2 and **ext3** filesystems did not yet incorporate use of extents to organize larger files. Instead, `i_block[]` array of pointers to data blocks, of length `EXT_N_BLOCKS=15` described by inode. The w ay this is handled is somew hat complex:

- First 12 elements in array simply point to first 12 data blocks in file
- 13th element points to block that represents a second-order array of block numbers; 14th to a third-order array, 15th to fourth-order array

Algorithm makes addressing small files go the fastest, as it should. For instance, w ith 4 KB block size, 48 KB file can be directly addressed, a 2 MB file requires a second-order process, a 1 GB a third-order, a 4 GB file a fourth-order.

20.11 dumpe2fs

Can use the utility **dumpe2fs** to scan filesystem information (limits, capabilities and flags, other attributes). Eg.:

```
$ sudo dumpe2fs /dev/sda1
```

20.12 tune2fs

tune2fs used to change filesystem parameters.

To change maximum number of mounts betw een filesystem checks (max-mount-count):

```
$ sudo tune2fs -c 25 /dev/sda1
```

To change time internal betw een checks (interval-betw een-checks):

```
$ sudo tune2fs -i 10 /dev/sda1
```

To list contents of superblock, including current values of parameters w hich can be changed:

```
$ sudo tune2fs -l /dev/sda1
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 21 The XFS and btrfs Filesystems - Notes

21.2 Introduction

XFS and **btrfs** filesystems have risen as important challengers to dominance of **ext4** in Enterprise Linux distributions. These next generation filesystems have robust capabilities with respect to handling large sizes, spanning multiple physical + logical volumes, and performance metrics.

21.3 Learning Objectives:

- Describe the **XFS** filesystem.
- Maintain the **XFS** filesystem.
- Describe the **btrfs** filesystem.

21.4 XFS Features

XFS filesystem designed + created by Silicon Graphics Inc. (SGI), used in IRIX operating system, subsequently ported to Linux. Engineered to deal with large data sets for SGI systems + handle parallel I/O tasks very effectively.

Can handle:

- Up to 16 EB (exabytes) for total filesystem
- UP to 8 EB for an individual file.

High performance -> one of the key design elements of **XFS**, which implements methods for:

- Employing **DMA** (**D**irect **M**emory **A**ccess) I/O
- Guaranteeing an I/O rate
- Adjusting stripe size to match underlying **RAID** or **LVM** storage devices

Unline other Linux filesystems, **XFS** can also journal quota information. Leads to decreased recovery time when quota-enabled filesystem uncleanly mounted. Furthermore, journal can be on external device.

As with other UNIX and Linux filesystems, **XFS** supports extended attributes.

21.5 XFS Details

Maintaining **XFS** filesystem made easier by the fact that most of the maintenance tasks can done on-line (ie. while filesystem fully mounted). Include:

- Defragmenting
- Resizing (enlarge only)
- Dumping/Restoring

Backup and restore can be done with native **XFS** utilities:

- **xfsdump**
- **xfsrestore**

which can be conveniently paused and then resumed at later time. Because these utility programs also multi-threaded, **XFS** dumps and restores can be accomplished very quickly.

Traditional quota commands can be used to manipulate per-filesystem quotas on XFS volume. However, if you use **xfs_quota** command, can use the per-directory quotas that XFS supports.

XFS filesystem does not directly support hardware or software **snapshots**. However, **xfs_freeze** utility can be used to quiesce filesystem for snapshots.

21.6 The btrfs Filesystem

Both Linux developers and Linux users with high performance and high capacity or specialized needs -> following the development and gradual deployment of **btrfs** filesystem, created by Chris Mason. Name stands for **B-tree file system**. Full documentation can be found on btrfs wiki page.

btrfs intended to address lack of pooling, snapshots, checksums, integral multi-device spanning in other Linus filesystems such as **ext4**. Such features can be crucial for Linus to scale into larger enterprise storage configurations.

While **btrfs** has been in mainline kernel since 2.6.29, has generally been viewed as experimental. However, some vendors using it in new products, default root filesystem on SLES and OpenSUSE.

One of the main features -> ability to take frequent snapshots of entire filesystems, or sub-volumes of entire file-systems in virtually no time. Because **btrfs** makes extense use of COW techniques (Copy on Write), such snapshot does not involve any more initial space for data blocks or any I/O activity except for some metadata updating.

One can easily revert to state described by earlier snapshots, even induce kernel to reboot off earlier root filesystem snapshot.

btrfs maintains it own internal framework for adding or removing new partitions and/or physical media to existing filesystems, such as LVM (Logical Volume Management) does.

Before **btrfs** suitable for day-to-day use in critical filesystems, some tasks require finishing. For review of development history of **btrfs** and expected evolution, see https://lwn.net/Articles/575841/.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 22 Encrypting Disks - Notes

22.2 Introduction

Filesystems may be **encrypted** to protect them from both prying eyes and attempts to corrupt data they contain. Encryption can be chosen at installation, or incorporated later. Linux distributions most often use **LUKS** method, and perform encrcyption-related tasks using **cryptsetup**.

22.3 Learning Objectives:

- Provide sound reasons for using encryption and know when it is called for.
- Understand how **LUKS** operates through the use of **cryptsetup**.
- Set up and use encrypted filesystems and partitions.
- Configure the system to mount encrypted partitions at boot.

22.4 Why Use Encryption?

Encryption should be used whenever sensitive data is being stored and transmitted. Configuring and using block device level encryption provides one of the strongest protections against hard caused by loss or compromise of data contained in hard drives + other media.

Modern Linux distributions offer choice of encrypting all or some of disk partitions during installation. Also straightforward to create + format encrypted partitions at later time, but cannot encrypt already existing partition in place without data copying operation.

22.5 LUKS

Modern Linux distributions provide block device level encryption mainly through use of **LUKS** (**L**inux **U**nified **K**ey **S**etup). Using block device encryption highly recommended for portable systems such as laptops, tablets, smart phones.

LUKS installed on top of **cryptsetup**, powerful utility that can also use other methods such as **plain dm-crypt** volumes, **loop-AES**, **TrueCrypt**-compatible format. Won't discuss these alternatives, as LUKS (which was already designed for Linux, but also been exported to other operating systems) standard method most often used in Linux.

dm-crypt kernel module uses **device mapper** kernel infrastructure that is also heavily used by LVM, which will be discussed later.

Because LUKS stores all necessary information in partition header itself, rather easy to migrate partitions to other disks or systems.

LUKS can also be used to transparently encrypt swap partitions.

22.6 cryptsetup

Basically, everything done with Swiss army knife program **cryptsetup**. Once encrypted volumes set up, can be mounted/unmounted with normal disk utilities.

General form of command:

```
cryptsetup [OPTION...] <action> <action-specific>
```

and full listing of possibilities generated by:

```
$ cryptsetup --help
```

22.7 Using an Encrypted Partition

If partition `/dev/sdc12` already exists, following commands will set up encryption, make it available to LUKS, format it, mount it, use it, and unmount it.

First, need to give partition to LUKS:

```
$ sudo cryptsetup luksFormat /dev/sdc12
```

Will be prompted for passphrase that will need to open use of encrypted volume later. Note: only have to do this step once, when setting up encryption. Kernel may not support default encryption method used by **cryptsetup**. In such case, can examine `/proc/crypto` to see methods supported by system, and can supply method:

```
$ sudo cryptsetup luksFormat --cipher aes /dev/sdc12
```

Can make volume available at any time with:

```
$ sudo cryptsetup --verbose luksOpen /dev/sdc12 SECRET
```

where you will be prompted to supply passphrase. Can format partition:

```
$ sudo mkfs.ext4 /dev/mapper/SECRET
```

mount it:

```
$ sudo mount /dev/mapper/SECRET /mnt
```

and then use to heart's content, just as if it were an unencrypted partition. When done, unmount with:

```
$ sudo umount /mnt
```

and then remove mapper association for now. Partition will always be available for later use:

```
$ sudo cryptsetup --verbose luksClose SECRET
```

22.8 Mounting at Boot

To mount encrypted partition at boot, two conditions have to be satisfied:

1. Make appropriate entry in `/etc/fstab`. Nothing special about this, does not refer to encryption in any way. Can be as simple as:

```
/dev/mapper/SECRET /mnt ext4 defaults 0 0
```

2. Add entry to `/etc/crypttab`. Can be as simple as:

```
SECRET /dev/sdc12
```

Can do more in this file, such as specifying password if you don't want to be prompted at boot (which seems counterproductive security-wise). See **man crypttab** to find out what can be done with this file.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 23 Title - Notes

23.3 Learning Objectives:

- Explain the concepts behind **LVM**.
- Create logical volumes.
- Display logical volumes.
- Resize logical volumes.
- Use **LVM** snapshots.

23.4 LVM

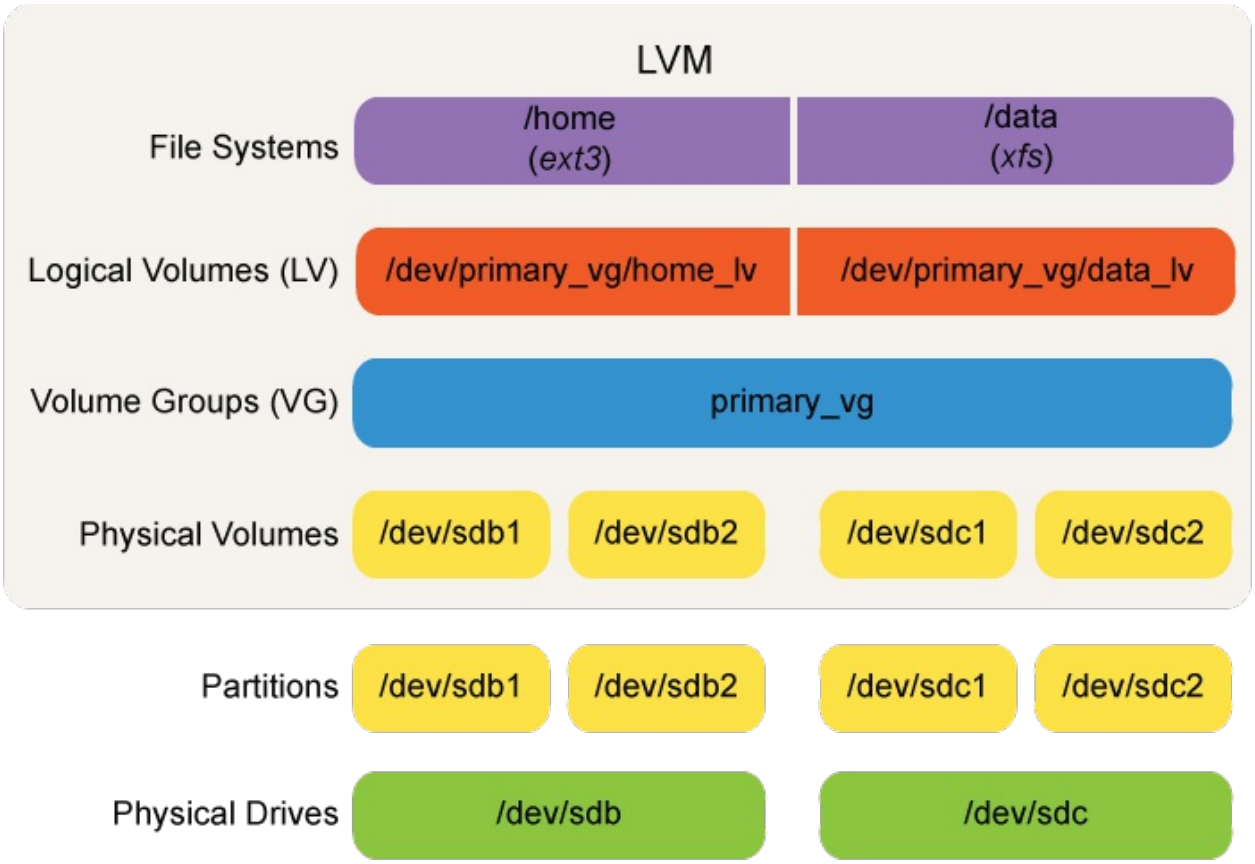
LVM (**L**ogical **V**olume **M**anagement) breaks up one virtual partition into multiple chunks, each of which can be on different partitions and/or disks.

Many advantages to using LVM. Particularly, becomes really easy to change size of logical partitions and filesystems, to add more storage space, rearrange things, etc.

One or more **physical volumes** (disk partitions) grouped together into **volume group**. Then, volume group subdivided into **logical volumes**, which mimic to system nominal physical disk partitions, and can be formatted to contain mountable filesystems.

Variety of command line utilities tasked to create, delete, resize etc. physical and logical volumes. For graphical tool, some Linux distributions use **system-config-lvm**. However, RHEL 7 does not support this tool, and there is currently no graphical tool that works reliably with most recent variations in filesystem types etc. Command line utilities not hard to use, and more flexible anyway.

LVM does impact performance. Definite additional cost that comes from overhead of LVM layer. However, even on non-RAID systems, if using **striping** (splitting of data to more than one disk), can achieve some parallelization improvements.



LVM Components

23.5 LVM and RAID

Like RAID (which will be discussed in next chapter), use of logical volumes = mechanism for creating filesystems which can span more than one physical disk.

Logical volumes created by putting all devices into large pool of disk space (the volume group), then allocating space from pool to create logical volume.

Logical volumes have features similar to RAID devices. Can actually be built on top of RAID device. Would give logical volume redundancy of RAID device with scalability of LVM.

LVM has better scalability than RAID: logical volumes can easily be resized, ie. enlarged or shrunk, as needs require. If more space needed, additional devices can be added to logical volume at any time.

23.6 Volumes and Volume Groups

Partitions converted into physical volumes, and multiple physical volumes grouped into volume groups. Can be more than one volume group on system.

Space in volume group divided into **extents**. 4 MB in size by default,but size can be easily changed when being allocated.

Number of command line utilities used to create/manipulate volume groups, whose name always start with **vg** including:

- **vgcreate**: Create volume groups
- **vgextend**: Adds to volume groups
- **vgreduce**: Shrinks volume groups

Utilities that manipulate what physical partitions enter or leave volume groups start with **pv** and include:

- **pvcreate**: Converts a partition to a physical volume
- **pvdisplay**: Shows the physical volumes being used
- **pvmove**: Moves the data from one physical colume within volume group to others. Might be required if a disk or partition is being removed for some reason. Would then be followed by:
- **pvremove**: Remove a partition from a physical volume

Typing **man lvm** will give full list of LVM utilities.

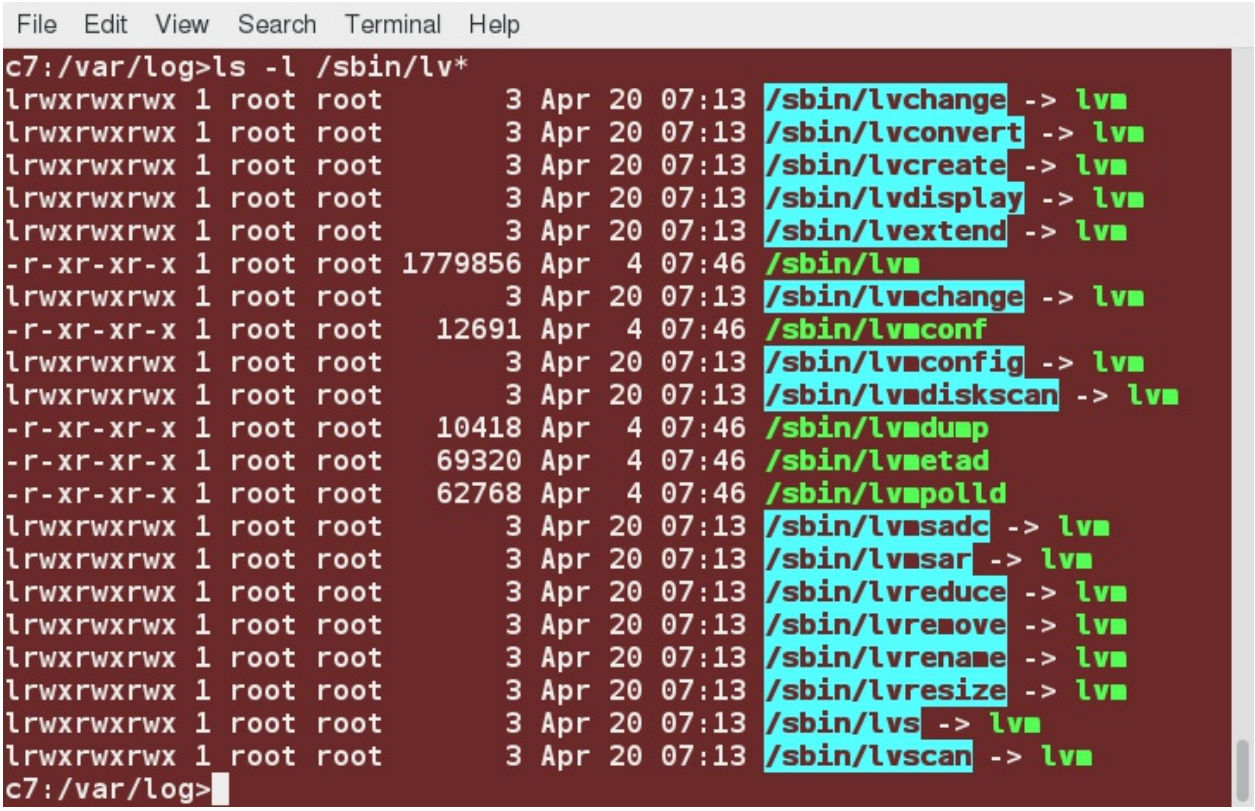
23.7 Logical Volume Utilities

Number of utilities that manipulate logical volumes. Unsurprisingly, all begin with **lv**. Will discuss most commonly used ones, but short list can be obtained using following command:

```
$ ls -lF /sbin/lv*
```

Note:

- These utilities are in `/sbin`, not `/usr/sbin`, as they may be needed either for boot or repair and recovery
- Most of them symbolically linked to **lvm**, Swiss army knife program that does all the work, but figures out what is being asked to do based on name it is invoked with. Also true for most of **pv*** and **vg*** utilities, as can be verified easily



23.8 Creating Logical Volumes

lvcreate allocates logical volumes from within volume groups. Size can be specified either in bytes or number of extents (remember, 4 MB by default). Name can be anything desired.

lvdisplay reports on available logical volumes.

Filesystems placed in logical volumes and formatted with **mkfs** as usual.

Starting with possibly creating new volume group, steps involved in setting up and using a new logical volume are:

1. Create partitions on disk drives (type `8e` in **fdisk**)
2. Create physical volumes from partitions
3. Create volume group
4. Allocate logical volumes from volume group
5. Format logical volumes
6. Mount logical volumes (also update `/etc/fstab` as needed)

For example, assuming already created partitions `/dev/sdb1` and `/dev/sdc1`, and given them type `8e`:

```
$ sudo pvcreate /dev/sdb1
$ sudo pvcreate /dev/sdc1
$ sudo vgcreate -s 16M vg /dev/sdb1
$ sudo vgextend vg /dev/sdc1
$ sudo lvcreate -L 50G -n mylvm vg
$ sudo mkfs -t ext4 /dev/vg/mylvm
$ mkdir /mylvm
$ sudo mount /dev/vg/mylvm /mylvm
```

Be sure to add line the line

```
/dev/vg/mylvm /mylvm ext defaults 1 2
```

to `/etc/fstab` to make this persistent mount.

23.9 Displaying Logical Volumes

In order to display information about LVM, following command line programs available:

- **pvdisplay** shows physical volumes. If you leave off physical volume name, lists all physical volumes:

```
$ pvdisplay
$ pvdisplay /dev/sda5
```

- **vgdisplay** shows volume groups. If you leave off volume group name, lists all volume groups:

```
$ vgdisplay
$ vgdisplay /dev/vg0
```

- **lvdisplay** shows logical volumes. If you leave off logical volume name, lists all logical volumes:

```
$ lvdisplay
$ lvdisplay /dev/vg0/lvm1
```

Without arguments, utilities will display all physical volumes, volume groups, or logical volumes.

23.10 Resizing Logical Volumes

One great advantage of using LVM: easy and quick to change size of logical volume, especially when compared with trying to do

so with physical partition that already contains filesystem. When doing this, extents can be added/subtracted from logical volume, and can come from anywhere in volume group. Need not be from physically contiguous sections of disk.

If volume contains filesystem, expanding/shrinking an entirely different operation than changing size of volume. WHen expanding logical volume with filesystem, must first expand volume, then expand filesystem. When shrinking logical volume with filesystem, must first shrink filesystem, then shrink volume.

Best done with **lvresize**:

```
$ sudo lvresize -r -L 20 GB /dev/VG/mylvm
```

where the `-r` option causes resizing of filesystem at same time as volume size change.

Filesystem cannot be mounted when being shrunk. However, some filesystems permit expansion while mounted.

Utilities which change filesystem size -> obviously filesystem dependent. Besides **lvresize**, can also use **lvextend**, **lvreduce** with **resize2fs**.

23.11 Examples of Resizing

To grow logical volume with **ext4** filesystem:

```
$ sudo lvextend -L +500M /dev/vg/mylvm
$ sudo resize2fs /dev/vg/mylvm
```

where the plus sign indicates adding space.

To shrink filesystem:

```
$ sudo umount /mylvm
$ sudo fsck -f /dev/vg/mylvm
$ sudo resize2fs /dev/vg/mylvm 200M
$ sudo lvreduce -L 200M /dev/vg/mylvm
$ sudo mount /dev/vg/mylvm
```

TUrns out that if you have recent version of **lvm** utilities, can do these operations in one step isntead of two, using `-r` option:

```
$ sudo lvextend -r -L +100M /dev/vg/mylvm
$ sudo lvreduce -r -L -100M /dev/vg/mylvm
```

where quantities use plus/minus signs to indicate change required. Uses underlying **fsadm** utility, which can resize any type of filesystem for which you have support. Would be good idea to read **man** page for **fsadm**.

Can also reduce volume group:

```
$ sudo pvmove /dev/ads1
$ sudo vgreduce vg /dev/sdc1
```

23.12 LVM Snapshots

LVM snapshots create exact copy of existing logical volume. Useful for backups, application testing, deploying VMs (Virtual Machines). Original state of snapshot kept as block map.

Snapshots only use space for storing deltas:

- When original logical volume changes, original data blocks copied to snapshot
- If data added to snapshot, stored only there

To create snapshot of existing logical volume:

```
$ sudo lvcreate -l 128 -s -n mysnap dev/vg/mylvm
```

To then make mount point and mount snapshot:

```
$ mkdir /mysnap
$ mount -o ro /dev/vg/mysnap /mysnap
```

To use snapshot and to remove snapshot:

```
$ sudo umount /mysnap
$ sudo lvremove /dev/vg/mysnap
```

Always be sure to remove snapshot when through with it. If you do not remove snapshot and fills up because of changes, automatically disabled. Snapshot with size of original will never overflow.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 24 RAID - Notes

24.2 Introduction

Use of RAID spreads I/O activity over multiple physical disks, rather than just one. Purpose to enhance data integrity and recoverability in case of failure, as well as to boost performance when used with modern storage devices. Number of different RAID levels which vary in their relative strengths in safety, performance, complexity, cost.

24.3 Learning Objectives:

- Explain the concept of RAID.
- Summarize RAID levels.
- Configure a RAID device using the essential steps provided.
- Monitor RAID devices in multiple ways.
- Use **hot spares**.

24.4 RAID

RAID (**R**edundant **A**rray of **I**ndependent **D**isks) spreads I/O over multiple disks. Can really increase performance in modern disk controller interfaces, such as SCSI, which can perform the work in parallel efficiently.

RAID can be implemented either in software (it is a mature part of Linux kernel) or in hardware. If your hardware RAID is known to be of good quality, should be more efficient than using software RAID. With hardware implementation, operating system is actually not directly aware of using RAID -> it is transparent. Eg. three 512GB hard drives (two for data, one for parity) configured with RAID-5 will just look like a single 1TB disk.

However, one disadvantage of using hardware RAID: if disk controller fails, must be replaced by compatible controller, which may not be easy to obtain. When using software RAID, same disks can be attached to + work with any disk controller. Such considerations more likely to be relevant for low/mid-range hardware.

Three essential features of RAID:

- **mirroring**: writing the same data to more than one disk
- **striping**: splitting of data to more than one disk
- **parity**: extra data is stored to allow problem detection and repair, yielding fault tolerance

Thus, use of RAID can improve both performance + reliability.

One of main purposes of a RAID device: to create filesystem which spans more than one disk. Allows us to create filesystems which are larger than any one drive. RAID devices typically created by combining partitions from several disks together.

Another advantage of RAID devices: ability to provide better performance, redundancy, or both. Striping provides better performance by spreading information over multiple devices so simultaneous writes possible. Mirroring writes same information to multiple drives, giving better redundancy.

mdadm: used to create/manage RAID devices.

Once created, the array name, `/dev/mdX`, can be used just like any other device, such as `/dev/sda1`.

24.5 RAID Levels

Number of RAID specifications of increasing complexity and use. Most commonly used levels: 0, 1, 5.

- **RAID 0** uses only striping. Data spread across multiple disks. However, in spite of name, no redundancy, no stability or recovery capabilities. In fact, if any disk fails, data will be lost. But performance can be improved significantly because of parallelization of I/O tasks.
- **RAID 1** uses only mirroring. Each disk has duplicate. Good for recovery. At least two disks required.
- **RAID 5** uses rotating parity stripe. A single drive failure will cause no loss of data, only performance drop. Must be at least 3 disks.
- **RAID 6** has striped disks with dual parity. Can handle loss of two disks, requires at least 4 disks. Because RAID 5 can impose significant stress on disks, which can lead to failures during recovery procedures, RAID 6 has become more important.
- **RAID 10**: mirrored and striped data set. At least 4 drives needed.

As general rule, adding more disks improves performance.

24.6 Software RAID Configuration

Essential steps on configuring a software RAID device:

- Create partitions on each disk (type `fd` in **fdisk**)
- Create RAID devices with **mdamd**
- Format RAID device
- Add device to `/etc/fstab`
- Mount RAID device
- Capture RAID details to ensure persistence

The command:

```
$ sudo mdamd -S
```

can be used to stop RAID.

For example, create two partitions of type `fd` on disks `sdb` and `sdc` (say `/dev/sdbX` and `/dev/sdcX`) by running **fdisk** on each:

```
$ sudo fdisk /dev/sdb
$ sudo fdisk /dev/sdc
```

Then, set up the array, format it, add to configuration, and mount it:

```
$ sudo mdadm --create /dev/md0 --level=1 --raid-disks=2 /dev/sdbX /dev/sdcX
$ sudo mkfs.ext4 /dev/md0
$ sudo bash -c "mdadm --detail --scan >> /etc/mdadm.conf"
$ sudo mkdir /myraid
$ sudo mount /dev/md0 /myraid
```

Be sure to add a line in `/etc/fstab` for the mount point:

```
/dev/md0 /myraid ext4 defaults 0 2
```

Can examine `/procmdstat` to see RAID status:

```
$ cat /proc/mdstat
    Personalities : [raid1]
    md0 : active raid1 sdb8[1] sdc7[0]
    ---------- 521984 blocks [2/2]
    unused devices: <none>
```

Can use:

```
$ sudo mdadm -S /dev/md0
```

to stop the RAID device.

24.7 Monitoring RAIDs

Can monitor RAID devices multiple ways:

```
$ sudo mdadm --detail /dev/md0
$ sudo /proc/mdstat
```

One can also use **mdmonitor**, which requires configuring `/etc/mdadm.conf`. Command:

```
$ sudo mdadm --detail /dev/mdX
```

will show current status of RAID device `/dev/mdX`. Another way to do this -> to examine `/proc` filesystem:

```
$ cat /proc/mdstat
```

Will show status of all RAID devices on system.

Can also use **mdmonitor** service by editing `/etc/mdadm.conf` and adding a line like:

```
MAILADDR email@somewhere.com
```

so that it notifies ou with email sent to `email@somewhere.com` when a problem occurs with a RAID device, eg. when any of the arrays fail to start or fall into degraded state. Can turn it on with:

```
$ sudo systemcl start mdmonitor
```

and make sure it starts at boot with:

```
$ sudo systemctl enable mdmonitor
```

(Note: on Ubuntu systems, the service is called **mdadm** rather than **mdmonitor**)

24.8 RAID Hot Spares

Redundancy: one of the important things that RAID provides. To help ensure any reduction in that redundancy is fixed as quickly as possible, a **hot spare** can be used.

To create hot space when creating RAID array:

```
$ sudo mdadm --create /dev/md0 -l 5 -n3 -x 1 /dev/sda8 /dev/sda9 /dev/sda10 /dev/sda11
```

`-x 1` switch tells **mdadm** to use one spare device. Note: hot spare can also be added at later time.

The command:

```
$ sudo mdadk --fail /dev/md0 /dev/sdb2
```

will test redundancy and hot spare of array.

To restore the tested drive, or new drive in legitimate failure situation, first remove "faulty" member, then add "new" member:

```
$ sudo mdadm --remove /dev/md0 /dev/sdb2
$ sudo mdadm --add /dev/md0 /dev/sde2
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 25 Kernel Services and Configuration - Notes

25.3 Learning Objectives:

- Grasp the main responsibilities the kernel must fulfill and how it achieves them.
- Explain what parameters can be set on the kernel command line and how to make them effective either for just one system boot, or persistently.
- Know where to find detailed documentation on these parameters.
- Use **sysctl** to set kernel parameters either after the system starts, or persistently across system reboots.

25.4 Kernel and Operating System

Narrowly defined, Linux is *only* the **kernel** of the **operating system**, which includes many other components, eg. libraries and applications that interact with the kernel.

Kernel: essential central component that connects hardware to software, and manages system resources eg. memory, CPU time allocation among competing applications/services. Handles all connected devices using **device drivers**, makes devices available for operating system use.

System running ***only*** a kernel has rather limited functionality. Will be found only in dedicated and focused **embedded devices**.

25.5 Main Kernel Tasks

Main responsibilities of kernel:

- System initialization and boot up
- Process scheduling
- Memory Management
- Controlling access to hardware
- I/O (Input/Output) between applications and storage devices
- Implementation of local and network filesystems
- Security control, both locally (such as filesystem permissions) and over the network
- Networking control

25.6 Kernel Command Line

Various parameters passed to system at boot on **kernel command line**. Normally, these on `kernel` (or `linux16`) line in GRUB configuration file, but can be modified at boot.

Sample kernel command line will depend on distribution, might look like:

```
linux  /boot/vmlinuz-4.19.0 root=UUID=7ef4747-afae-48e3-90b4-9be8be8d0258 ro quiet \
    crashkernel=384M-:128M
```

or, with more minimal set of options:

```
linux16 /boot/vmlinuz-4.19.4 root=LABEL=RHEL7 LANG=en_US.UTF-8 ro rhgb quiet
```

and would be found in `/boot/grub2/grub.cfg`.

Everything after `vmlinuz` file specified is an option. Any options not understood by kernel will be passed to **init** (`pid = 1`), the first user process to be run on system.

Should *not* edit this file directly, but modify the relevant files under `/etc`.

In examples provided above, long lines broken up for display, but they are each all one long line. To see what command line a system was booted with, type:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-4.9.0-8-amd64 root=UUID=7ef4e747-afae-48e3-90b4-9be8be8d0258 ro quiet \
    crashkernel=384M-:128M
```

25.7 Kernel Boot Parameters

Rather surprisingly long list of available kernel parameters. Detailed documentation can be found:

- In kernel source in file `Documentation/kernel-parameters.txt`
- Online, using kernel's command line parameters documentation
- On system in kernel documentation package provided by most distributions with name like **kernel-doc** or **linux-doc**
- By typing **man bootparam**

Parameters can be specified simply as a value given as an argument, or in the form `param=value`, where given value can be a string, integer, array of integers, etc., as explained in documentation file.

```
vmlinuz root=/dev/sda6 ..... noapic .... crashkernel=256M
```

Kernel options placed at end of kernel line, separated by spaces. Example of kernel boot parameter (all in one line):

```
linux16 /boot/vmlinuz-3.19.1.0 root=UUID=178d0092-4154-4688-af24-cda272265e08 ro \
        vconsole.keymap=us crashkernel=auto vconsole.font=latarcyrheb-sun16 rhgb quiet LANG=en_US.UTF-8
```

Below, can see explanation of some of the boot parameters, some of which were displayed previously:

- `root` : root filesystem
- `ro` : mounts root device read-only on boot
- `vconsole.keymap` : which keyboard to use on the console
- `crashkernel` : how much memory to set aside for **kernel crashdumps**
- `vconsole.font` : which font to use on the console
- `rhgb` : for graphical boot
- `quiet` : disables most log messages
- `LANG` : system language

By convention, should be no intentionally hidden or secret parameters. Should all be explained in documentation, and patches to kernel source with new parameters should always include patches to documentation file.

25.8 sysctl

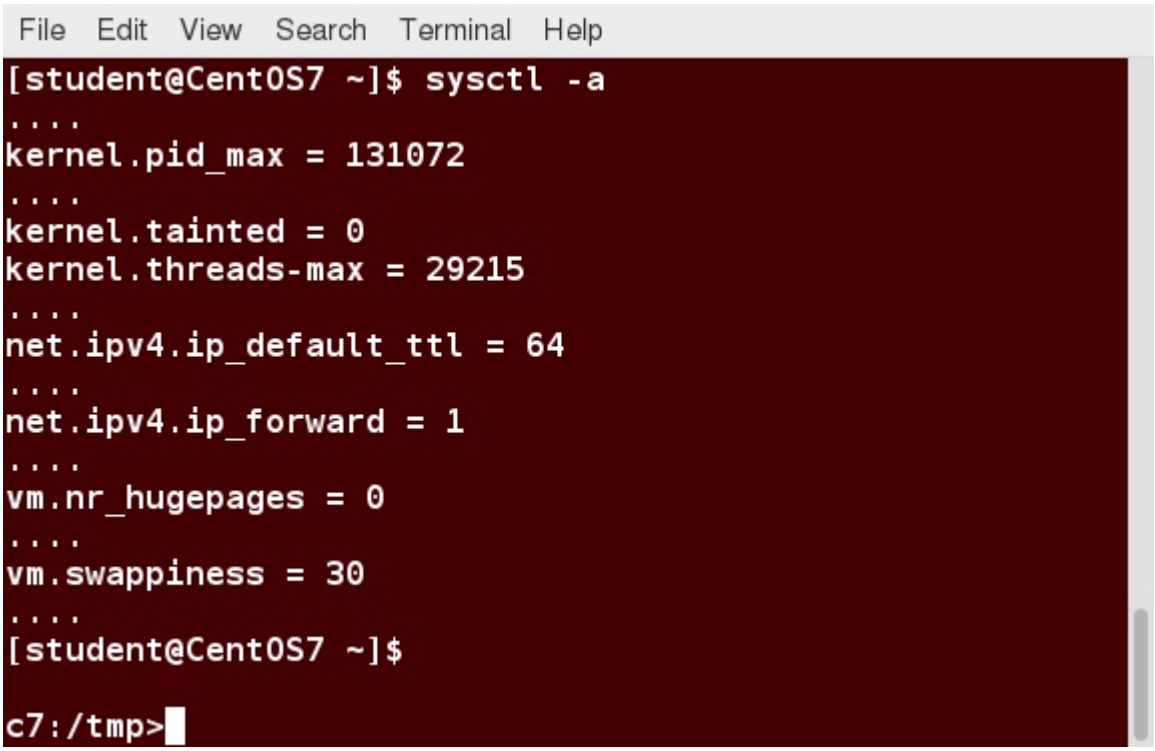
sysctl interface used to read/tune kernel parameters at runtime. Below shows how current values can be displayed by doing:

```
$ sysctl -a
```

Each value corresponds to particular pseudofile residing under `/proc/sys`, with directory slashes being replaced by dots. Eg. following two statements equivalent:

```
$ sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
$ sudo sysctl net.ipv4.ip_forward=1
```

where second form used to set value with **sysctl** command line interface. Do not leave spaces around `=` sign in this command. Note: in first form, cannot just use simple **sudo** with **echo**; command must be done in complicated way shown, or executed as root.



Browsing through pseudofiles under `/proc/sys` will render same information as `sysctl -a`. Can get full details on how to use **sysctl** by doing **man 8 sysctl**. To get information about using `sysctl()` function from programs to do the same operations, do **man 2 sysctl**.

If settings placed in `/etc/sysctl.conf` (see **man sysctl.conf** for details), settings can be fixed at boot time.

Note: typing:

```
$ sudo sysctl -p
```

effectuates immediate digestion of file, settings all parameters as found. Also part of boot process.

On some recent distributions based on **systemd** (such as RHEL 7), actual settings file is now `/usr/lib/sysctl.d/00-system.conf`, but original file still supported, as it is self-documented in that file.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 26 Kernel Modules - Notes

26.2 Introduction

Linux kernel makes extensive use of **modules**, which contains important software that can be loaded/unloaded as needed after system starts. Many modules incorporate **device drivers** to control hardware either inside system, or attached peripherally. Other modules can control network protocols, support different filesystem types, and many other purposes. Parameters can be specified when loading modules to control their behavior. End result: great flexibility and agility in responding to changing conditions and needs.

26.3 Learning Objectives:

- List the advantages of utilizing kernel modules.
- Use **insmod**, **rmmod**, and **modprobe** to load and unload kernel modules.
- Use **modinfo** to find out information about kernel modules.

26.4 Advantages of Kernel Modules

Many facilities in Linux kernel designed to be built-in to kernel when kernel initially loaded, or to be added (or removed) later as **modules** as necessary. All but the most central kernel components integrated in such fashion.

Such modules may or may not be device drivers. Eg. may implements certain network protocol or filesystem, rather than drive hardware/software device. Even in cases where functionality will virtually always be needed, incorporation of ability to load/unload as module facilitates development, as kernel reboots not required to test changes.

Even with widespread usage of kernel modules, Linux retains **monolithic** kernel architecture, rather than **microkernel** one. This is because once a module is loaded, becomes fully functional part of kernel, with few restrictions. Communicates with all kernel sub-systems primarily through shared resources, such as memory and locks, rather than through message passing as might a microkernel.

Linux hardly the only operating system to use modules. Solaris does it as well, as does AIX, which terms them **kernel extensions**. However, Linux uses them in particularly robust fashion.

26.5 Module Utilities

Number of utility programs used with kernel modules:

- **lsmod**: List loaded modules
- **insmod**: Directly load modules
- **rmmod**: Directly remove modules
- **modprobe**: Load or unload modules, using a pre-build module database with dependency and location information
- **depmod**: Rebuild the module dependency database; needed by **modprobe** and **modinfo**
- **modinfo**: Display information about a module

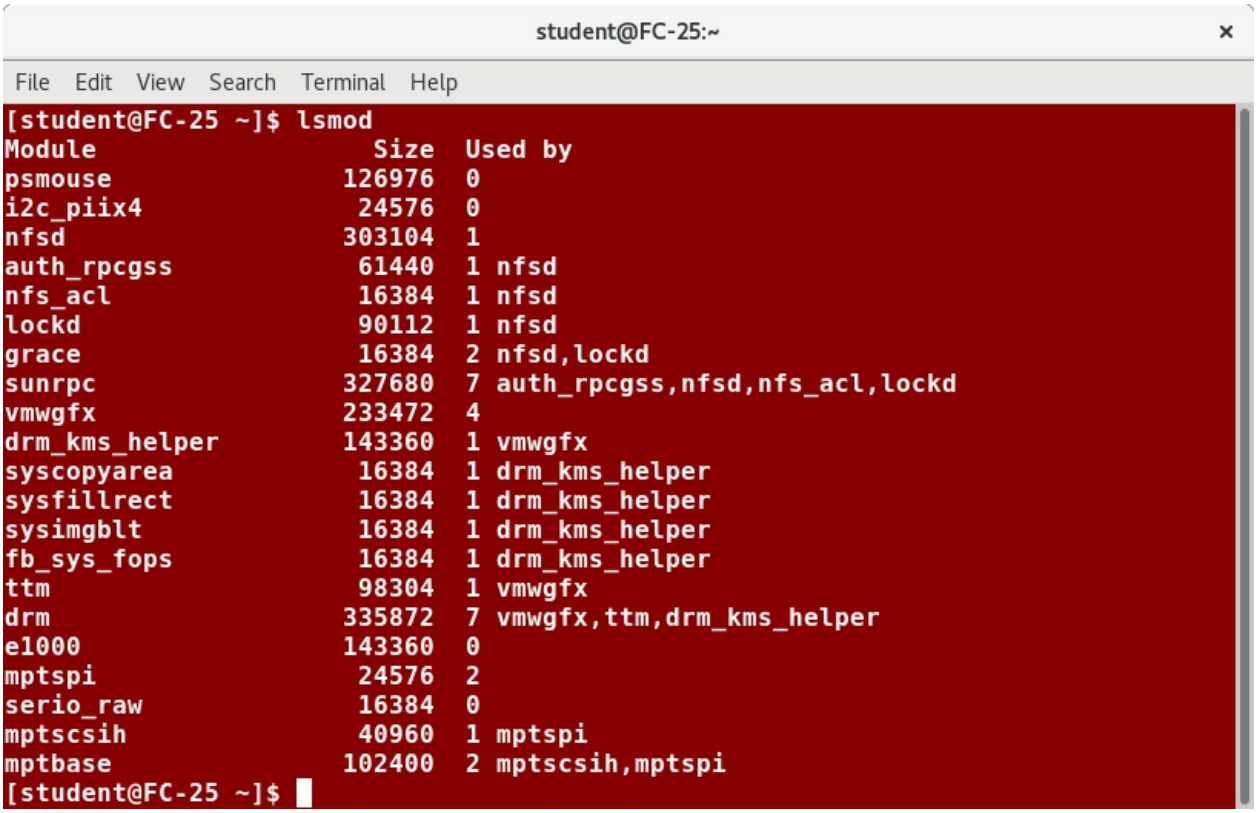
26.6 Module Loading and Unloading

While module is loaded, can always see its status with **lsmod**, as shown below.

Module removal can always be done directly with:

```
$ sudo /sbin/rmmod module_name
```

Note: not necessary to supply either full path name or `.ko` extension when removing module.



Module loading/unloading must be done as root user. If full path name known, can always load module directly with:

```
$ sudo /sbin/insmod <pathto>/module_name.ko
```

Normal filesystem location for kernel modulesL under directory tree at `/lib/modules/<kernel-version>`. Kernel module always has file extension of `.ko`, as in `.e1000e.ko`, `ext4.ko`, or `usbserial.ko`.

Kernel modules -> kernel version specific, must match running kernel or they cannot be loaded. Must be compiled either when kernel itself compiled, or later, on system which retails enough of kernel source and compilation configuration to do this properly.

26.7 modprobe

In most cirucmstances, modules not loaded/unloaded with **insmod** and **rmmod**. Rather, one uses **modprobe**:

```
$ sudo /sbin/modprobe module_name
$ sudo /sbin/modprobe -r module_name
```

with second form being used for removal. For **modprobe** to work, modules must be installed in proper location, generally under `/lib/modules/$(uname -r)`, where `$(uname -r)` gives current kernel version, such as `4.14.2`.

Can use **depmode** to generate/update the file `/lib/modules/$(uname -r)/modules.dep`.

26.8 Some Considerations with Modules

Some important things to keep in mind when loading/unloading modules:

- Impossible to unload module being used by one or more other **modules**, can ascertain this from **lsmod** listing
- Impossible to unload module being used by one or more **processes**, can also be seen from **lsmod** listing. However, some modules do not keep track of this reference count, eg. network device driver modules, as it would make it too difficult to temporarily replace module without shutting down and restarting much of the whole network stack
- When module loaded with **modprobe**, system will automatically load any other modules that need to be loaded first (dependencies)
- When module unloaded with **modprobe -r**, system will automatically unload any other modules being used by the module, if they are not being simultaneously used by any other loaded modules

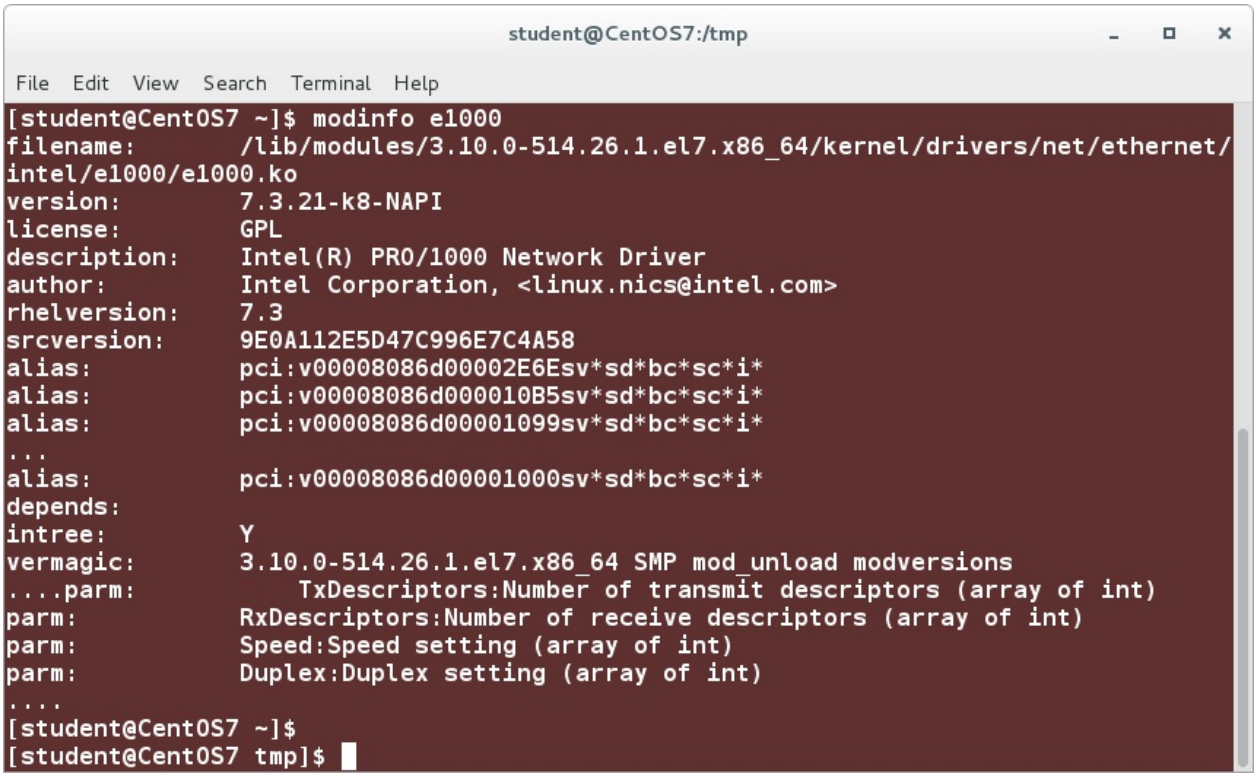
26.9 modinfo

modinfo can be used to find out information about kernel modules (whether or not they are currently loaded):

```
$ /sbin/modinfo my_module
$ /sbin/modinfo <pathto>/my_module.ko
```

Can see example below, which displays information about version, file name, which hardware devices the device driver module can handle, and what parameters can be supplied on loading.

Much information about modules also seen in `/sys` pseudo-filesystem directory tree. In example, would look under `/sys/module/e1000` and some, if not all parameters, can be read and/or written under `/sys/module/e1000/parameters`. Will show hot to set them next.



26.10 Module Parameters

Many modules can be loaded while specifying parameter values:

```
$ sudo /sbin/insmod <pathto>/e1000.ko debug=2 copybreak=256
```

or, for module already in proper system location, easier with:

```
$ sudo /sbin/modprobe e1000e debug=2 copybreak=256
```

26.11 Kernel Module Configuration

Files in `/etc/modprobe.d` directory control some parameters that come into play when loading with **modprobe**. These parameters include module name **aliases** and automatically supplied options. Can also blacklist specific modules o avoid them being used.

Settings apply to modules as they are loaded/unloaded, and configurations can be changed as needs change.

Format of files in `/etc/modprobe.d` simple: one command per line, with blanks lines and lines starting with `#` ignored (useful for adding comments). Backslash at end of line causes it to continue on next line, which makes file a bit neater.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 27 Devices and udev - Notes

27.2 Introduction

Linux uses **udev**, an intelligent apparatus for discovering hardware and peripheral devices both during boot, and later when they are connected to the system. **Device nodes** are created automatically, and then used by applications/operating system subsystems to communicate with + transfer data to/from devices. System administrators can control how **udev** operates and craft special **udev** rules to assure desired behavior results.

27.3 Learning Objectives:

- Explain the role of **device nodes** and how they use **major** and **minor** numbers.
- Understand the need for the **udev** method and list its key components.
- Describe how the **udev** device manager functions.
- Identify **udev** rule files and learn how to create custom rules.

27.4 Device Nodes

Character and block devices have filesystem entries associated with them. Networ k devices in Linux *do not*. These **device nodes** can be used by programs to communicate with devices, using normal I/O system calls such as `open()`, `close()`, `read()`, and `write()`. On other hand, network devices work by transmitting/receiving **packets**, which must be constructed by breaking up streams of data, or reassembled into streams when received.

A device driver may manage multiple device nodes, which are normally placed in `/dev` directory. Can be seen with:

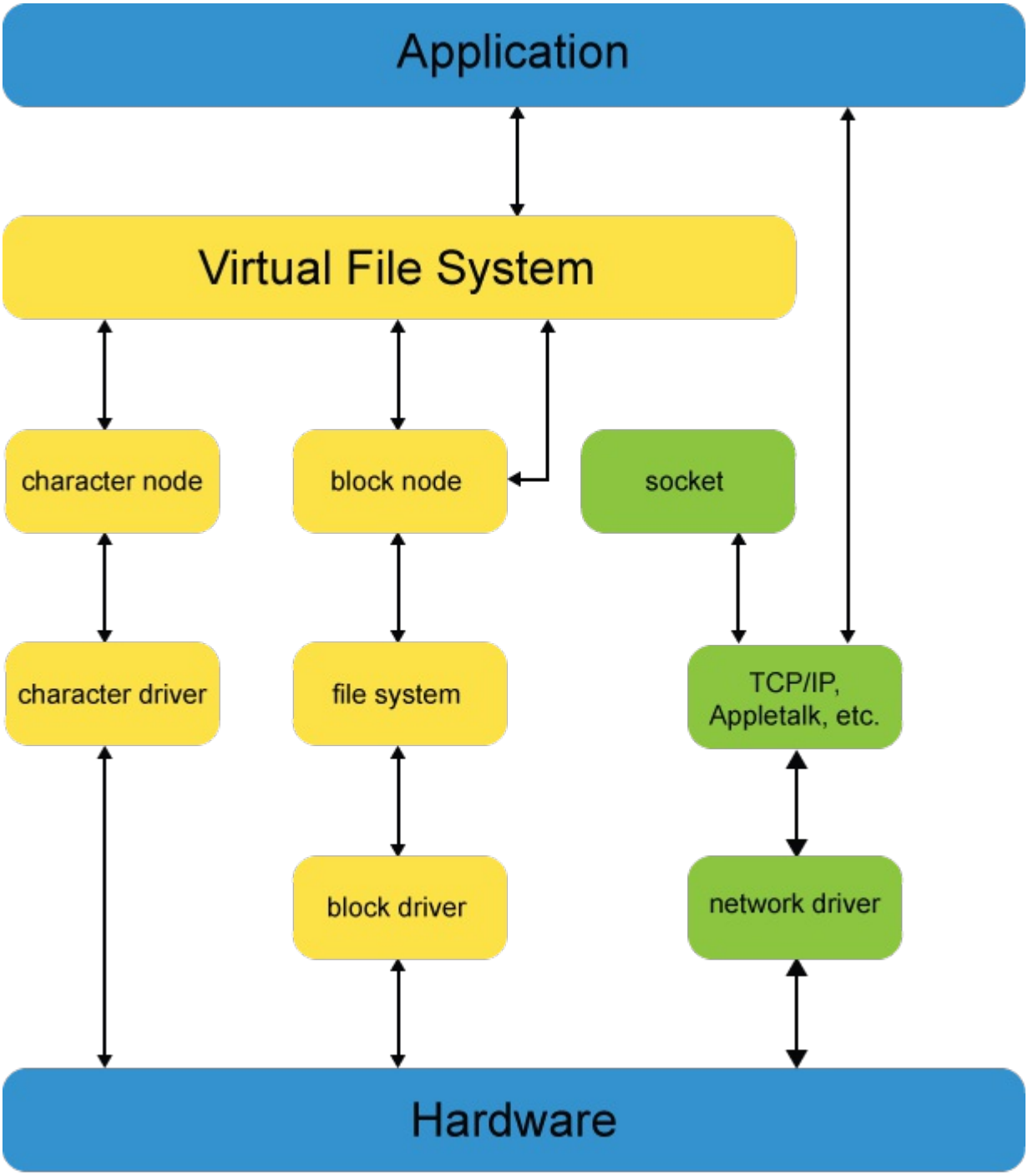
```
$ ls -l /dev
```

Device nodes can be created with:

```
sudo mknod [-m mode] /dev/name <type> <major> <minor>
```

For example:

```
mknod -m 666 /dev/mycdrv c 254 1
```



Device nodes

27.5 Major and Minor Numbers

Major and **minor** numbers identify driver associated with device, with driver uniquely reserving a group of numbers. In most cases (but not all), device nodes of the same type (block or character) with the same major number use the same driver.

If you list some device nodes:

```
$ ls -l /dev/sda*
brw-rw---- 1 root disk 8,  0 Dec 29 06:40 /dev/sda
brw-rw---- 1 root disk 8,  1 Dec 29 06:40 /dev/sda1
brw-rw---- 1 root disk 8,  2 Dec 29 06:40 /dev/sda2
.......
```

Major and **minor** numbers appear in same place that file size would when looking at normal file. In above example as **8**, **1**, etc. While normal users will probably never need to refer explicitly to major and minor numbers and will refer to devices by name, system administrators may have to untangle them from time to time if system gets confused about devices, or has some hardware added at runtime.

Minor number used only by device driver to differentiate between different devices it may control, or how they are used. These may either be different instances of same kind of device (such as first and second sound card, or hard disk partition), or different modes of operation of given device (such as different density floppy drive media).

Device numbers have meaning in user-space as well. Two system calls, `mknod()` and `sta()` return information about **major** and **minor** numbers.

27.6 udev

Methods of managing device nodes became clumsy and difficult as Linux evolved. Number of device nodes lying in `/dev` + its subdirectories reached numbers in 15,000 - 20,000 range in most installations during 2.4 kernel version series. Nodes for devices which would never be used on most installations were still created by default, as distributors could never be sure exactly which hardware would be present on system.

Many developers + system administrators trimmed list to what was actually needed, especially in embedded configurations, but this essentially manual and potentially error-prone task.

Note: while device nodes *not* normal files and don't take up significant space on filesystem, having huge directories slowed down access to device nodes, especially upon first usage. Exhaustion of available major and minor numbers required more modern + dynamic approach to creation/maintenance of device nodes. Ideally, one would like to register devices by name. However, major and minor numbers cannot be gotten rid of altogether, as **POSIX** standard requires them. (POSIX: acronym for **Portable Operating System Interface**, a family of standards designed to ensure compatibility between different operating systems.)

udev method created device nodes on the fly as needed. No need to maintain a ton of device nodes that will never be used. The `u` in **udev** stands for **user**, and indicates that most of the work of creating, removing, modifying nodes is done in user-space.

udev handles dynamical generation of device nodes, evolved to replace earlier mechanisms such as **devfs**, **hotplug**. Supports **persistent device naming**. Names need not depend on order of device connection or plugging in. Such behavior controlled by specification of **udev rules**.

27.7 udev Components

udev runs as **daemon** (either **udevd** or **systemd-udevd**), monitors a **netlink** socket. When new devices initialized/removed, **uevent** kernel facility sends message through the socket, which **udev** receives and takes appropriate action to create/remove device nodes of right names and properties according to the rules.

Three components of **udev**:

1. The **libudev** library which allows access to information about the devices
2. The **udevd** or **systemd-udevd** daemon that manages the `dev` directory
3. The **udevadm** utility for control and diagnostics

Cleanest way to use **udev**: to have a pure system. `dev` directory empty upon initial kernel boot, then populated with device nodes as they are needed. When used this way, must boot using **initramfs** image, which may contain set of preliminary device nodes, as well as **udev** infrastructure.

27.8 udev and Hotplug

As devices added/removed from system, working with the hotplug subsystem, **udev** acts upon notification of events to create/remove device nodes. Information necessary to create them with the right names, major and minor numbers, permissions,

etc., gathered by examination of information already registered in **sysfs** pseudo-filesystem (mounted at `/sys`) and a set of configuration files.

Main configuration file: `/etc/udev/udev.conf`. Contains information such as where to place device nodes, default permissions and ownership, etc. By default, rules for device naming located in `/etc/udev/rules.d` and `/usr/lib/udev/rules.d` directories. BY reading **man** page for **udev**, can get lot of specific information about how to set up rules for common situations.

27.9 The udev Device Manager

When **udev** receives message from kernel about devices being added/removed, it parses the rule-setting files in `/etc/udev/rules.d/*.rules` and `/usr/lib/udev/rules.d/*.rules` to see if there are any rules relevant to device being added/removed.

It then takes appropriate actions including:

- Device node naming
- Device node and symbolic links creation
- Setting file permissions and ownership for the device node
- Taking other actions to initialize and make device available.

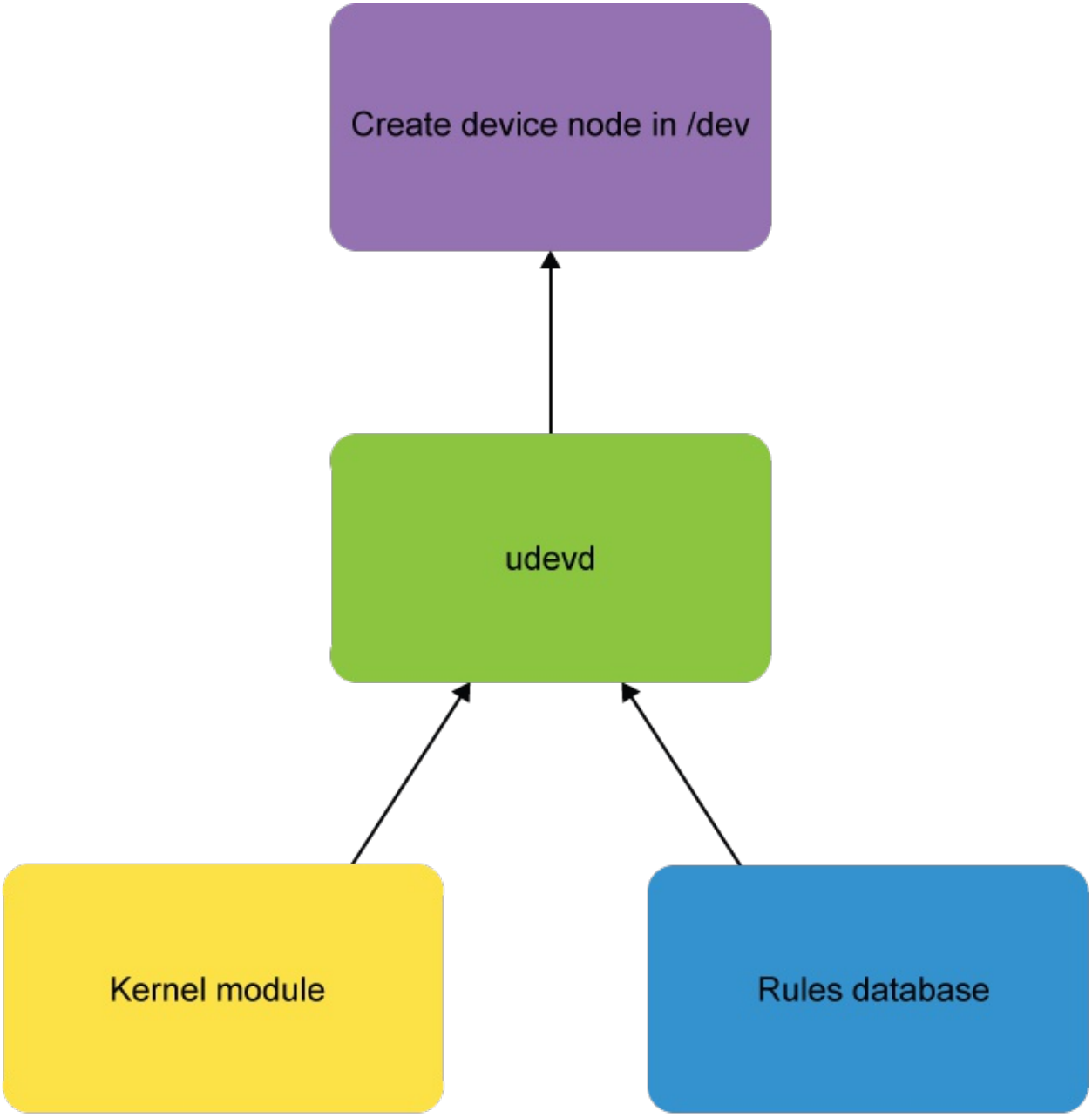
These rules are completely customizable.

27.10 udev Rule Files

udev rules files are located under `/etc/udev/rules.d/<rulename>.rules` with names like:

- `30-usb.rules`
- `90-mycustom.rules`

By default, when **udev** reads the rules files, looks for files that have suffix of `.rules`. If it finds more than one file, reads them one by one, lexicographically, ie. in ascending alphabetical order. Standard rule file name: generally a two digit number followed by descriptive name (for the rules), folowed by `.rules` suffix.



udev Rule

27.11 Creating udev Rules

Format for **udev** rule simple:

```
<match><op>value [, ...] <assignment><op>value [, ...]
```

Two separate files defined on single line:

- First part consists of one or more match pairs denoted by `==`. These try to match device's attributes and/or characteristics to some value
- Second part consists of one or more assignment key-value pairs that assign a value to a name, such as a file name, group assignment, even file permissions etc.

If no matching rule found, uses default device node name and other attributes.

27.12 Some Examples of Rules Files

Example of rules file for **Fitbit** device:

```
$ cat /usr/lib/udev/rules.d/99-fitbit.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", SYMLINK+="fitbit", MODE="0666"
```

Example for creating crash dumps and fast kernel loading with **kdump**/**kexec**:

```
$ cat /usr/lib/udev/rules.d/98-kexec.rules
SUBSYSTEM=="cpu", ACTION=="online", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="cpu", ACTION=="offline", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="add", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="remove", PROGRAM="/bin/systemctl try-restart kdump.service"
```

Example for kvm virtual machine hypervisor:

```
$ cat /usr/lib/udev/rules.d/80-kvm.rules
KERNEL=="kvm", GROUP="kvm", MODE="0666"

$ cat /usr/lib/udev/rules.d/99-fuse.rules
KERNEL=="fuse", MODE="0666",OWNER="root",GROUP="root"
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 28 Virtualization Overview - Notes

28.2 Introduction

Virtualization entails creating (in software) an abstracted complete machine environment, under which end user software runs unaware of the physical details of the machine it is running on.

Quite a few approaches to accomplish this mission, but generally involve host operating system and filesystem which directly uses hardware. One or more guest systems run on top (or underneath, depending on how you look at it) of host system at lower level of privilege. Access to hardware such as network adapters usually done through some kind of virtual device, which may access physical device when required.

Many uses for virtual machine. Major ones: more efficient use of hardware, software isolation, security, stability, safe development options, ability to run multiple operating systems at same time without rebooting.

28.3 Learning Objectives:

- Understand the concepts behind **virtualization** and how it came to be widely used.
- Understand the rules of **hosts** and **guests**.
- Discuss the difference between **emulation** and **virtualization**.
- Distinguish between the different types of **hypervisors**.
- Be familiar with how Linux distributions use and depend on **libvirt**.
- Use the qemu hypervisor.
- Install, use, and manage KVM.

28.4 What Is Virtualization?

In this chapter, will be concerned with creation, deployment, and maintenance of Virtual Machines (VMs).

Virtual Machines: virtualized instance of an entire operating system, may fulfill the role of a **server** or a **desktop/workstation**.

Outside world sees the VM as if it were actual physical machine, present somewhere on network. Applications running in VM generally unaware of their non-physical environment.

Other kinds of virtualizations include:

- **Network**

  Details of actual physical entwork, such as type of hardware, routers, etc. abstracted and need not be known by software running on it and configuring it: Software Defined Networking or SDN.

- **Storage**

  Multiple network storage devices configured to look like one big storage unit, such as disk: Network Attached Storage or NAS.

- **Application**

  Application isolated into standalone format, such as **container**, which will be discussed later.

There are differences between physical and virtual machines that are not always easy to ignore. Eg. performance tuning at low level needs to be done (separately) on both VM and physical machine residing underneath it.

28.5 Virtualization History

Virtualization has long proud history. Implemented originally on mainframes decades ago:

- Enables better hardware utilization
- Operating systems often progress more quickly than hardware
- It is microcode driven
- Not particularly user-friendly

Later on, virtualization technology migrated to PCs and workstations:

- Initially, it was done using **Emulation**
- CPUs enhanced to support virtualization led to boost in performance, easier configuration, more flexibility in VM installation and migration

From early **mainframes** to **mini-computers**, virtualization has been used for expanding limit, debugging and administration enhancements.

Today, virtualization found everywhere in many forms. Each of the different forms and implementations that are available provide particular advantages.

28.6 Hosts and Guests

Host: underlying physical operating system managing one or more virtual machines.

Guest: the VM which is an instance of a complete operating system, running one or more applications. Sometimes, guest also called **client**. In most cases, guest should not care what host it is running on, can be **migrated** from one host to another, sometimes while actually running.

In fact, possible to convert physical machines to virtual ones, by copying the entire contents into a VM. Specialized software utilities can make this easier.

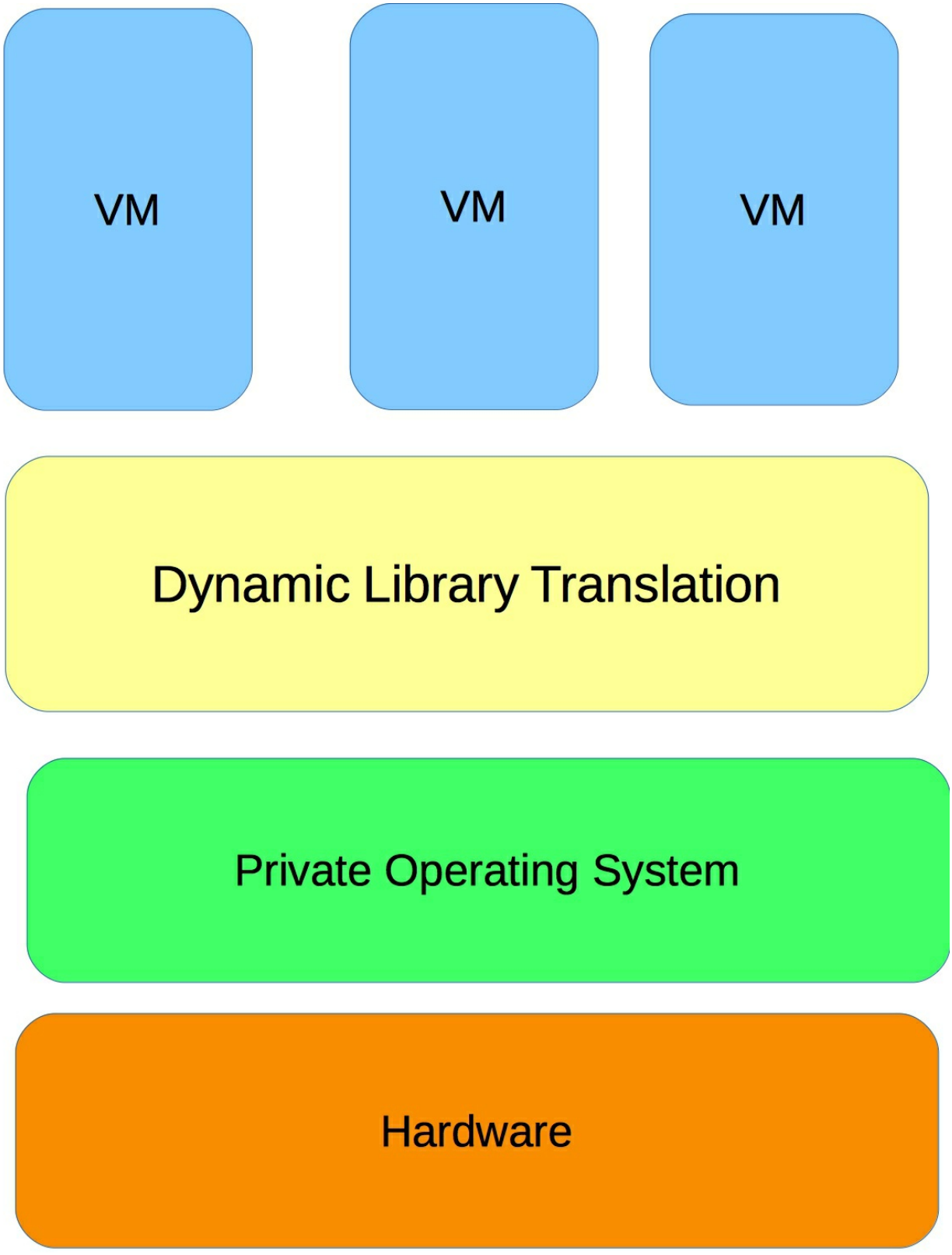
Low-level performance tuning on areas such as CPU utilization, networking throughput, memory utilization, often best done on host, as guest may have only simulated quantities not directly useful.

Application tuning done most on guest.

28.7 Emulators

First implementations of virtualization on PC architecture -> through the use of **emulators**. Application running on current operating system would appear to another OS as specific hardware environment. Emulators generally do not require special hardware to operate.

Qemu -> one such emulator.



Emulator

28.8 Emulation vs. Virtualization

An Emulator runs completely in softw are. Hardw are constructs replaced by softw are. Useful for running virtual machines on different architectures, eg. running pretend ARM guest machine on X86 host. Emulation often used for developing operating system for new CPU, even before hardw are is available. Performance relatively slow.

28.9 Types of Virtualization Hypervisors

Host system (besides functioning normally with respect to software that it runs) also acts as hypervisor that initiates, terminates, manages guests. Also called **Virtual Machine Monitor** (**VMM**).

Two basic methods of virtualization:

- **Hardware virtualization** (also known as **Full Virtualization**)

  Guest system runs without being aware it is running as virtualized guest, does not require modifications to be run in this fashion.

- **Para-virtualization**

  Guest system is aware it is running in virtualized environment, has been modified specifically to work with it.

Recent CPUs from Intel and AMD incorporate virtualization extensions to the x86 architecture that allow the hypervisor to run fully virtualized (ie. unmodified) guest operating systems with only minor performance penalties.

The Intel extension (Intel Virtualization Technology), usually abbreviated as VT, IVT, VT-32, or VT-64, also known under the development code name of Vanderpool. Has been available since the spring of 2005.

The AMD extension usually called AMD-V, still sometimes referred to by the development code name of Pacifica.

For detailed explanation and comparison of how these two extensions work, see the Xen and the new processors article.

Can check directly if your CPU supports hardware virtualization extensions by looking at `/proc/cpuinfo`. If you have an IVT-capable chip, will see `vmx` in `flags` field. If you have an AMD-V capable chip, will see `svm` in same field. May also have to ensure virtualization capability is turned on in your CMOS.

While choice of operating systems tends to be more limited for para-virtualized guests, originally they tended to run more efficiently than fully virtualized guests. Recent advances in virtualization techniques have narrowed or eliminated such advantages, and the wider availability of the hardware support needed for full virtualization has made para-virtualization less advantageous and less popular.

Most modern hardware has hardware virtualization abilities (must be turned on in BIOS).

28.10 External and in-Kernel Hypervisors

Hypervisor can be:

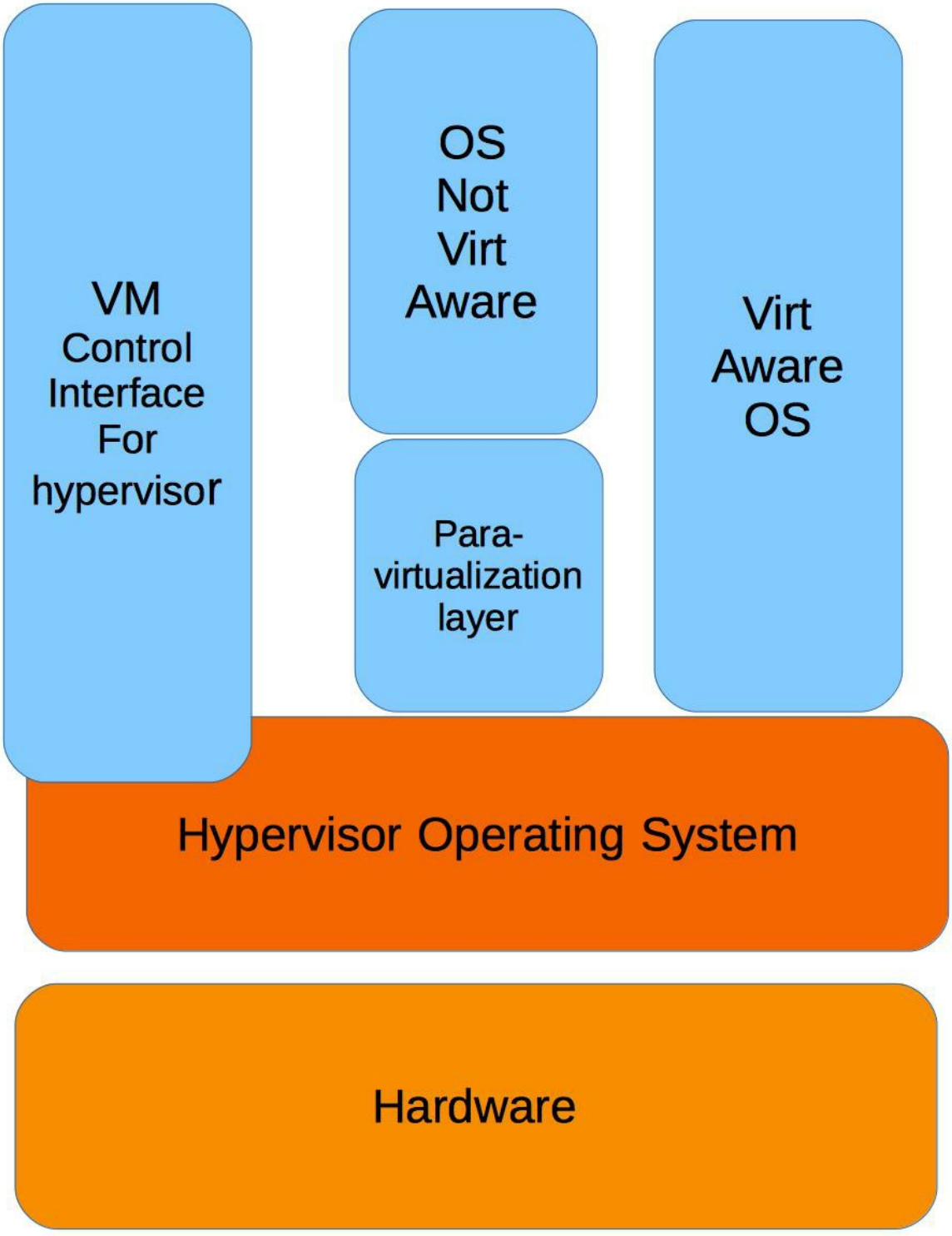
- External to the host operating system kernel: VMware
- Internal to the host operating system kernel: KVM

In this course, will concentrate on KVM, as it is all open source, and requires no external third party hypervisor program.

28.11 Dedicated Hypervisor

Going past emulation, merging of hypervisor program into specially-designed lightweight kernel was next step in Virtualization deployment.

VMware ESX (and related friends): an example of hypervisor embedded into operating system.

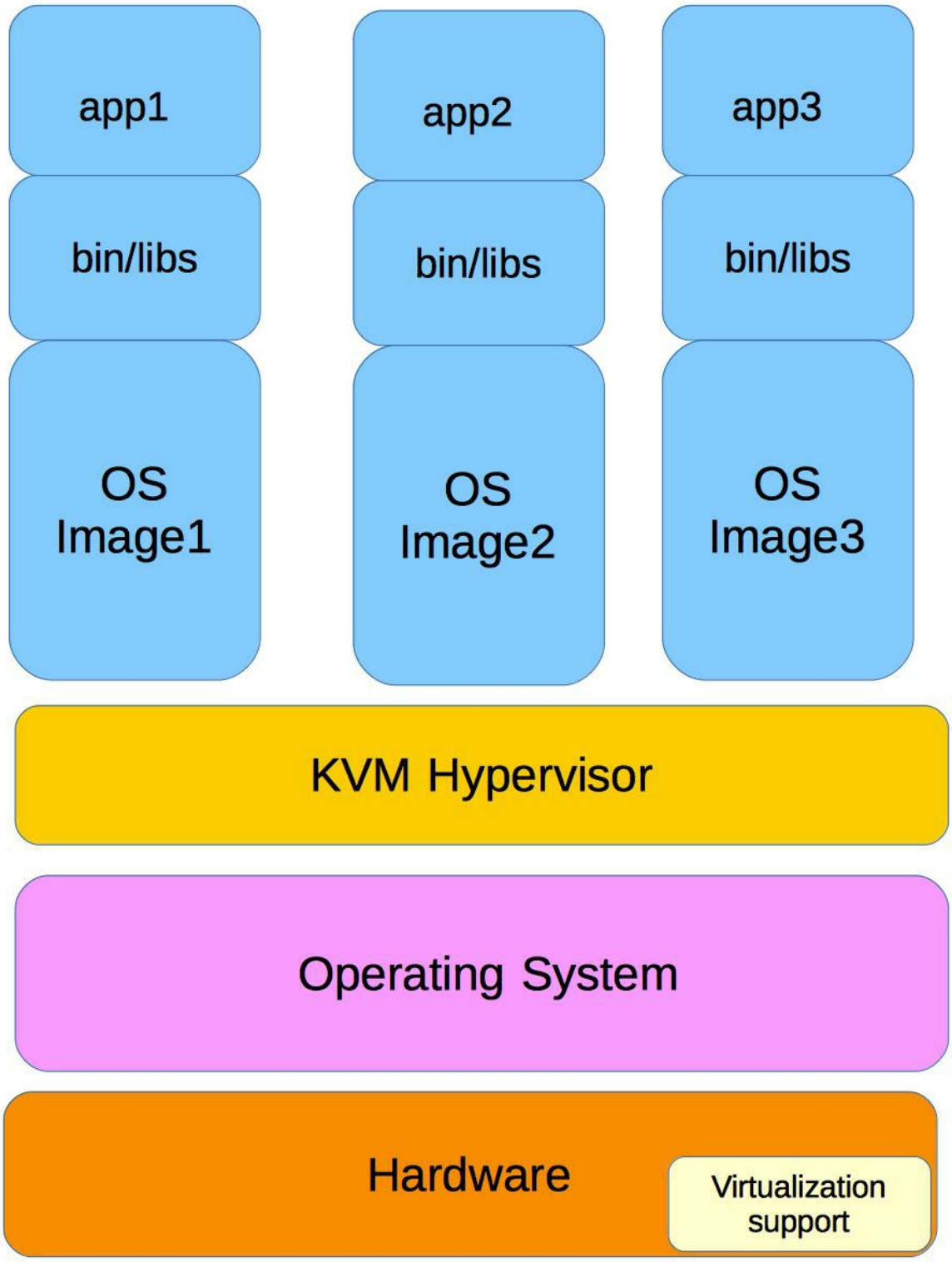


Dedicated Hypervisor

28.12 Hypervisor in the Kernel

The KVM project added hypervisor capabilities into the Linux kernel.

As discussed, specific CPU chip functions and facilities were required and deployed for this type of virtualization.



Hypervisor in the Kernel

28.13 libvirt

The libvirt project: toolkit to interact w ith virtualization technologies. Provides management for virtual machines, virtual netw orks, and storage. Available on all enterprise Linux distributions.

Many application programs interface w ith libvirt. Some of the most common ones: **virt-manager**, **virt-viewer**, **virt-install**,

virsh.

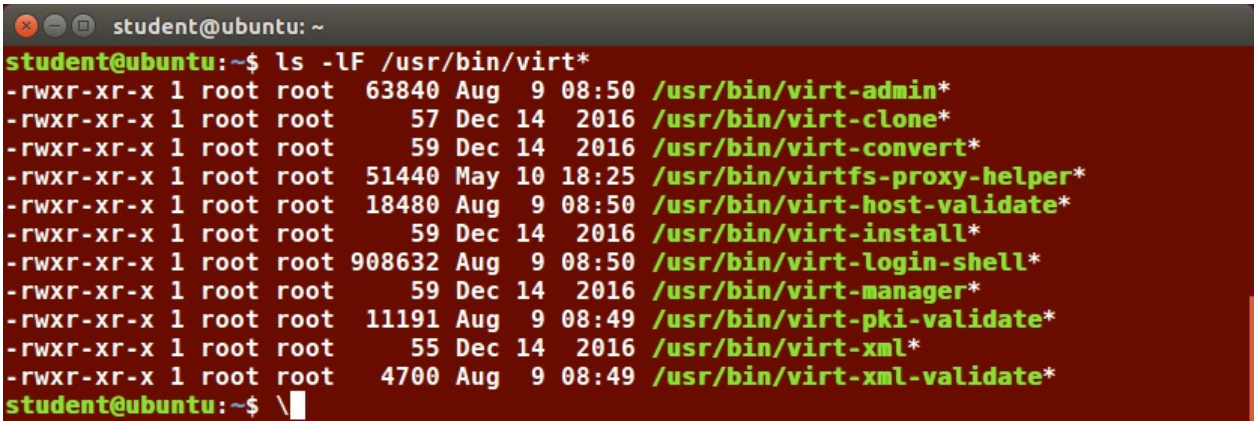
Complete list of currently supported hypervisors can be found on libvirt website:

- QEMU/KVM
- Xen
- Oracle VirtualBox
- VMware ESX
- VMware Workstation/Player
- Microsoft Hyper-V
- IBM PowerVM (phyp)
- OpenVZ
- UML (User Mode Linux)
- LXC (Linux Containers)
- Virtuozzo
- Bhyve (The BSD Hypervisor)
- Test (Used for testing)

28.14 Programs Using libvirt

Many utilities using **libvirt**. Exact list will depend on Linux distribution. Full list can be found on libvirt website.

In this course, will work through the use of robust GUI, **virt-manager**, rather than make much use of command line utilities, which lead to more flexibility, as well as use on non-graphical servers.



28.15 What is QEMU?

QEMU stands for **Q**uick **EMU**lator. Originally written by Fabrice Bellard in 2002. (Bellard is also known for feats such as holding, at one point, the world record for calculating **π**, reaching 2.7 trillion digits.)

QEMU: hypervisor that performs hardware **emulation**, or **virtualization**. Emulates CPUs by dynamically translating binary instructions between host architecture and emulated architecture.

Host and emulated architectures may be different, or the same. Numerous choices for both host and guest operating systems.

QEMU can also be used to emulate just particular applications, not entire operating systems.

By itself, QEMU much slower than host machine. But, can be used together with **KVM** (**K**ernel **V**irtual **M**achine) to reach speeds close to those of native host.

Guest operating systems do not require rewriting to run under QEMU. QEMU can save, pause, restore a virtual machine at any time. QEMU is a free software licensed under the GPL.

QEMU has the capability of supporting many architectures, including: IA-32 (i386), x86-64, MIPS, SPARC, ARM, SH4, PowerPC, CRIS, MicroBlaze, etc.

Cross-compilation abilities of QEMU make it extremely useful when doing development for embedded processors.

In fact, QEMU has often been used to develop processors which have either not yet been physically produced, or released to market.

28.16 Third Party Hypervisor Integration

Used by itself, QEMU relatively slow. However, can be integrated with third party hypervisors, and then reach near native speeds. Note: some of these systems are very close cousins of QEMU; others are more remote. A few main ones:

- KVM offers particularly tight integration with QEMU. WHen host and target are same architecture, full acceleration and high speed delivered. KVM native to **Linux**. Will discuss this is detail.
- Xen, also native to Linux, can run in hardware virtualization mode if architecture offers it, as does x86 and some ARM variants.
- Oracle Virtual Box can use qcow2 formatted images, and has very close relationship with QEMU.

In this course, recommend using (and will use in labs) **virt-manager** to configure and run virtual machines. Will also give instructions on how to run them using **qemu** command line utilities.

28.17 Image Formats

QEMU supports many formats for disk image files. However, only two used primarily, with the rest being available for historical reasons and for conversion utilities. These two main formats:

- **raw** (default)

  The simplest and easiest format to export to other non-QEMU emulators. Empty sectors do not take up space.

- **qcow2**

  COW stands for **C**opy **O**n **W**rite. Many options. See **man qemu-img** for more details.

To get list of supported formats:

```
c7:/tmp> qemu-img --help | grep formats:
Supported formats: vfat vpc vmdk vhdx vdi ssh sheepdog rbd raw host_cdrom host_floppy host_device file \
qed qcow2 qcow parallels nbd iscsi gluster dmg tftp ftps ftp https http cloop bochs blkverify blkdebug
```

Note in particular:

- **vdi**: Used by Oracle VirtualBox
- **vmdk**: Used by VMware

Note: `qemu-img` can be used to translate between formats, eg.:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

28.18 KVM and Linux

KVM uses the Linux kernel for computing resources including memory management, scheduling, synchronization, and more. When running virtual machine, KVM engages in co-processing relationship with Linux kernel.

In this format, KVM runs the virtual machine monitor within one or more of the CPUs, using VMX or SVM instructions. At the same time, Linux kernel is executing on other CPUs.

The virtual machine monitor runs the guest, which is running at full hardware speed, until it executes an instruction that causes the virtual machine monitor to take over.

At this point, the virtual machine monitor can use any Linux resource to emulate a guest instruction, restart the guest at the last instruction, or do something else.

By loading the KVM modules and starting a guest, turn Linux into hypervisor. The Linux personality is still there, but also have the hardware virtual machine monitor. Can control the Virtual Machine using standard Linux resource and process control tools, such as **cgroups**, **nice**, **numactl**, etc.

KVM first appeared (pre-merge) as part of a Windows Virtual Desktop product. At the time of its upstream merge in 2007, KVM required recent x86-64 processors. On x86-64 platforms, KVM primarily (but not always) a driver for the processor's virtualization subsystem.

KVM appeared as a trio of Linux kernel modules in 2007. When paired with modified version of QEMU (also provided at the same time), created a hypervisor that used the Linux kernel for most of its runtime services.

Shortly after Avi Kivity, the author of KVM, submitted source code to the Linux development community, Linus merged KVM into his Linux tree. This was surprising to a lot of folks.

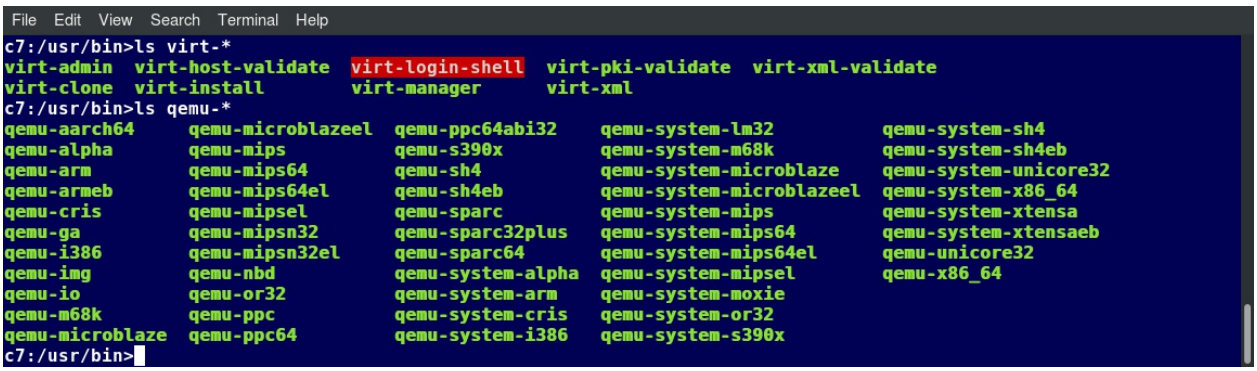
28.19 Managing KVM

Many low level commands for creating, converting, manipulating, deploying, maintaining virtual machine images.

Managing KVM can be done with both command line and graphical interfaces.

Command line tools include: **virt-*** and **qemu-***. Graphical interfaces include virt-manager, kimchi, OpenStack, oVirt, etc.

As you develop more expertise, will become practiced in using them. But, for all basic operations, **virt-manager** will suffice and that is what we will use.



##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 29 Containers Overview - Notes

29.2 Introduction

Containers offer light w eight method to isolate application/set of applications from other running processes. Unlike virtual machines (each constitutes complete operating system), multiple containers can run simultaneously on one system (either virtualized or physical).

Very common method of deploying containers: using Docker, w hich w ill be discussed in some detail.

29.3 Learning Objectives:

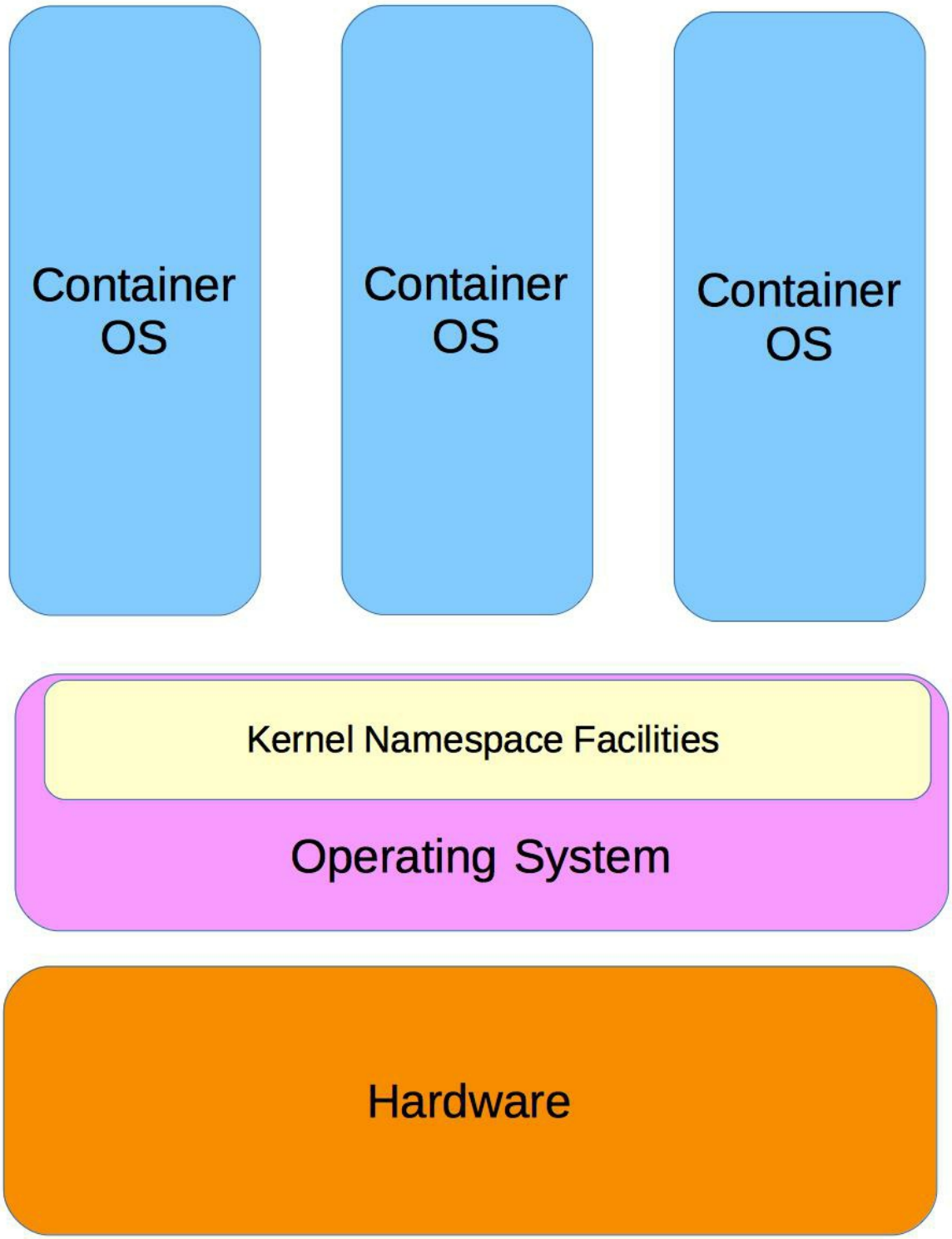
- Understand the basic parameters and methods that define containers.
- Understand the difference betw een containers and virtual machines.
- Be familiar w ith Docker and know the steps necessary to use it properly.
- Use the major commands associated w ith containers and Docker.

29.4 Container Basics

Further integration betw een hypervisor and Linux kernel allow ed creation of operating system-level virtual machines, or containers. Containers share many facilities in Linux kernel, make use of some recent kernel additions such as **namespaces** and **cgroups**. Containers very lightw eight, reduce overhead associated w ith having w hole virtual machines.

First flavor of containers: **OS container**. This type of container runs image of operating system w ith ability to run **init** processes and spaw n multiple applications.

One example: LXC (Linux Containers)

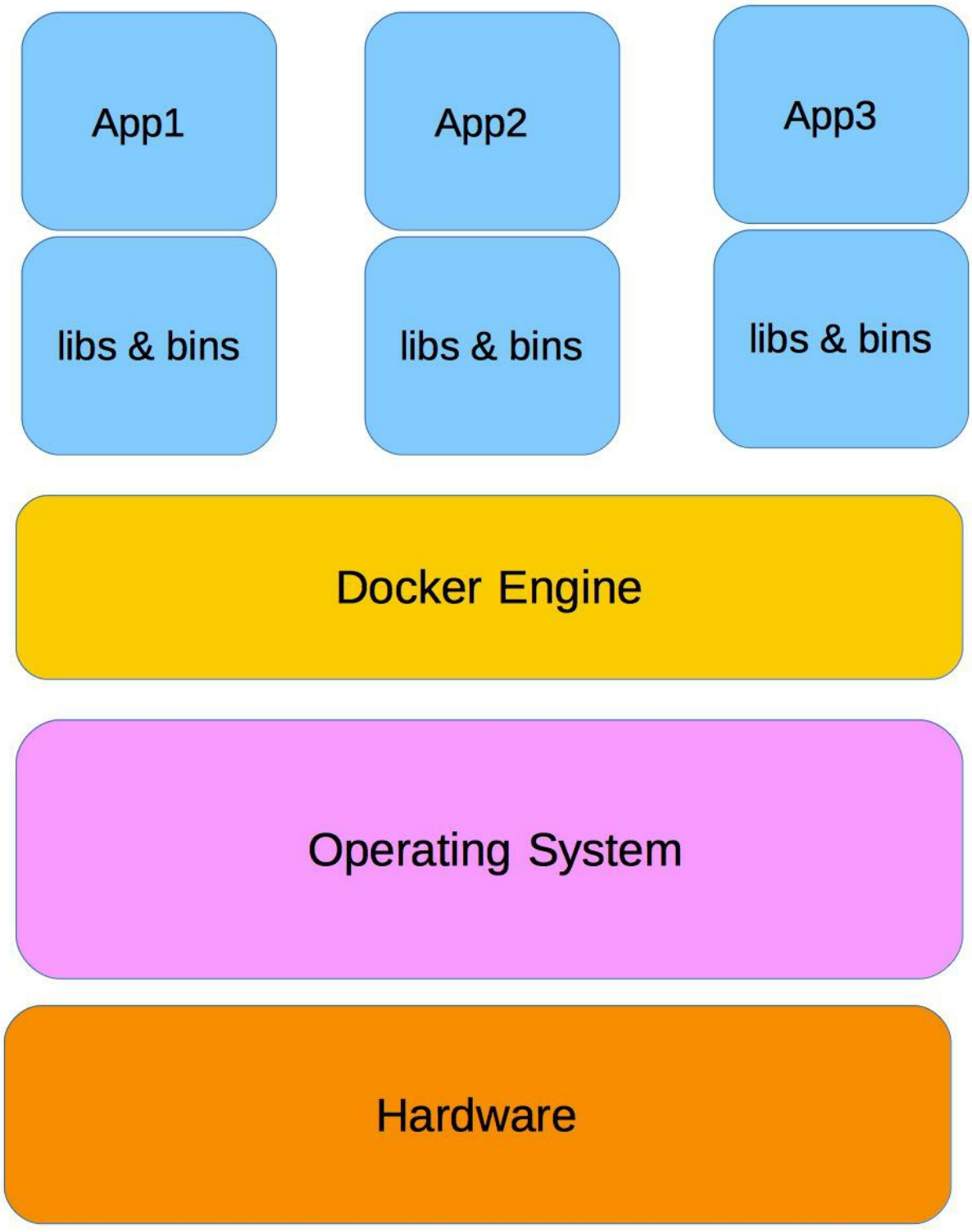


Containers

29.5 Application Virtualization

To further reduce overhead associated w ith virtual machines, **application virtualization** rising in popularity. Application virtualization runs one application for each container. Many single application containers typically initialized on single machine. Using smaller components creates greater flexibility, reduces overhead normally associated w ith virtualization.

One such project: Docker



29.6 Containers vs. Virtual Machines

Both Virtual Machines and Containers satisfy important needs. Each have use, seen as way to go for almost everything.

Both have long histories:

- Mainframe computers have had softw are partitioning and virtual machines for decades, in rather specialized w ays
- Operating systems have had **chroot** and **BSD Jail** implementations for many years, that share basic isolation motivation w ith containers

If number of different services and applications need to be tightly integrated, virtual machine functioning as a service may be best solution.

If applications being run w ere w ritten expecting a complete operating system environment w ith w ide range of services and other libraries/applications, virtual machines may be best.

Virtual machines run complete operating systems, and can run many services and applications. Virtual machines use more resources than containers. Containers usually run one thing. Containers more portable, and can be run inside VM. Containers harder to secure, but usually start faster. Multiple containers ahre one OS kernel, w hile each VM has its ow n.

Scaling w orkloads different for containers and virtual machines. **Orchestration** systems such as Kubernetes or Mesos can decide on proper quantity of containers needed, do load balancing, replicate images, and remove them etc. as needed.

Overall, virtual machines still often the best solution.

29.7 Docker

Docker: application-level virtualization using many individual images to build up necessary services to support target application. Images packages into containers - they are components in containers. Images may contain:

- Application code
- Runtime libraries
- System tools
- Or just about anything required for an application

Images may reside on Docker Hub or registry server.

Docker w ebsite has documentation, tutorials, and training information.

One of the most compelling features of Docker: an application can be packaged up w ith all of its dependent code/services, deployed as single unit w ith minimum of overhead. This deployment can be easily repeated as often as desired. Reduces requirement of building up server w ith layered services to support end application.

29.8 Docker Steps

Starting up a Docker containerized application may include only few steps:

- Install Docker service package w ith your favorite tool
- Start Docker service
- Search for appropriate image from Docker Hub or private repository
- Pull the image
- Run the image
- Finally, test application

Steps detailed above just a very minimal example of testing a Docker application.

Many options that may be used, including functions that create an image, set system variables or configuration parameters, then store the result as new image. In some cases, w ritable image required, rather than non-w ritable one.

Most Docker commands have individual **man** pages. Examples include: `docker(1)`, `docker-search(1)`, `docker-pull(1)`, `docker-create(1)`, `docker-run(1)`.

29.9 docker Command

docker command has more than 40 sub-commands, some with 50 or more options. To learn more about **docker** command, can always do:

```
$ docker <command> --help
```

Many **docker** sub-commands tend to be somewhat self-documenting. Often confused: `run`, `create`, and `exec`. `ps` command will list running containers, or all containers if you include `-all` option.

`docker run` will start new container and execute a command within. Common options: `-t` attached to **tty**, `-d` to run the container in the background.

`docker create` command creates a container. Has many options for configuring container settings and attachments.

If container already running, and you want to execute something inside of it, can use `docker exec` command. Also accepts `-t` and `-d` options.

`docker image` command will show images in various outputs. `docker rmi` command will remove images and delete untagged parents by default.

Can also leverage shell functions to operate upon all containers. Eg. to remove all stopped containers, can do:

```
$ docker rm $(docker ps -a -q)
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 30 User Account Management - Notes

30.3 Learning Objectives:

- Explain the purpose of individual user accounts and list their main attributes.
- Create new user accounts and modify existing account properties, as well as remove of lock accounts.
- Understand how user passwords are set, encrypted, and stored, and how to require changes in passwords over time for security purposes.
- Explain how restricted shells and restricted accounts work.
- Understand the role of the root account and when to use it.
- Use Secure Shell (**ssh**) and remove logins and commands.

30.4 User Accounts

Linux systems provide **multi-user** environment which permits people and processes to have separate simultaneous work environments.

Purposes of having individual user accounts include:

- Providing each user with their own individualized private space
- Creating particular user accounts for specific dedicated purposes
- Distinguishing privileges among users

One special user account is for the **root** user, who is able to do ***anything*** on the system. To avoid making costly mistakes, and for security reasons, the root account should only be used when absolutely necessary.

Normal user accounts are for people who will work on the system. Some user accounts (like the **daemon** account) exist for the purpose of allowing processes to run as a user other than root.

In next chapter, will continue with discussion of **group** management, where subsets of the users on system can share files, privileges, etc., according to common interests.

30.5 Attributes of a User Account

Each user on the system has a corresponding line in the `etc/passwd` file that describes their basic account attributes. (Will talk about passwords, as well as this file, later). For example:

```
....
beav:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
warden:1001:1001:Ward Cleaver:/home/warden:/bin/bash
dobie:1002:1002:Dobie Gillis:/home/dobie:/bin/bash
....
```

The seven elements here are:

- User name: unique name assigned to each user
- User password: password assigned to each user
- User identification number (**UID**): unique number assigned to user account. UID is used by system for variety of purposes,

including determination of user privileges and activity tracking
- Group identification number (**GID**): indicates primary, principal, or default **group** of user
- Comment or GECOS information: defined method to use comment field for contact information (full name, email, office, contact number) (No need to worry about meaning of **GECOS**, very old term)
- Home directory: for most users, unique directory that offers working are for user. Normally, this directory owned by user, and except for **root** will be found on the system somewhere under `/home`
- Login shell: normally, shell program such as `/bin/bash` or `/bin/csh`. Sometimes, however, alternative program referenced here for special cases. In general, this field will accept any executable

30.6 Creating User Accounts with useradd

The command:

```
$ sudo useradd dexter
```

will create an account for user `dexter`, using default algorithms for assigning user and group id, home directory, and shell choice.

Specifically, the **useradd** command above causes following steps to execute:

- Next available UID greater than `UID_MIN` (specified in `/etc/login.defs`) by default assigned as `dexter`\'s `UID`
- A group called `dexter` with a `GID=UID` also created and assigned as `dexter`\'s primary group
- A home directory `/home/dexter` created and owned by `dexter`
- `dexter`\'s login shell will be `/bin/bash`
- Contents of `/etc/skel` copied to `/home/dexter`. By default, `/etc/skel` includes startup files for **bash** and for the X Window system
- Entry of either `!!` placed in password field of `/etc/shadow` file for `dexter`\'s entry, thus requiring administrator to assign a password for the account to be usable

Defaults can be easily overruled by using options to **useradd**:

```
$ sudo useradd -s /bin/csh -m -k /etc/skel -c "Bullwinkle J Moose" bmoose
```

where explicit non-default values have been given for some of the user attributes.

30.7 Modifying and Deleting User Accounts

Root user can remove user accounts using **userdel**:

```
$ sudo userdel morgan
```

All references to user `morgan` will be erased from `/etc/passwd`, `/etc/shadow`, and `/etc/group`.

While this removes account, does not delete the home directory (usually `/home/morgan`) in case the account may be re-established later. If `-r` option given to **userdel**, home directory will also be obliterated. However, all other files on system owned by removed user will remain.

usermod can be used to change characteristics of a user account, such as group memberships, home directory, login name, password, default shell, user id, etc. For example, the command

```
$ sudo usermod -L dexter
```

locks the account for `dexter`, so he cannot login.

Usage pretty straightforw ard. Note: **usermod** w ill take care of any modifications to files in `/etc` directory as necessary.

usermod

30.8 Locked Accounts

Linus ships w ith some system accounts that are **locked** (such as `bin`, `daemon`, `sys`), w hich means they can run programs, but can never login tot he system and have no valid passow ord associated w ith them. For example, `etc/passwd` has entries like:

```
bin:1:1:bin:/bin:/sbin/nologin
daemon:2:2:daemon:/sbin:/sbin/nologin
```

The `nologin` shell returns the follow ing if a locked user tries to login to the system:

```
This account is currently not available.
```

or w hatever message may be stored in `/et/nologin.txt`.

Such locked accounts are created for special purposes, either by system services or applications; if you scan `/etc/passwd` for users w ith the `nologin` shell, you can see w ho they are on your system.

Also possible to lock account of particular user:

```
$ sudo usermod -L dexter
```

w hich means the accounts stays on the system but logging in is impossible. Unlocking can be done w ith `-U` option.

Customary practice: lock user\'s accounts w henever they leave the organization or is on an extended leave of absence.

Another w ay to lock an account: use **chage** to change expiration date of account to date in the past:

```
$ sudo chage -E 2014-09-11 morgan
```

Actual date irrelevant as long as it is in the past. Will discuss **chage** shortly.

Another approach is to edit `etc/shadow` file and replace user\'s hashed passw ord w ith `!!` or some other invalid string.

30.9 User IDs and /etc/passwd

Have already seen how `etc/passwd` containsn one record (one line) for each user on system:

```
beav:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
rsquirrel:1001:1001:Rocket J Squirrel:/home/rsquirrel:/bin/bash
```

and have already discussed the fields in here. Each record consists of number of fields separated by colons:

- `username` - the user\'s unique name
- `password` - either the hashed passw ord (if `/etc/shadow` is not used) or a placeholder ("x" w hen `/etc/shadow` is used)
- `UID` - user identification number
- `GID` - primary group identification number for the user
- `comment` - comment area, usually the user\'s real name
- `home` - directory pathname for the user\'s home directory
- `shell` - absolutely qualified name of the shell to invoke at login

If `/etc/shadow` is not used, passw ord field contains hashed passw ord. If used, contains a placeholder ("**x**").

The convention most Linux distributions have used is that any account w ith a user ID less than `1000` is considered special and belongs to the system; normal user accounts start at `1000`. Actual value defined as `UID-MIN` and is defined in `/etc/login.defs`.

If User ID if not specified w hen using **useradd**, system w ill incrementally assign UIDS starting at `UID_MIN`

Additionally, each user gets a **Primary Group ID** w hick, by default, is the same number as the UID. These are sometimes called **User Private Groups (UPG)**.

Bad practice to edit `/etc/passwd`, `/etc/group`, `/etc/shadow` directly. Either user appropriate utilities such as **usermod**, or use **vipw** special editor to do so as it is careful about file locking, data corruption, etc.

30.10 Why Use /etc/shadow?

Use of `/etc/shadow` enables passw ord aging on a per user basis. At same time, also allow s for maintaining greater security of hashed passw ords.

Default permissions of `/etc/passwd` are **644** (`-rw-r--r--`); anyone can read the file. Unfortunately necessary because system programs and user applications need to read information contained in file. These system programs do not run as user root; in any event, only root may change the file.

Of particular concern: hashed passw ords themselves. If appear in `/etc/passwd`, anyone may make copy of hashed passw ords and then make use of utilities such as **Crack** and **John the Ripper** to guess original cleartext passw ords given hashed passw ord. Huge security risk!

`etc/shadow` has permission settings of `400` (`-r--------`), means that only root can access this file. Makes it more difficult for someone to collect hashed passw ords.

Unless compelling good reason not to, should use `/etc/shadow` file.

30.11 /etc/shadow

`/etc/shadow` contains one record (one line) for each user:

```
daemon:*:16141:0:99999:7:::
.....
beav:$6$iCZyCnBJH9rmq7P.$RYNm10Jg3wrhAtUnahBZ/mTMg.RzQE6iBXyqaXHvxxbKTYqj.d9wpoQFuRp7fPEE3hMK3W2gcIYhiXa9MIA9w1:16316:0:99
```

Colon-separated fields:

- `username`: unique user name

- `password` : hashed (**sha512**) value of the passw ord
- `lastchange` : days since Jan 1, 1970 that passw ord w as last changed
- `mindays` : minimum days before passw ord can be changed
- `maxdays` : maximum days after w hich passw ord must be changed
- `warn` : days before passw ord expires that user is w arned
- `grace` : days after passw ord expires that account is disabled
- `expire` : date that account is/w ill be disabled
- `reserved` : reserved field

Username in each record must match *exactly* that found in `etc/passwd` , and also must appear in identical order.

All dates stored as number of days since Jan. 1 1970 (**epoch** date).

Passw ord hash: string "**6**" follow ed by eight character salt value, w hich is then follow ed by **$** and an **88** character (**sha512**) passw ord hash.

30.12 Password Management

Passow rds can be changed w ith **passwd**; a normal user can change only their ow n passw ord, w hile root can change any user passw ord. When you type your passw ord, not show n; echoing back to screen suppressed.

By default, passw ord choice examined by `pam_cracklib.so` , w hich furthers making good passw ord choices.

Normal user changing passw ord:

```
$ passwd
Changing password for clyde
(current) UNIX password: <clyde\'s password>
New UNIX password: <clyde\'s-new-password>
Retype new UNIX password: <clyde\'s-new-password>
passwd: all authentication tokens updated successfully
```

Note: w hen root changes a user\'s passw ord, root not prompted for current passw ord:

```
$ sudo passwd kevin
New UNIX password: <kevin\'s-new-password>
Retype new UNIX password: <kevin\'s-new-password>
passwd: all authentication tokens updated successfully
```

Note: normal users w ill not be allow ed to set bad passw ords, such as ones that are too short, or based on dictionary w ords. How ever, root allow ed to do so.

30.13 chage: Password Aging

Generally considered important to change passw ords periodically. Limits amount of time cracked passw ord can be useful to intruder and also can be used to lock unused accounts. Dow nside: user can find policy annoying, w ind up w riting dow n ever-changing passw ords, thus making them easier to steal.

Utility to manage this: **chage**:

```
chage [-m mindays] [-M maxdays] [-d lastday] [-I inactive] [-E expiredate] [-W warndays] user
```

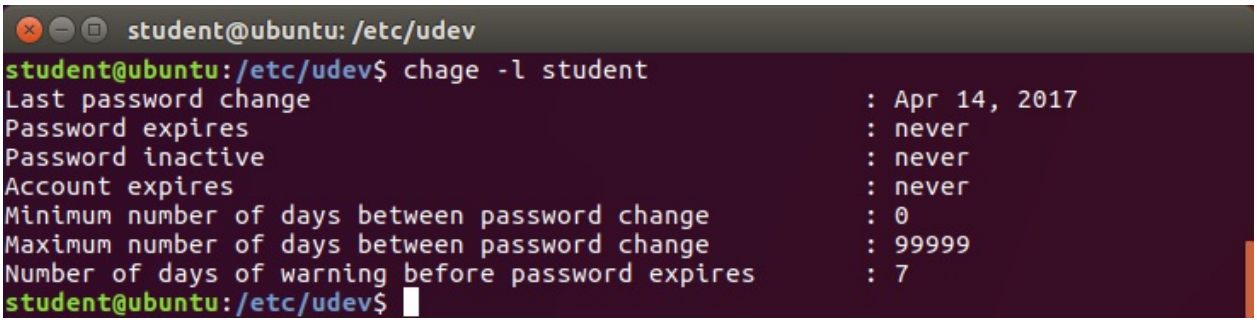
Examples:

```
$ sudo chage -l dexter
$ sudo chage -m 14 -M 30 kevlin
$ sudo chage -E 2012-4-1 morgan
$ sudo chage -d 0 clyde
```

Only root user can use **chage**. One exception: any user can run **chage -l** to see their aging, as shown below.

To force user to change password at next login:

```
$ sudo chage -d 0 Username
```



30.14 Restricted shell

Under Linux, one can use **restricted shell**, invoked as:

```
$ bash -r
```

(Some distributions may define an **rbash** command to same effect.)

Restricted shell: functions in more tightly controlled environment than standard shell, but otherwise functions normally. In particular:

- Prevents user from using **cd** to change directories
- Prevents user from redefining following environment variables: `SHELL`, `ENV`, `PATH`
- Does not permit user to specify absolute path or executable command names starting from `/`
- Prevents user from redirecting input and/or output

Note: there are other restrictions; best way to see them all is to do **man bash** and search for `RESTRICTED SHELL`.

Because restricted shell executes `$HOME/.bash_profile` without restriction, user must have neither write nor execute permission on `/home` directory.

Restricted accounts can also be enabled by creating symlink to `/bin/bash`, named `/bin/rbash`, and using in `/etc/passwd`, as will discuss next.

30.15 Restricted Accounts

There are times when granting access to user necessary, but should be limited in scope. Setting up restricted user account can be useful in this context. A restricted account:

- Uses the restricted shell

- Limits available system programs and user applications
- Limits system resources
- Limits access times
- Limits access locations

From command line, or from script, restricted shell may be invoked with `/bin/bash -r`. However, flags may not be specified in `/etc/passwd` file. Simple way to get around this restriction would be to do one of the following:

```
$ cd /bin ; sudo ln -s bash rbash
$ cd /bin ; sudo ln bash rbash
$ cd /bin ; sudo cp bash rbash
```

and then, use `/bin/bash` as shell in `/etc/passwd`.

When setting up such account, should avoid inadvertently adding system directories to `PATH` environment variable; this would grant restricted user ability to execute other system programs, such as unrestricted shell.

Restricted accounts also sometimes referred to as **limited accounts**.

30.16 The root Account

Root account should only be used for administrative purposes when absolutely necessary, never used as regular account. Mistakes very costly, both for integrity and stability, and for system security.

Default: root logins through network generally prohibited for security reasons. Can permit **Secure Shell** logins using **ssh**, configured with `/etc/ssh/sshd_config`, and **PAM** (**P**luggable **A**uthentication **M**odules), discussed later, through `pam_securetty.so` module and associated `/etc/securetty` file. Root login permitted only from devices listed in `/etc/securetty`.

Generally recommended that all root access be through **su** or **sudo** (causing audit trail of all root access through **sudo**). Note: some distributions (such as Ubuntu), by default actually prohibit logging in directly to the root account.

PAM can also be used to restrict which users allowed to **su** to root. Might also be worth it to configure **auditd** to log all commands executed as root.

30.17 SSH

One often needs to login through network into remote system, either with same user name or another. Or, one needs to transfer files to/from remote machine. In either case, one wants to do this securely, free from interception.

SSH (**S**ecure **SH**ell) exists for this purpose. Uses encryption based on strong algorithms. Assuming proper **ssh** packages installed on system, one needs no further setup to begin using **ssh**.

To sign onto remote system:

```
$ whoami
student
$ ssh farflung.com
student@farflung.com\'s password: (type here)
$
```

where assuming there is a student account on farflung.com. To log in as different user:

```
$ ssh root@farflung.com
```

```
root@farflung.com\'s password: (type here)
```

or

```
$ ssh -l root farflung.com
root@farflung.com\'s password: (type here)
```

To copy files from one system to another:

```
$ scp file.txt farflung.com:/tmp
$ scp file.tex student@farflung.com/home/student
$ scp -r some_dir farflung.com:/tmp/some_dir
```

(Omitted request for password to save space; if configured properly with encryption keys as discussed next, will not need to supply password.)

To run command on multiple machines simultaneously:

```
$ for machines in node1 node2 node3
do
      (ssh $ machines some_command &)
done
```

30.18 SSH Configuration Files

Can configure **SSH** to further expedite use, in particular to permit logging in without password. User-specific configuration files created under every user's home directory in hidden `.ssh` directory:

```
$ ls -l ~/.ssh
total 20
-rw-r--r-- 1 hilda hilda 1172 Sep 27  2014 authorized_keys
-rw------- 1 hilda hilda  207 Aug  9  2011 config
-rw------- 1 hilda hilda 1675 Dec  8  2010 id_rsa
-rw-r--r-- 1 hilda hilda  393 Dec  8  2010 id_rsa.pub
-rw-r--r-- 1 hilda hilda 1980 Apr 28 07:36 known_hosts
```

which contains:

- **`id_rsa`** : the user's private encryption key
- **`id_rsa.pub`** : the user's public encryption key
- **`authorized_keys`** : a list of public keys that are permitted to login
- **`known_hosts`** : a list of hosts from which logins have been allowed in the past
- **`config`** : a configuration file for specifying various options

First, user has to generate private and public encryption keys with **ssh-keygen**:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hilda/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hilda/.ssh/id_rsa
Your public key has been saved in /home/hilda/.ssh/id_rsa.pub
```

```
The key fingerprint is:
76:da:d3:51:1e:c8:2d:3b:34:28:46:b2:2b:db:d1:c4 hilda@c7
The key\'s randomart image is:
+--[ RSA 2048]----+
|      . .        |
|       =   o o   |
|      . E . * +  |
|       = . . * . |
|    . o S . + .  |
|     + o + . o   |
|    . . . o .    |
|           .     |
|                 |
+-----------------+
```

This will also generate the public key, `/.ssh/id_rsa.pub`.

Private key must never ever be shared with anyone!

Public key can be given to any machine with which you want to permit password-less access. Should also be added to your `authorized_keys` files, together with all public keys from other users who have accounts on you machine and you want tot permit password-less access to their accounts.

`known_hosts` file gradually built up as **ssh** accesses occur. If system detects changes in users who are trying to log in through **ssh**, will warn you of them and afford opportunity to deny access. Note: `authorized_keys` file contains information about users and machines:

```
$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQ
...0000aSd...hilda@sbc
```

while `known_hosts` only contains information about computer nodes:

```
$ cat known_hosts
192.30.252.129 ssh_rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAq2A7hRGmdnm9tUDb09IDSw
....BK6tb...==
```

Can examine **man ssh_config** page to see what kinds of options can go into **ssh** configuration files.

30.19 Remote Graphical Login

Login into remote machine with full graphical desktop. Often, may use VNC (Virtual Network Computing) to connect to system. Common implementation: **`tigervnc`**.

To test, first make sure that **vnc** packages installed:

```
$ sudo yum install tigervnc tigervnc-server
$ sudo zypper install tigervnc tigervnc-server
$ sudo apt install tigervnc tigervnc-server
```

using right package management system command.

Start the server as a normal user with:

```
$ vncserver
```

Can test with:

```
$ vncviewer localhost:2
```

(May have to play with numbers other than 2, such as 1, 3, 4…, depending on what you are running at the moment, and how your machine is configured.)

To view from remote machine, just slightly different:

```
$ vncviewer -via student@some_machine localhost:2
```

If you get a rather strange message about having to authenticate because of 'color profile', and no passwords work, have to kill **colord** daemon on server machine:

```
$ sudo systemctl stop colord
```

This is a bug (not a feature), will only appear in some distributions and some systems for unclear reasons.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 31 Title - Notes

31.2 Introduction

Linux systems form collections of users called **groups**, whose members share common purposes. To further that end, they share certain files/directories and maintain some common privileges. This separates them from other on system, sometimes collectively called **world**. Using groups aids collaborative projects enormously.

31.3 Learning Objectives:

- Explain why it is useful to have Linux users belong to one or more groups.
- Use utlilities such as **groupadd**, **groupdel**, **groupmod**, and **usermod** to create, remove, and manipulate groups and their membership.
- Describe User Private Groups.
- Explain the concept of group membership.

31.4 Groups

Groups: collections of users whose members share some common purpose within Linux systems. Share certain files/directories, maintain some common privileges. Separates them from other on the system, sometimes collectively called the world. Using groups aids collaborative projects enormously. Users belong to one or more groups.

Groups defined in `/etc/group`, which has same role for groups as `/etc/passwd` has for users. Each line of the file looks like:

```
groupname:password:GID:user1, user2,...
```

where:

- `groupname` is the name of the group
- `password` is the password place holder. Group passwords may be set, but only if the `/etc/gshadow` file exists
- `GID` is the group identifier. Values between 0 and 99 are for system groups. Values between 100 and `GID_MIN` (as defined in `/etc/login.defs` and usually the same as `UID_MIN`) are considered special. Values over `GID_MIN` are for **UPG** (**U**ser **P**rivate **G**roups)
- `user1, user2,...` is a comma-separated list of users who are members of the group. The user need not be listed here is this groups is the user's principal group

31.5 Group Management

Group accounts may be managed/maintained with:

- **groupadd**: Add a new group
- **groupdel**: Remove a group
- **groupmod**: Modify a group's properties
- **usermod**: Modify a user's group memberships (add or remove).

Can also edit `/etc/group` directly, but better to use **vigr** utility, which is generally symbolically linked to **vipw** utility mentioned earlier.

These group manipulation utilities modify `/etc/group` and (if it exists) `/etc/gshadow`, and may only be executed by root:

Examples:

```
$ sudo groupadd -r -g 215 staff
$ sudo groupmod -g 101 blah
$ sudo groupdel newgroup
$ sudo usermod -G student,group1,group2 student
```

Note: Be very careful with **usermod -G** command: the group list that follows is the complete list of groups, not just the changes. Any supplementail groups left out will be gone! Non-destructive use should utilize `-a` option, which will preserve pre-existing group memberships when adding new ones.

31.6 User Private Groups

Linux uses User Private Groups (UPG).

Idea behind UPGs: each user will have his/her own group. However, UPGs are **not guaranteed** to be private. Additional members may be added to someone's private group in `/etc/group`.

By default, users whose accounts are created with **useradd** have: primary `GID` = `UID` and group name is also identical to user name.

As specified in `/etc/profile`, **umask** is set to `002` for all users created with UPG. Under this scheme, user files thus created with permissions `664` (`rw-rw-r--`) and directories with `775` (`rwxrwxr-x`). Will discuss **umask** in next section.

31.7 Group Membership

A Linux user has one **primary** group; listed in `/etc/passwd` and will also be listed in `/etc/group`. User may belong to between 0 and 15 **secondary groups**.

Primary group is the `GID` that is used whenever the user creates files or directories. Membership in other, secondary, groups grant user additional permissions.

Group membership can be identified by running either of the following commands:

```
$ groups [user1 user2 ...]
$ id -Gn [user1 user2 ...]
```

With no arguments, either command reports on current user. Note: default groups can differ by distribution:

On CentOS:

```
[student@CentOS7 ~]$ groups
student
[student@CentOS7 ~]$
```

On Ubuntu:

```
student@ubuntu:~$ groups
student adm cdrom sudo dip plugdev lpadmin sambashare libvirt
student@ubuntu:~$
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 32 File Permissions and Ownership - Notes

32.3 Learning Objectives:

- Explain the concepts of **owner**, **group**, and **world**.
- Set file access rights (read, write, and execute) for each category.
- **Authenticate** requests for file access, respecting proper permissions.
- Use **chmod** to change file permissions, **chown** to change user ownership, and **chgrp** to change group ownership.
- Understand the role of **umask** in establishing desired permissions on newly created files.
- Use ACLs to extend the simpler user, group, world and read, write, execute model.

32.4 Owner, Group, and World

When you do an `ls -l` as in:

```
$ ls -l a_file
-rw-rw-r-- 1 coop aproject 1601 Mar 9 15:03  a_file
```

after the first character (which indicates the type of the file object), there are nine more which indicate the access rights granted to potential file users. There are arranged in three groups of three:

- **owner**: the user who owns the file (also called **user**)
- **group**: the group of users who have access
- **world**: the rest of the world (also called **other**)

In above listing, user is `coop` and group is `aproject`.

32.5 File Access Rights

If you do a long listing of a file, as in:

```
$ ls -l /usr/bin/vi
-rwxr-xr-x. 1 root root 910200 Jan 30 2014 /usr/bin/vi
```

each of the triplets can have each of the following sets:

- **r**: read access is allowed
- **w**: write access is allowed
- **x**: execute access is allowed

If permission not allowed, a `-` appears instead of one of these characters.

In addition, other specialized permissions exist for each category, such as the **setuid/setgid** permissions.

This, in preceding example, user `coop` and members of the group `aproject` have read and write access, while anyone else has only read access.

32.6 File Permissions and Security and Authentication

File access permissions are critical part of Linux security system. Any request to access file requires comparison of credentials and identity of requesting user to those of the owner of the file.

This **authentication** is granted depending on one of these three sets of permissions, in following order:

1. If the requester is the file owner, the file owner permissions are used
2. Otherwise, is the requester is in the group that owns the files, the group permissions are examined
3. If that doesn't succeed, the world permissions are examined

32.7 Changing Permissions: chmod

Changing file permissions is done with **chmod**. You can only change permissions on files you own, unless you are the superuser.

There are a number of different ways to use **chmod**. For instance, to give owner and world execute permission, and remove group write permission:

```
$ ls -l a_file
-rw-rw-r-- 1 coop coop 1601 Mar 9 15:04  a_file
$ chmod uo+x,g-w a_file
$ ls -l a_file
-rwxr--r-x 1 coop coop 1601 Mar 9 15:04  a_file
```

where `u` stands for user (owner), `o` stands for other (world), and `g` stands for group.

Permissions can be represented either as a bitmap, usually written in octal, or in a symbolic form. Octal bitmaps usually look like `0755`, which symbolic representations look like `u+rwx,g+rwx,o+rx`.

32.8 Octal Digits

Symbolic syntax can be difficult to type and remember, so one often uses octal shorthand, which lets you set all the permissions in one step. Done with simple algorithm, and a single digit suffices to specify all three permission bits for each entity. Octal number representation if **sum** for each digit of:

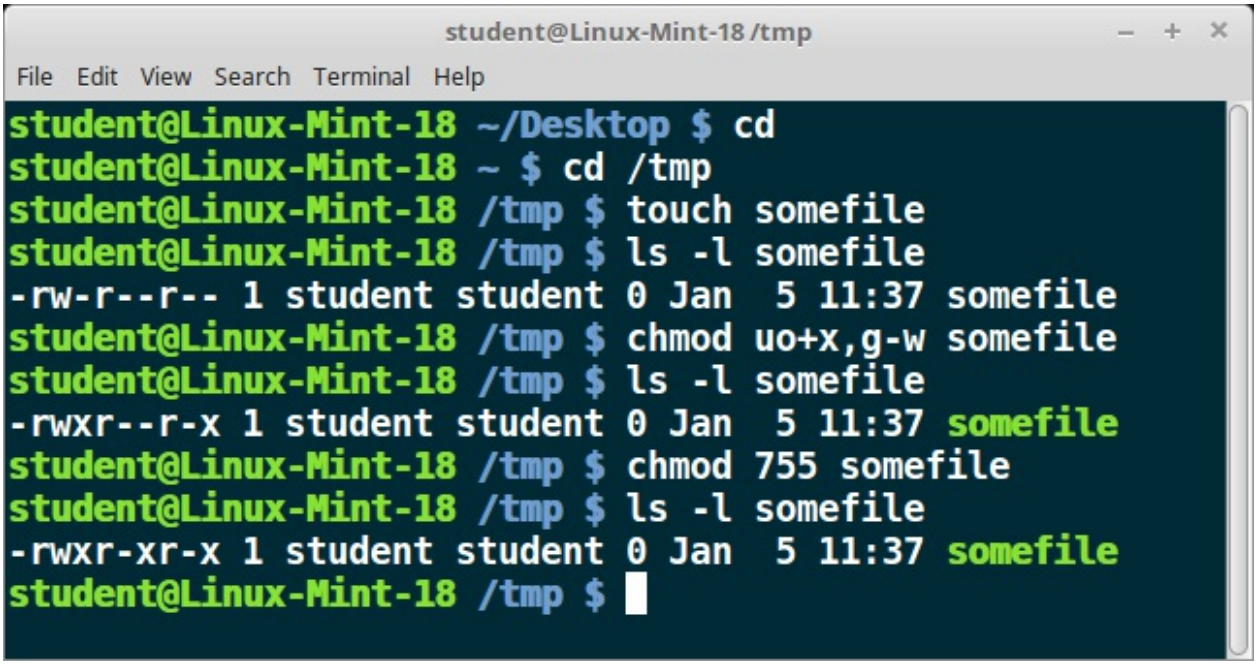
- **4** if read permission desired
- **2** if write permission desired
- **1** if execute permission desired

Thus, **7** means read/write/execute, **6** means read/write, **5** means read/execute.

When you apply this with **chmod**, have to give a value for each of the three digits:

```
$ chmod 755 a_file
$ ls -l a_file
-rwxr-xr-x 1 coop coop 1602 Mar 9 15:04  a_file
```

Permissions can be represented either as a bitmap, usually written in octal, or in a symbolic form. Octal bitmaps usually look like `0755`, which symbolic representations look like `u+rwx,g+rwx,o+rx`.



32.9 chown and chgrp

Changing file ownership is done with **chown** and changing the group is done with **chgrp**.

Only the superuser can change ownership on files. Likewise, can only change group ownership to groups that you are a member of.

Changing group ownership of file:

```
$ chgrp cleavers somefile
```

and changing ownership (only superuser can do this):

```
$ chown wally somefile
```

Can change both at same time with:

```
$ chown wally:cleavers somefile
```

where you separate owner and group with colon (or period).

All three of these programs take `-R` option, which stands for recursive. For example:

```
$ chown -R wally:cleavers ./
$ chown -r wally:wally subdir
```

will change owner and group of all files in current directory and all its subdirectories in first command, and in `subdir` and all its subdirectories in second command.

32.10 umask

Default permissions given when creating file are read/write for owner, group **and** world (`0666`), and for a directory it is read/write/execute for everyone (`0777`). However, if you do the following:

```
$ touch afile
$ mkdir adir
$ ls -l | grep -e afile -e adir
drwxrwxr-x 2 coop coop 4096 Sep 16 11:18 adir
-rw-rw-r-- 1 coop coop    0 Sep 16 11:17 afile
```

you will notice actual permissions have changed to `664` for the file and `775` for the directory. They have been modified by current **umask** whose purpose is to show which permissions should be denied. The current value can be shown by:

```
$ umask
0002
```

which is the most conventional value set by system administrators for users. This value if combined with the file creation permissions to get the actual result; i.e.,

```
0666 & ~002 = 0664; i.e., rw-rw-r--
```

Can change **umask** at any time with **umask** command:

```
$ umask 0022
```

32.12 Filesystem ACLs

Linux contains full implementation of POSIX ACLs (**A**ccess **C**ontrol **L**ists) which extends simpler user, group, world and read, write, execute model.

Particular privileges can be granted to specific users or groups of users when accessing certain objects or classes of objects. Files and directories can be shared without using `777` permissions.

While Linux kernel enables use of ACLs, still must be implemented as well in particular filesystem. All major filesystems used in modern Linux distributions incorporate the ACL extensions, and one can use the option `-acl` when mounting. Default set of ACLs is created at system install.

32.13 Getting and Setting ACLs

To see ACLs:

```
$ getfacl file|directory
```

Example:

```
$ getfacl file1
```

To set ACLs:

```
$ setfacl options permissions file|directory
```

Examples:

```
$ setfacl -m u:isabelle:rx /home/stephane/file1
$ setfacl -x u:isabelle    /home/stephane/file
```

Note: new files inherit default ACL (if set) from the directory they reside in. Also note: **mv** and **cp -p** preserves ACLs.

To remove an ACL:

```
$ setfacl -x u:isabelle /home/stephane/file1
```

To set default on directory:

```
$ setfacl -m d:u:isabelle:rx somedir
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 33 Pluggable Authentication Modules (PAM) - Notes

33.2 Introduction

Pluggable Authentication Modules provide uniform mechanism to ensure that users and applications properly identified and authenticated. Conditional rules can be applied to limit scope of permissions and control can be established over what to do in case of either success or failure. PAM can also work with LDAP to centralize authentication throughout network.

33.3 Learning Objectives:

- Explain the basic concepts that motivate the user of PAM.
- List the steps involved in the authentication process.
- Use and modify PAM configuration files.
- Know how to interpret PAM rules and create new ones.
- Apply LDAP to use and administer distributed directory services over the network.

33.4 PAM: A Unified Approach to Authentication

Historically, authentication of users performed individually by individual applications; i.e., **su**, **login**, **ssh** would authenticate and establish user accounts independently of each other.

Most modern Linux applications have been written/rewritten to exploit PAM (**P**luggable **A**uthentication **M**odules) so that authentication can be done in one uniform way, using **libpam**.

This library of modules provides enormous flexibility and consistency with respect to authentication, password, session, account services.

PAM incorporates following components:

- PAM-aware applications
- Configuration files in `/etc/pam.d`
- PAM modules in the `libpam*` libraries, which can be found in different locations depending on the Linux distribution

Each PAM-aware application/service may be configured with respect to PAM by individual configuration file in `/etc/pam.d`.

33.5 Authentication Process

Several steps involved in authentication:

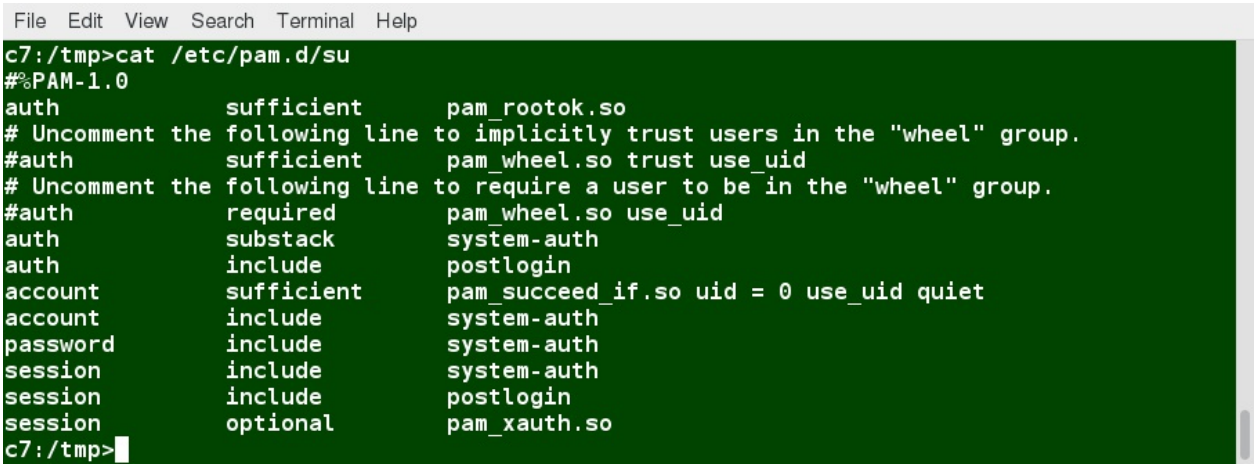
- User invokes PAM-aware application, such as **login**, **su**, **ssh**
- Application calls **libpam**
- Library checks for files in `/etc/pam.d`; these delineate which PAM modules to invoke, including **system-auth**
- Each referenced module executed in accordance with rules of relevant configuration file for that application

33.6 PAM Configuration Files

Each file in `/etc/pam.d` corresponds to a **service** and each (non-commented) line in the file specifies a rule. The rule is formatted as list of space-separated tokens, the first two of which are case insensitive:

```
type control module-path module-arguments
```

Example: screenshot here shows contents of `/etc/pam.d/su` on RHEL 7 system. Notice that there is a stack; **su** will require loading of **system-auth** etc.



33.7 PAM Rules

Module `type` specifies **management group** that module is to be associated with:

- `auth`: Instructs application to prompt user for identification (username, password, etc). May set credentials and grant privileges
- `account`: Checks on aspects of user's account, such as password aging, access control, etc.
- `password`: Responsible for updating user authentication token, usually a password
- `session`: Used to provide functions before/after session is established (such as setting up environment, logging, etc)

The `control` flag controls how the success/failure of module affects overall authentication process:

- `required`: Must return success for the service to be granted. If part of stack, all other modules are still executed. Application not told which module/modules failed
- `requisite`: Same as `required`, except failure in any module terminates stack and return status sent to application
- `optional`: Module not required. If the only module, then return status to application may cause failure
- `sufficient`: If this module succeeds, then no subsequent modules in stack executed. If it fails, then it doesn't necessarily cause the stack to fail, unless it is the only one in the stack

There are other `control` flags, such as `include`, `substack`. See **man pam.d** for details.

`module-path` gives file name of library to be found in `/lib*/security`, in either absolute or relative path form.

`module-arguments` can be given to modify **PAM** module's behavior.

33.8 LDAP Authentication

LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol): industry standard protocol for using/administering distributed directory services over network, meant to be both open and vendor-neutral.

When using LDAP for centralized authentication, each system/client connects to centralized LDAP server for user authentication.

Using TLS makes it a secure option and is recommended.

LDAP uses PAM and **system-config-authentication** or **authconfig-tui**. One has to specify the server, search base DN (domain name), and TLS (**T**ransport **L**ayer **S**ecurity). Also required: **openldap-clients**, **pam ldap**, **nss-pam-ldapd**.

When configuring system for LDAP authentication, five files changed:

- `/etc/openldap/ldap.conf`
- `/etc/pam_ldap.conf`
- `/etc/nslcd.conf`
- `/etc/sssd/sssd.conf`
- `/etc/nsswitch.conf`

Can edit these files manually or use one of the utility programs available (**system-config-authentication** or **authconfig-tui**).

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 34 Network Addresses - Notes

34.3 Learning Objectives:

- Differentiate between different types of IPv4 and IPv6 addresses.
- Understand the role of netmasks.
- Get, set, and change the hostname, based on the system you are using.

34.4 IP Addresses

IP addresses used globally to uniquely identify nodes across internet. Registered through **ISP**s (**I**nternet **S**ervice **P**roviders).

IP address is the number that identifies your system on the network. Comes in two varieties:

- **IPv4**: A 32-bit address, composed of 4 **octets** (an octet is just 8 bits, or a byte)

  Example: `148.114.252.10`

- **IPv6**: A 128-bit address, composed of 8 16-bit octet pairs.

  Example: `2003:0db5:6123:0000:1f4f:0000:5529:fe23`

In either case, set of **reserved** addresses also included. Will focus somewhat more on IPv4, as it is still what is most commonly used.

34.5 IPv4 Address Types

IPv4 address types include:

- **Unicast**: an address associated with specific host. Might be something like `140.211.169.4` or `64.254.248.193`
- **Network**: an address whose **host** portion is set to all binary zeroes. Ex. `192.168.1.0` (the host portion can be the last 1-3 octets as discussed later; here it is just the last octet)
- **Broadcast**: an address to which each member of a particular network will listen. Will have the host portion set to all 1 bits, such as in `172.16.255.255` or `148.114.255.255` or `192.168.1.255` (Host portion is last two octets in the first two cases, just the last one in the third case)
- **Multicast**: an address to which approximately configured nodes will listen. The address `224.0.0.2` is an example of a multicast address. Only nodes specifically configured to pay attention to specific multicast address will interpret packets for that multicast group

34.6 Reserved Addresses

Certain addresses and address ranges are reserved for special purposes:

- `127.x.x.x`: reserved for loopback (local system) interface, where `0 <= x <= 254`. Generally, `127.0.0.1`

- `0.0.0.0`: used by systems that do not yet known their own address. Protocols like DHCP and BOOTP use this address when attempting to communicate with a server

- `255.255.255.255` : generic broadcast private address, reserve for internal use. These addresses are never assigned or registered to anyone. Generally not routable
- Other examples of reserved address ranges include:

  `10.0.0.0` -- `10.255.255.255`

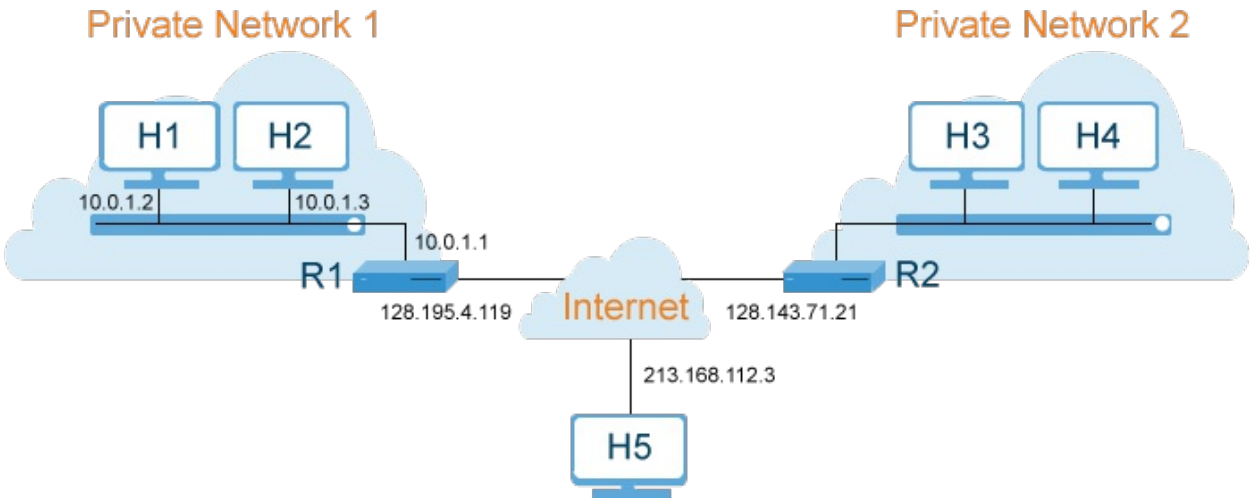
  `172.16.0.0` -- `172.31.255.255`

  `192.168.0.0` -- `192.168.255.255`

  etc.

  Each of these has purpose. For example, the familiar address range `192.168.x.x` is used only for local communications w ithin a private netw ork.

Can see long list of reserved addresses for both IPv4 and IPv6 on the Reserved IP addresses Wikipedia page.



Private vs. Public IP addresses

34.7 IPv6 Address Types

IPv6 address types include:

- **Unicast**: packet delivered to one interface
  - **Link-local**: auto-configured to for every interface to have one. Non-routable
  - **Global**: dynamically or manualle assigned. routable
  - Reserved for documentation
- **Multicast**: a packet is delivered to multiple interfaces
- **Anycast**: a packet is delivered to the nearest of multiple interfaces (in terms of routing distance)
- **IPv4-mapped**: an IPv4 address mapped to IPv4. For example, `::FFFF:a.b.c.d/96`

In addition, IPv6 has some special types of addresses such as loopback, w hich is assigned to the `lo` interface, as `::1/128`.

34.8 IPv4 Address Classes

Historically, IP addresses based on defined **classes**. Classes A, B, C used to distinguish netw ork portion of address from host portion of address. This is used for routing purposes.

Address Classes

Network Class	Highest order octet range	Notes
A	1-127	128 networks, 16,772,214 hosts per network, 127.x.x.x reserved for loopback
B	128-191	16,384 networks, 65,534 hosts per network
C	192-223	2,097,152 networks, 254 hosts per network
D	224-239	Multicast addresses
E	240-254	Reserved address range

34.9 Netmasks

Netmask used to determine how much of address used for network portion and how much for host portion, as seen. Also used to determine network/broadcast addresses.

Address Classes and Netmasks

Network Class	Decimal	Hex	Binary
A	`255.0.0.0`	`ff:00:00:00`	`11111111 00000000 00000000 00000000`
B	`255.255.0.0`	`ff:ff:00:00`	`11111111 11111111 00000000 00000000`
C	`255.255.255.0`	`ff:ff:ff:00`	`11111111 11111111 11111111 00000000`

Class A addresses use 8 bits for network portion of address and 24 bits for host portion of address.

Class B addresses use 16 for network, 16 for host.

Class C addresses use 24 for network, 8 for host.

Class D addresses used for multicasting.

Class E addresses currently not used.

Network address obtained by **anding** (logical and - &) IP address with netmask. Interested in network addresses because they define local network which consists of collection of nodes connected via same media and sharing same network address. All nodes on same network can directly see each other.

Example:

```
172.16.2.17  ip address
&255.255.0.0  netmask
-----------------
172.16.0.0  network address
```

34.10 Hostname

Hostname: simply a label used to identify networked device to distinguish from other elements on network. Historically, also been called nodename.

For DNS purposes, hostnames appended with period (dot) and domain name, so that machine with hostname of `antje` could have **fully qualified domain name (FQDN)** of `antje.linuxfoundation.org`.

Hostname generally specified at installation time, can be modified at any time later.

34.11 Getting and Setting a Hostname

At any given time, ascertaining hostname as simple as:

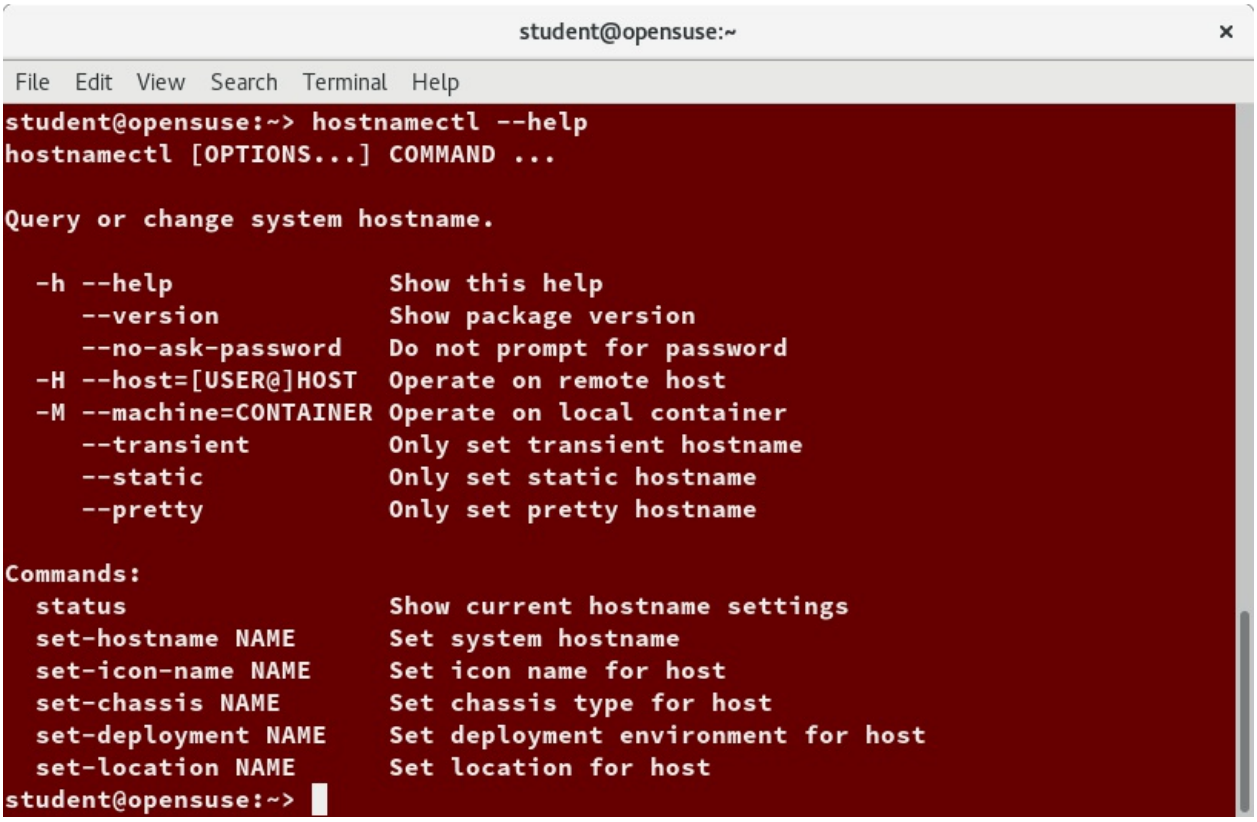
```
$ hostname
wally
```

Changing hostname involved giving parameter, requires root privilege:

```
$ sudo hostname lumpy
lumpy
```

Current value always stored in `/etc/hostname` on most Linux distributions.

Changing hostname in this fashion -> not persistent; when system rebooted, reverts to value before modification. As usual, making persistent changes involves changing configuration files in `/etc` directory tree. Best done by using **hostnamectl** facility, which arises from **systemd** infrastructure.



Changing hostname persistently (surviving reboot):

```
$ sudo hostnamectl set-hostname MYPC
```

Most distributions do not use "pretty" hostname for anything.

On almost all Linux systems, one can simply edit (as root) the file `/etc/hostname` and put in new name.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 35 Network Devices and Configuration - Notes

35.2 Introduction

Network devices such as Ethernet, wireless connections require careful configuration, especially when there are multiple devices of same type. Consistent, persistent device naming can become tricky in such circumstances. Recent adoption of new schemes -> naming more predictable. Number of important utilities used to bring devices up/down, configure properties, establish routes etc. System administrators must become adept at their use.

35.3 Learning Objectives:

- Identify network devices and understand how the operating system names them and binds them to specific duties.
- Use the **ip** utility to display and control devices, routing, policy-based routing, and tunneling.
- Use the older **ifconfig** to configure, control, and query network interface parameters from either the command line or from system configuration scripts.
- Understand the Predictable Network Interface Device Names scheme.
- Know the main network configuration files in `/etc`.
- Use **NetworkManager** (**nmtui** and **nmcli**) to configure network interfaces in a distribution-independent manner.
- Know how to set default routes and static routes.
- Configure name resolution as well as run diagnostic utilities.

35.4 Network Devices

Network devices *not* associated with **special device files** (also known as **device nodes**), unlike block/character devices. Know by their names rather than having associated entries in `/dev` directory.

Names consist of type identifier followed by number:

- `eth0`, `eth1`, `eno1`, `eno2`, etc. for Ethernet devices
- `wlan0`, `wlan1`, `wlan2`, `wlp3s0`, `wlp3s2`, etc. for wireless devices
- `br0`, `br1`, `br2`, etc. for bridge interfaces
- `vmnet0`, `vmnet1`, `vmnet2`, etc. for virtual devices for communicating with virtual clients

Historically, multiple virtual devices could be associated with single physical devices.

35.5 ip

ip: the command line utility used to configure, control, query interface parameters and control devices, routing, etc. Preferred to the venerable **ifconfig** discussed next, since more versatile + more efficient because it uses **netlink** sockets rather than **ioctl** system calls.

ip can be used for wide variety of tasks. Can be used to configure, control, query devices and interface parameters. Also manipulate routing, policy-based routing, tunneling.

Basic syntax:

```
ip [ OPTIONS ] OBJECT [ COMMAND | help ]
ip [ -force ] -batch filename
```

where second form can read commands from designated file.

ip a multiplex utility. `OBJECT` argument describes what kind of action going to be performed. Possible `COMMANDS` depend on which `OBJECT` selected.

Some of the main values of `OBJECT`:

Main ip `object`s

`OBJECT`	Function
`address`	**IPv4** or **IPv6** protocol device address
`link`	Network Devices
`maddress`	Multicast Address
`monitor`	Watch for netlink messages
`route`	Routing table entry
`rule`	Rule in the routing policy database
`tunnel`	Tunnel over IP

36.6 Examples of Using ip

ip utility can be used in many ways:

- Show information for all network interfaces:

```
$ ip link show
```

- Show information for `eth0` network interface, including statistics:

```
$ ip -s link show eth0
```

- Set IP address for `eth0`:

```
$ sudo ip addr add 192.168.1.7 dev eth0
```

- Bring `eth0` down:

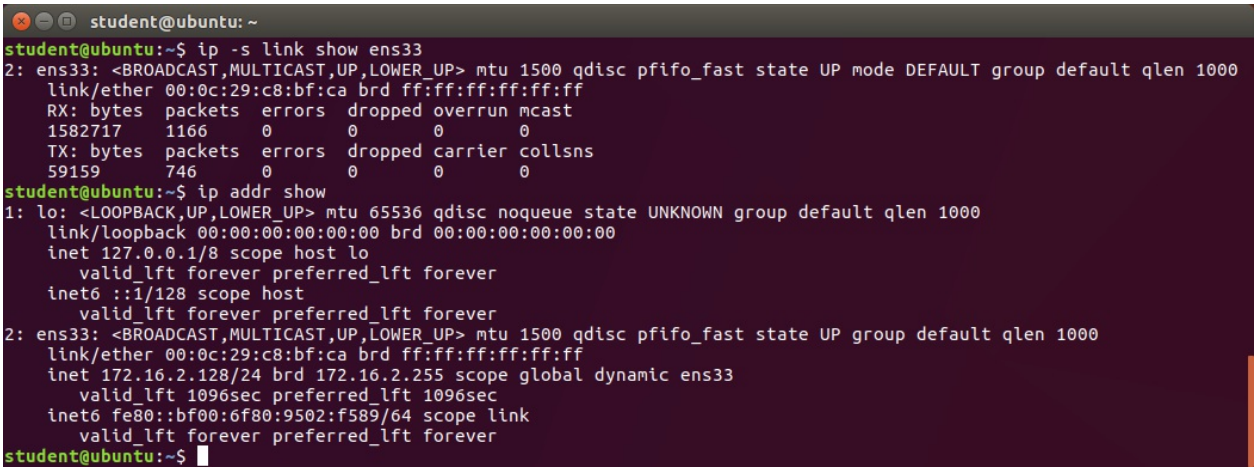
```
$ sudo ip link set eth0 down
```

- Set **MTU** to 1480 bytes for `eth0`:

```
$ sudo ip link set eth0 mtu 1480
```

- Set netw orking route:

```
$ sudo ip route add 172.16.1.0/24 via 192.168.1.5
```



35.7 ifconfig

ifconfig: system administration utility long found in UNIX-like operating systems, used to configure, control, query netw ork interface parameters from either command line or from system configuration scripts. Superseded by **ip**, some Linux distributions no longer install by default. Some usage examples:

- Display some information about all interfaces:

```
$ ifconfig
```

- Display information about only `eth0`:

```
$ ifconfig eth0
```

- Set IP address to `192.168.1.50` on interface `eth0`:

```
$ sudo ifconfig eth0 192.168.1.50
```

- Set the *netmask* to 24-bit:

```
$ sudo ifconfig eth0 netmask 255.255.255.0
```

- Bring interface `eth0` up:

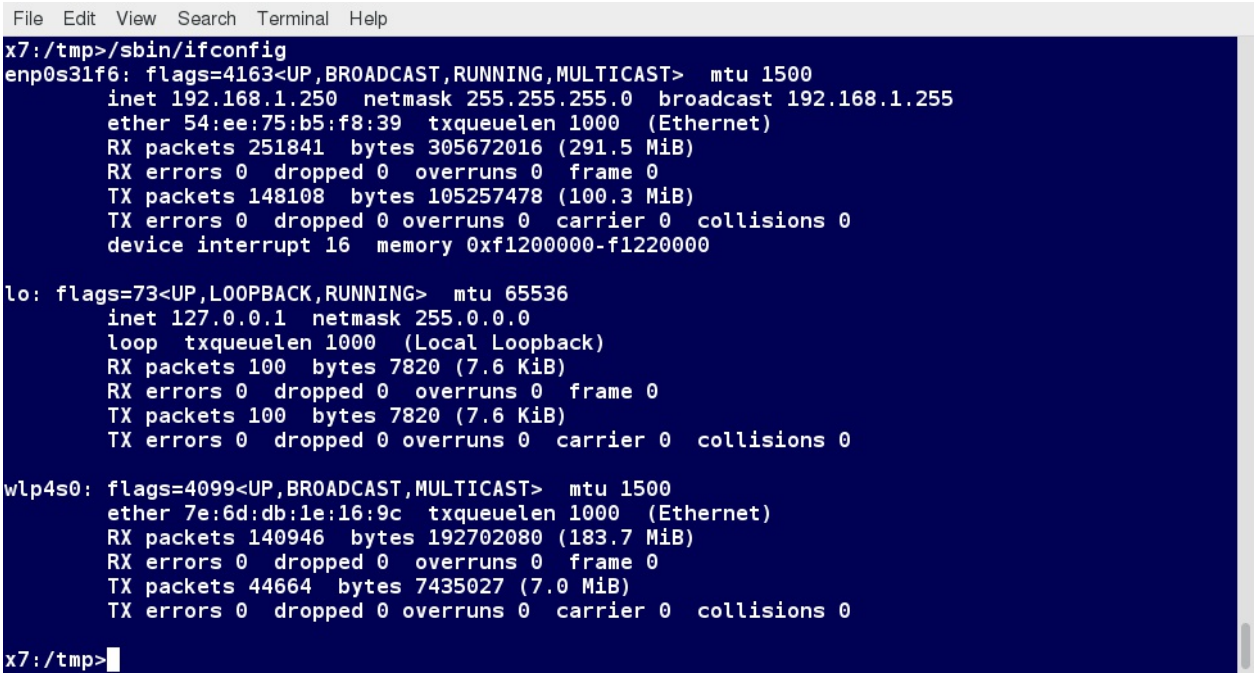
```
$ sudo ifconfig eth0 up
```

- Bring interface `eth0` down:

```
$ sudo ifconfig eth0 down
```

- Set the **MTU** (**M**aximum **T**ransfer **U**nit)

```
$ sudo ifconfig eth0 mtu 1480
```



35.8 Problems with Network Device Names

Classic device naming conventions described earlier encountered difficulties, especially when multiple interfaces of same type present. Eg., if two network cards, one named `eth0` and other `eth1`, but which physical device should be associated with each name?

Simplest method: have first device found be `eth0`, second `eth1` etc. Unfortunately, probing for devices not deterministic for modern systems, and devices may be located/plugged in unpredictable order. Thus, one might wind up with Internet interface swapped with local interface. Even if no change in hardware, order in which interfaces located known to vary with kernel version and configuration.

Many system administrators solved this problem in a simple manner: by hardcoding associations between hardware (MAC) addresses and device names in system configuration files and startup scripts. While this method worked for years, required manual tuning + had other problems, eg., when MAC addresses not fixed. Can happen in both embedded and virtualized systems.

35.9 Predictable Network Interface Device Names

Predictable Network Interface Device Names (**PNIDN**): strongly correlated with use of **udev** and integration with **systemd**. 5 types of names that devices can be given:

1. Incorporating Firmware or BIOS provided index numbers for on-board devices:

Example: `eno1`

2. Incorporating **Firmware** or **BIOS** provided **PCI Express** hotplug slow index numbers:

   Example: `ens1`

3. Incorporating physical and/or geographical location of the hardware connection:

   Example: `enp2s0`

4. Incorporating the `MAC` address:

   Example: `enx7837d1ea46da`

5. Using the old classic method:

   Example: `eth0`

35.10 Examples of the New Naming Scheme

For example, on machine with two onboard PCI network interfaces that would have been `eth0` and `eth1`:

```
$ ip link show | grep enp
2: enp4s2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo-fast state DOWN mode DEFAULT qlen 1000
3: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo-fast state UP mode DEFAULT qlen 1000
```

These names are correlated with physical locations of the hardware on the PCI system.

Likewise, for wireless device that previously would have been simply named `wlan0`:

```
17:/home/coop>ip link show | grep w1
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT qlen 1000
```

See same pattern. Easy to turn off new scheme, go back to classic names. Left as research project. In following, will mostly use classic names for definiteness + simplicity.

35.11 NIC Configuration Files

While network interfaces can be configured on the fly using either **ip** or **ifconfig** utilities, settings not persistent. Thus, number of Linux distribution-dependent files store persistent network interface and device configuration information.

Each distribution has own set of files and/or directories. Depending on version, might be:

Red Hat

```
/etc/sysconfig/network
/etc/sysconfig/network-scripts/ifcfg-ethX
/etc/sysconfig/network-scripts/ifcfg-ethX:Y
/etc/sysconfig/network-scripts/route-ethX
```

Debian

```
/etc/network/interfaces
```

SUSE

```
/etc/sysconfig/network
```

When using **systemd**, preferable to use Network Manager, rather than try to configure underlying text files. In fact, in new Linux distributions, many of these files non-existent, empty, or much smaller, there only for backward compatibility reasons.

35.12 Network Manager

Once upon a time, network connections almost all wired (Ethernet), did not change unless there was significant change to either the hardware, software, or network configuration. During system boot, files in `/etc` consulted to establish all device configurations.

However, modern systems often have dynamic configurations:

- Networks may change as a device is moved from place to place
- Wireless devices may have a large choice of networks to hook into
- Devices may change as hardware such as wireless devices are plugged in or turned on and off

Previously discussed configuration files created to deal with more static situations, very distribution dependent.

Network Manager still uses configuration files, but administrator can avoid directly manipulating them. Its use hopefully almost the same on different systems.

35.13 Network Manager Interfaces

GUI (Graphical User Interface)

If using your laptop in a hotel room or a coffee shop, probably going to use whatever graphical interface your Linux distribution's desktop offers. Can use this to select between different networks, configure security/passwords, turn devices off/on etc.

nmtui

If making configuration change on system that is likely to last for a while, likely to use **nmtui** as it has almost no learning curve + will edit underlying configuration files for you.

nmcli

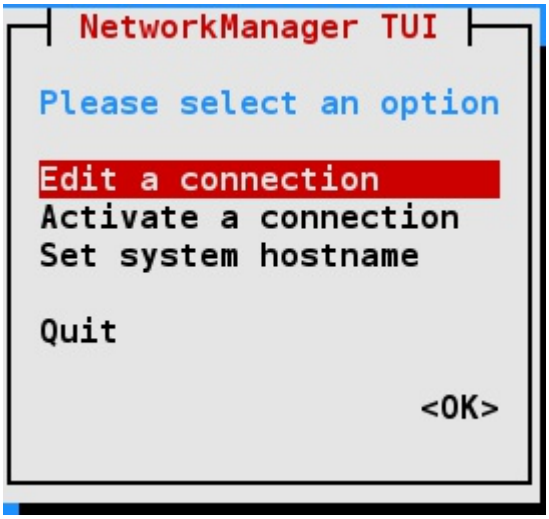
If need to run scripts that change network configuration, will want to use **nmcli**. Of, if command line junkie, may want to use this instead of **nmtui**.

If GUI properly done, should be able to accomplish any task using any of these three methods. However, will focus on **nmtui** and **nmcli** because essentially distribution independent + hide any differences in underlying configuration files.

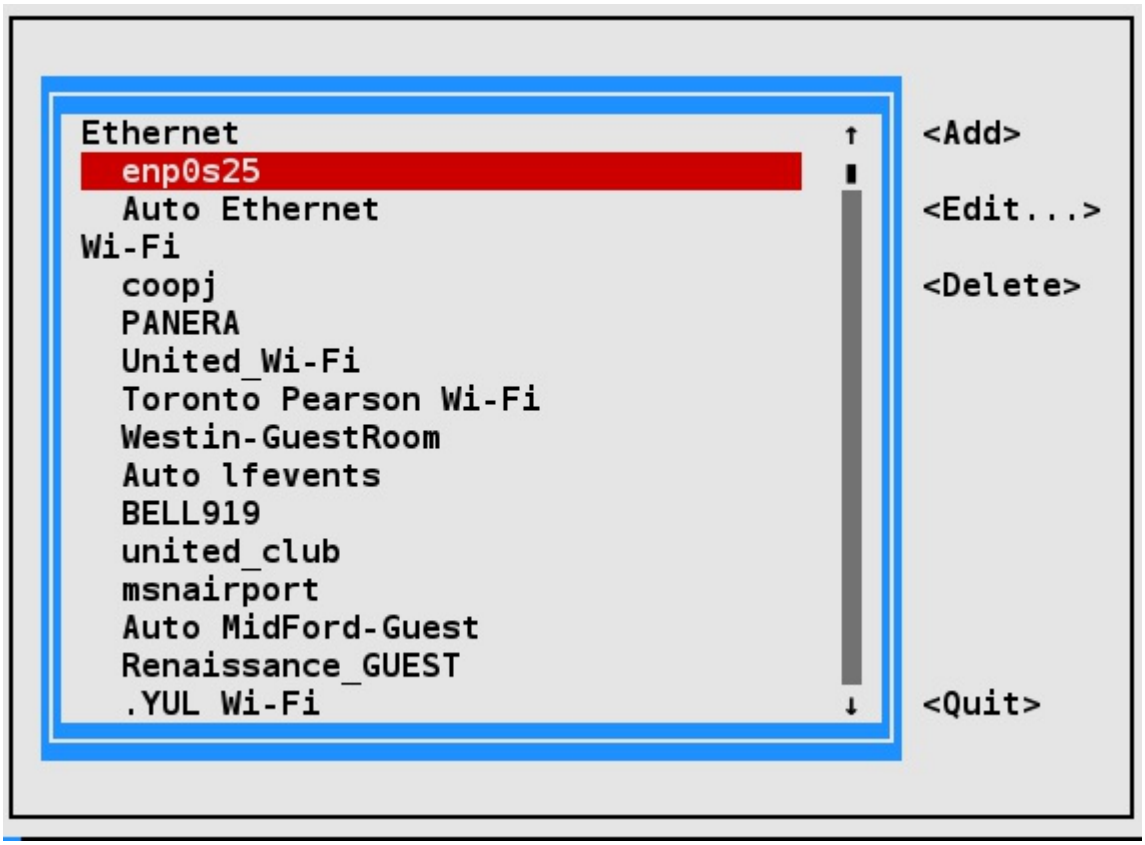
35.14 nmtui

nmtui straightforward to use. Can navigate with either arrow keys or tab key.

Besides activating/editing connections, also set system hostname. However, some operations, such as this, cannot be done by normal users, will be prompted for root password to go forward.

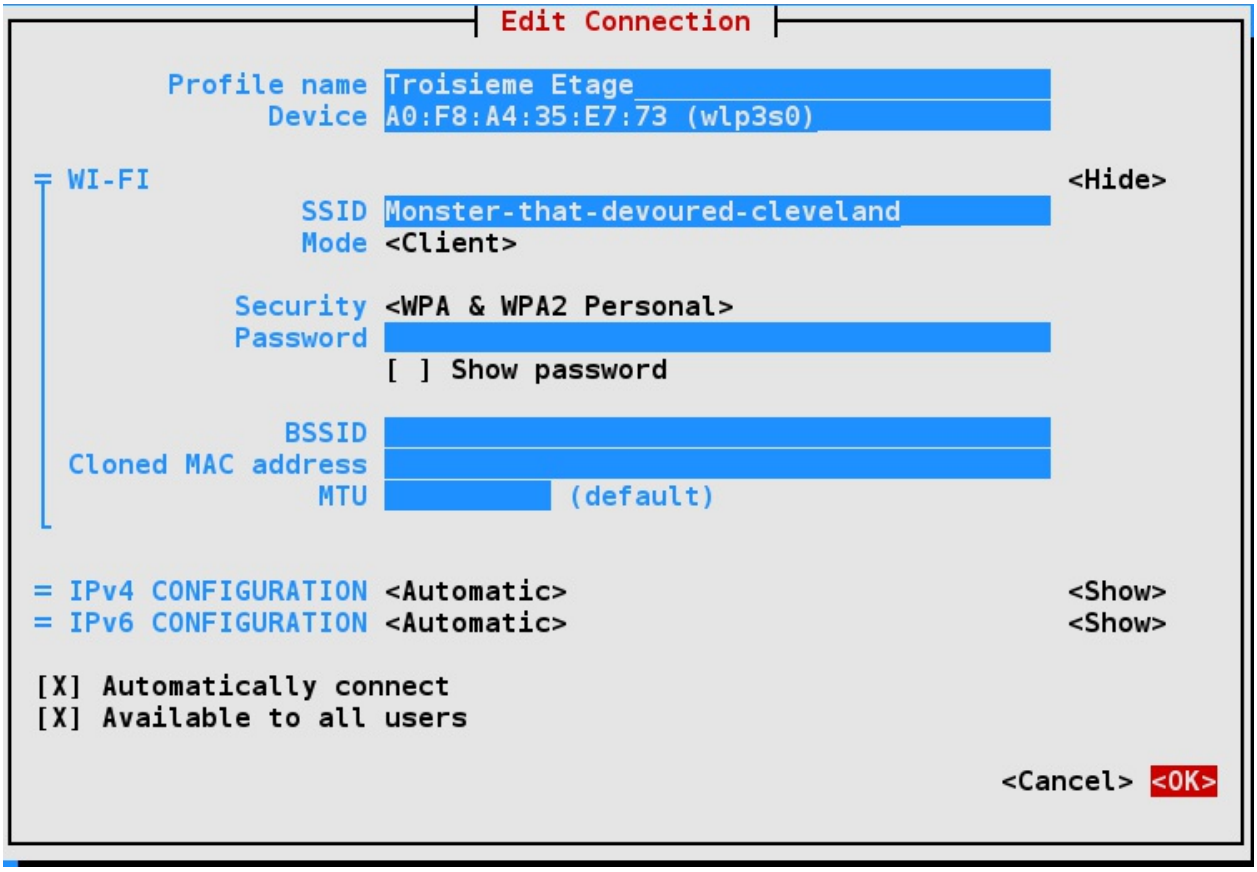


nmtui Main Screen



nmtui Edit Screen

35.15 nmtui Wireless Configuration



nmtui Wireless Configuration

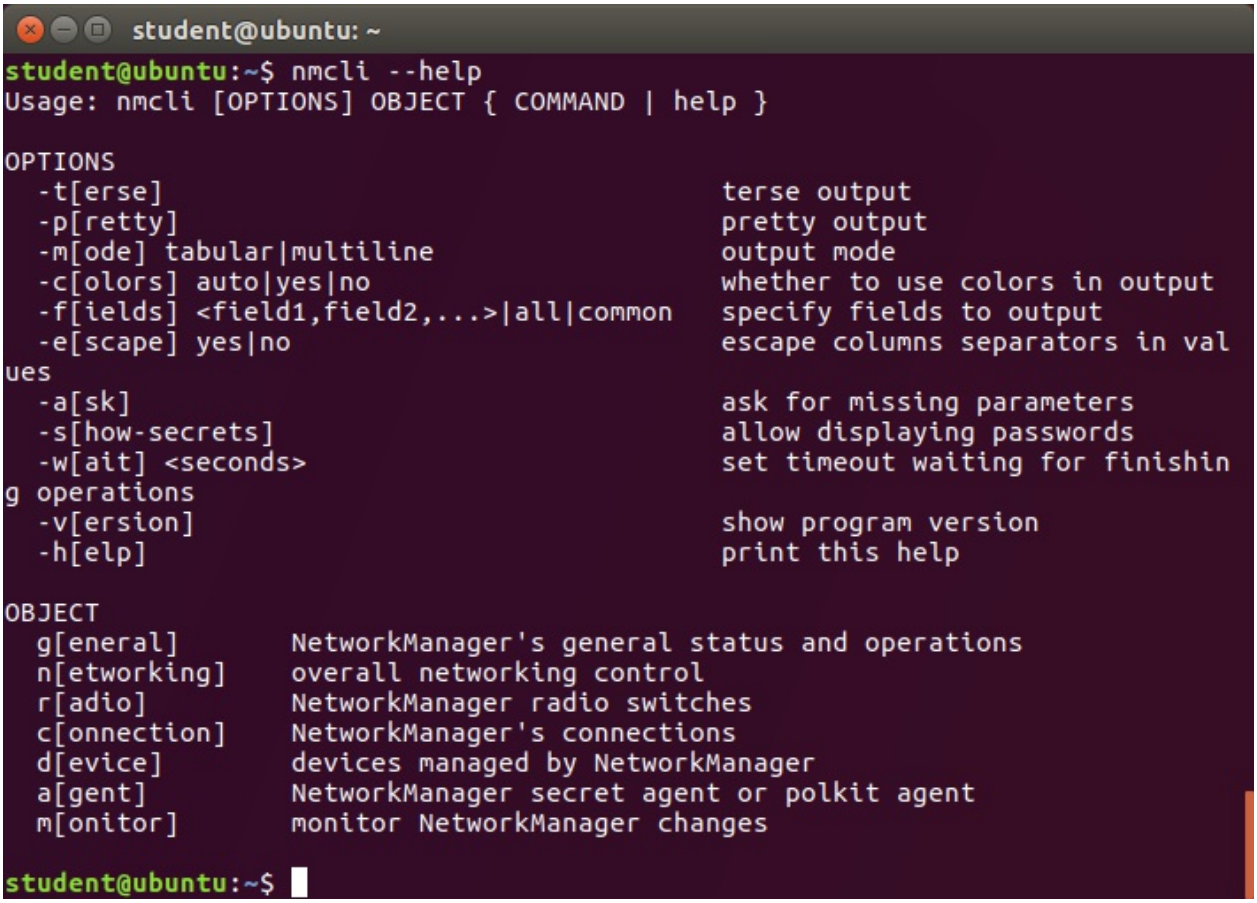
35.16 nmcli

nmcli: command line interface to **Network Manager**. Can issue direct commands, but also has interactive mode.

For many details/examples, can visit Networking/CLI Fedora wiki webpage or can type:

```
$ man nmcli-examples
```

Will explore use of **nmcli** in lab exercises.



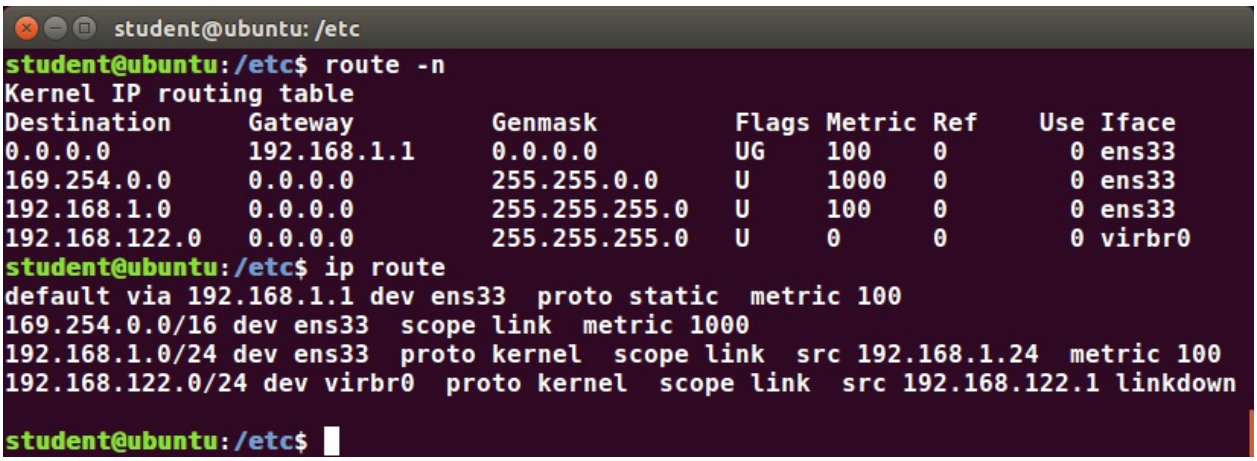
35.17 Routing

Routing: process of selecting paths in netw ork along w hich to send netw ork traffic.

Routing table: list of routes to other netw orks managed by system. Defines paths to all netw orks and hosts, sending remote traffic to routers.

To see the current routing table, can use **route** or **ip**:

```
$ route -n
$ ip route
```



35.18 Default Route

Default route: the way packets are send when there is no other match in routing table for reaching specified network.

Can be obtained dynamically using DHCP. However, can also be manually configured (static). With **nmcli** can be done via:

```
$ sudo nmcli con mod virbr0 ipv4.routes 192.168.10.0/24 \
             +ipv4.gateway 192.168.122.0
$ sudo nmcli con up virbr0
```

or can modify configuration files directly. On Red Hat-based systems, can modify `/etc/sysconfig/network` putting in the line:

```
GATEWAY=x.x.x.x
```

or alternatively in `/etc/sysconfig/network-scripts/ifcfg-ethX` on device-specific basis in configuration file for individual NIC. On Debian-based systems, equivalent is putting:

```
gateway=x.x.x.x
```

in `etc/network/interfaces`.

On either system can set default gateway at runtime with:

```
$ sudo route add default gw 192.168.1.10 enp2s0
$ route
Kernel IP routing table
Destination   Gateway      Genmask       Flags Metric Ref Use Iface
default       192.168.1.10 0.0.0.0       UG    0      0     0 enp2s0
default       192.168.1.1  0.0.0.0       UG    1024   0     0 enp2s0
172.16.132.0  0.0.0.0      255.255.255.0 U     0      0     0 vmnet1
192.168.1.0   0.0.0.0      255.255.255.0 U     0      0     0 enp2s0
192.168.113.0 0.0.0.0      255.255.255.0 U     0      0     0 vmnet8
```

Note: this might wipe out your network connection! Can restore either by resetting network, or in above example by doing:

```
$ sudo route add default gw 192.168.1.1 enp2s0
```

These changes not persistent, will not survive system restart.

35.19 Static Routes

Static routes: used to control packet flow when there is more than one router or route. Defined for each interface and can be either persistent or non-persistent.

When system can access more than one router, or perhaps there are multiple interfaces, useful to selectively control which packets go to which router.

Either **route** or **ip** command can be used to set a non-persistent route as in:

```
$ sudo ip route add 10.5.0.0/16 via 192.168.1.100
$ route
Destination  Gateway      Genmask        Flags Metric Ref Use Iface
default      192.168.1.1  0.0.0.0        UG    0      0   0   eth0
10.5.0.0     quad         255.255.0.0    UG    0      0   0   eth0
192.168.1.0  *            255.255.255.0  U     1      0   0   eth0
```

On Red Hat-bases system, persistent route can be set up editing `/etc/sysconfig/network-scripts/route-ethX` as show n by:

```
$ cat /etc/sysconfig/network-scripts/route-eth0
10.5.0.0/16 via 172.17.9.1
```

On Debian-based systems, need to add lines to `/etc/network/interfaces` such as:

```
iface eth1 inet dhcp
    post-up route add -host 10.1.2.51 eth1
    post-up route add -host 10.1.2.52 eth1
```

On SUSE-based system, need to add or create file such as `/etc/sysconfig/network/ifroute-eth0` w ith lines like:

```
# Destination Gateway Netmask Interface [Type] [Options]
192.168.1.150 192.168.1.1 255.255.255.255 eth0
10.1.1.150 192.168.233.1.1 eth0
10.1.1.0/24 192.168.1.1 - eth0
```

w here each field is separated by tabs.

35.20 Name Resolution

Name Resolution: act of translating hostnames to IP addresses of their hosts. Eg., brow ser or email client w ill take `training.linuxfoundation.org` and resolve he name to the IP address of the server (or servers) that serve `training.linuxfoundation.org` in order to transmit to and from that location.

There are tw o facilities for doing this translation:

- Static name resolution (using `/etc/hosts`).
- Dynamic name resolution (using DNS servers).

There are several command line tools that can be used to resolve IP address of hostname:

```
$ [dig | host | nslookup] linuxfoundation.org
```

- **dig**: generates the most information, has many options
- **host**: more compact
- **nslookup**: older

dig is the new est and the other are sometimes considered deprecated, but the output for host is easiest to read and contains the basic information.

One sometimes also required **reverse resolution**: converint IP address to host name. Try feeding these three utilities a know n IP address instead of hostname, and examine output.

35.21 /etc/hosts

`/etc/hosts` holds local database of hostnames and IP addresses. Contains set of records (each taking one line) which map IP addresses with corresponding hostnames and aliases.

Typical `/etc/hosts` file looks like:

```
$ cat /etc/host
127.0.01      localhost localhost.localdomain localhost4 localhost4 localdomain4
::1           localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.1.100 hans hans7 hans64
192.168.1.150 bethe bethe7 bethe64
192.168.1.2   hp-printer
192.168.1.10  test32 test64 oldpc
```

Such static name resolution primarily used for local, small, isolated networks. Generally checked before DNS attempted to reolve address; however, priority can be controlled by `/etc/nsswitch.conf`.

The other host-related files in `/etc` are `/etc/hosts.deny` and `/etc/hosts.allow`. These are self-documenting and their purpose is obvious from their names. The `allow` file searched first, and `deny` file only searched if query is not found there.

`/etc/host.conf` contains general configuration information; rarely used.

35.22 DNS

If name resolution cannot be done locally using `/etc/hosts`, then system will query a DNS (**D**omain **N**ame **S**erver) server.

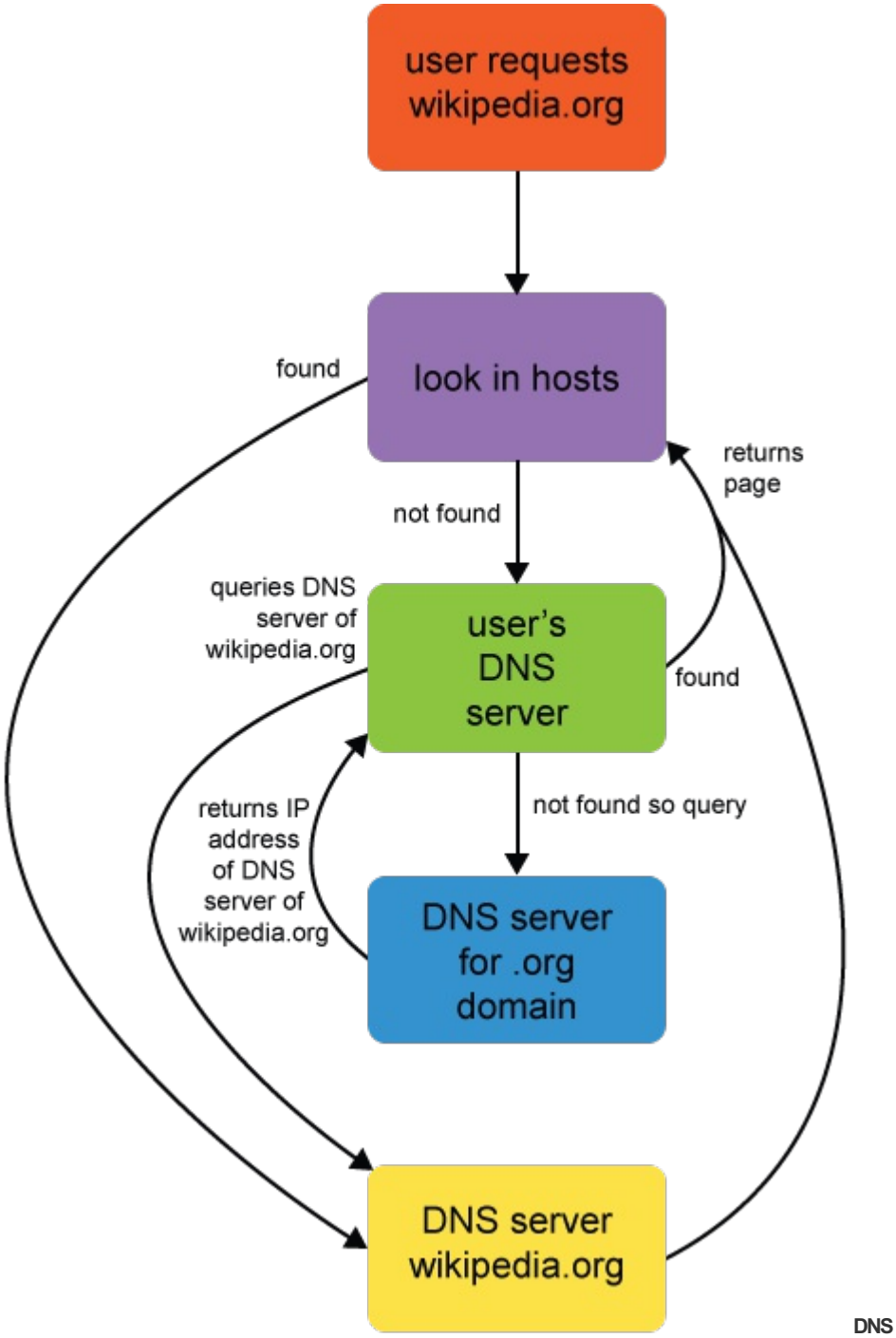
DNS dynamic and consists of network of servers which client uses to look up names. Service distributed; any one DNS server has only information about its **zone of authority**; however, all of them together can cooperate to resolve any name.

Machine's usage of DNS configured in `/etc/resolv.conf`, which historically has looked like:

```
search example.com aps.org
nameserver 192.168.1.1
nameserver 8.8.8.8
```

which:

- Can specify particular domains to search
- Defines a strict order of nameservers to query



DNS

35.23 Network Diagnostics

Number of basic network utilities in every system administrator's toolbox, including:

- **ping**

  Send 64-byte test packets to designated network hosts and (if it finds them) tries to report back on the time required to reach it (in milliseconds), any lost packets, and some other parameters. Note that the exact output will vary according to the host being targeted, but you can at least see that the network is working and the host is reachable.

- **traceroute**

  Is used to display a network path to a destination. Shows the routers packets flow through to get to a host, as well as the

time it takes for each **hop**

- **mtr**

  Combines functionality of **ping** and **traceroute** and creates a continuously updated display, like **top**

- **dig**

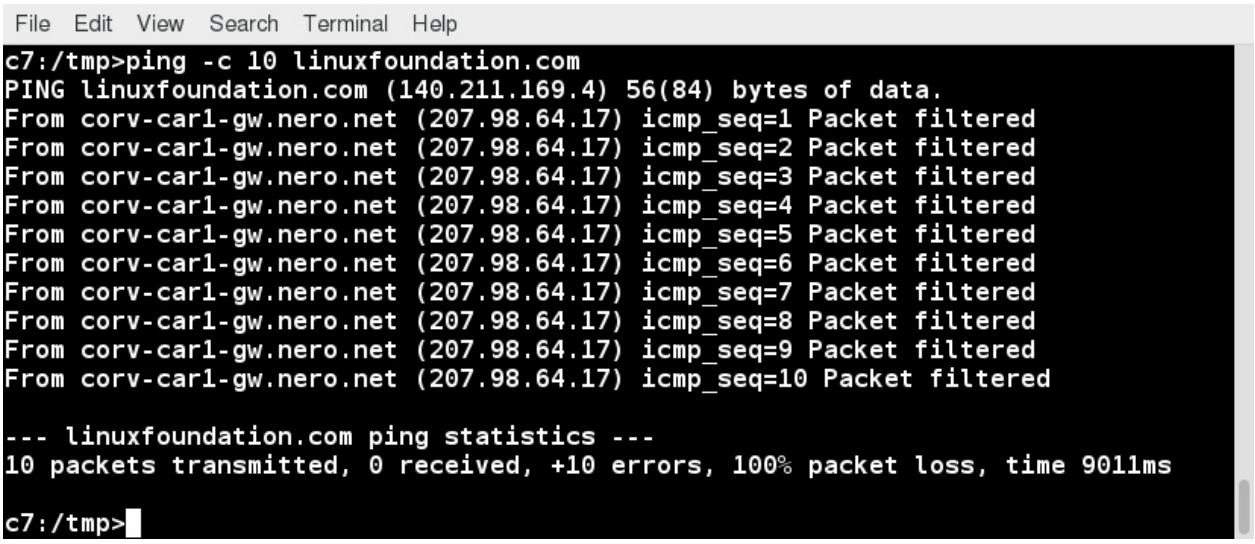
  Is useful for testing DNS functionality. Note that one can also use **host** or **nslookup**, older programs that also try to return DNS information about a host.

Note: some recent distributions (such as RHEL 7) require root privilege (as with **sudo**) in order to run the first three diagnostic utilities.

Examples:

```
$ ping -c 10 linuxfoundation.org
$ traceroute linuxfoundation.org
$ mtr linuxfoundation.org
```

35.24 ping Example



35.25 traceroute Example

```
File  Edit  View  Search  Terminal  Help
c7:/tmp>traceroute www.linuxfoundation.org
traceroute to www.linuxfoundation.org (140.211.169.4), 30 hops max, 60 byte packets
 1  gateway (192.168.1.1)  0.386 ms  0.455 ms  0.510 ms
 2  * * *
 3  dtr02ftbgwi-tge-0-6-0-3.ftbg.wi.charter.com (96.34.24.122)  24.020 ms  24.476 ms  24.555 ms
 4  crr01ftbgwi-bue-101.ftbg.wi.charter.com (96.34.30.246)  24.273 ms  24.371 ms  24.623 ms
 5  crr01euclwi-bue-400.eucl.wi.charter.com (96.34.24.224)  26.629 ms  26.521 ms  26.707 ms
 6  crr02sgnwmi-tge-0-0-0-4.sgnw.mi.charter.com (96.34.2.153)  32.672 ms  17.489 ms  18.903 ms
 7  bbr02euclwi-bue-5.eucl.wi.charter.com (96.34.0.7)  21.910 ms  22.064 ms  21.976 ms
 8  bbr01chcgil-bue-1.chcg.il.charter.com (96.34.0.9)  47.642 ms  47.696 ms  47.540 ms
 9  prr01chcgil-bue-2.chcg.il.charter.com (96.34.3.9)  50.501 ms  50.580 ms  50.366 ms
10  ge-7-2-3.0.rtr.eqch.net.internet2.edu (64.57.20.49)  50.277 ms  56.285 ms ge-7-2-0.0.rtr.eqch.
net.internet2.edu (162.252.69.153)  40.472 ms
11  ae-1.80.rtsw.chic.net.internet2.edu (64.57.20.150)  39.146 ms  39.147 ms  39.155 ms
12  ae-0.80.rtsw.kans.net.internet2.edu (64.57.20.148)  50.416 ms  48.018 ms  52.376 ms
13  ae-0.80.rtsw.salt.net.internet2.edu (64.57.20.146)  81.953 ms  79.413 ms  84.802 ms
14  ae-2.80.rtsw.losa.net.internet2.edu (64.57.20.144)  86.829 ms  86.982 ms  84.276 ms
15  nero-cps1.ptck-core1-gw.oregon-gigapop.net (198.32.165.198)  93.184 ms  93.248 ms  94.575 ms
16  ptck-p2-gw.nero.net (207.98.64.170)  96.579 ms ptck-p1-gw.nero.net (207.98.64.168)  97.037 ms
ptck-p2-gw.nero.net (207.98.64.170)  95.485 ms
17  corv-p2-gw.nero.net (207.98.64.27)  97.880 ms corv-p1-gw.nero.net (207.98.64.25)  98.413 ms co
rv-p2-gw.nero.net (207.98.64.27)  102.088 ms
18  corv-car1-gw.nero.net (207.98.64.19)  98.220 ms  95.977 ms corv-car1-gw.nero.net (207.98.64.17
)  98.659 ms
19  corv-car1-gw.nero.net (207.98.64.17)  95.829 ms !X  93.986 ms !X *
c7:/tmp>
```

35.26 mtr Example

```
File  Edit  View  Search  Terminal  Help
                                 My traceroute  [v0.85]
c7 (0.0.0.0)                                                  Thu May 25 09:51:00 2017
Unable to allocate IPv6 socket for nameserver communication: Address family not supported by pro
tocol
                                                 Packets               Pings
 Host                                          Loss%   Snt   Last   Avg  Best  Wrst StDev
                                                0.0%    16    0.3   0.3   0.2   0.3   0.0
 2. ???
 3. dtr02ftbgwi-tge-0-6-0-3.ftbg.wi.charter.com      80.0%    16   11.9   9.4   7.9  11.9   2.1
 4. crr01ftbgwi-bue-101.ftbg.wi.charter.com          80.0%    16    8.2   8.0   7.0   8.7   0.7
 5. crr01euclwi-bue-400.eucl.wi.charter.com          80.0%    16   15.0  14.0  13.3  15.0   0.7
 6. crr02sgnwmi-tge-0-0-0-4.sgnw.mi.charter.com      86.7%    16   21.7  20.2  18.7  21.7   2.0
 7. bbr02euclwi-bue-5.eucl.wi.charter.com            86.7%    16   25.6  22.7  19.7  25.6   4.1
 8. bbr01chcgil-bue-1.chcg.il.charter.com            86.7%    16   37.7  40.9  37.7  44.2   4.5
 9. prr01chcgil-bue-2.chcg.il.charter.com            81.8%    12   37.0  36.3  35.7  37.0   0.0
10. xe-8-1-2.0.rtr.eqch.net.internet2.edu            81.8%    12   35.4  35.9  35.4  36.3   0.0
11. ae-1.80.rtsw.chic.net.internet2.edu              81.8%    12   37.3  36.5  35.7  37.3   1.0
12. et-3-1-0.4072.rtsw.kans.net.internet2.edu        81.8%    12   47.8  47.8  47.8  47.8   0.0
13. ae-0.80.rtsw.salt.net.internet2.edu              81.8%    12   79.4  79.9  79.4  80.3   0.0
14. ae-2.80.rtsw.losa.net.internet2.edu              81.8%    12   74.0  75.1  74.0  76.3   1.4
15. nero-cps1.ptck-core1-gw.oregon-gigapop.net       81.8%    12   97.5  94.2  90.9  97.5   4.7
16. ptck-p1-gw.nero.net                              81.8%    12   93.7  93.9  93.7  94.2   0.0
17. corv-p1-gw.nero.net                              80.0%    11   95.0  97.7  95.0 100.4   3.7
18. corv-car1-gw.nero.net                            80.0%    11   94.8  94.7  94.7  94.8   0.0
19. ???
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 36 Firewalls - Notes

36.2 Introduction

Firewalls are used to control both incoming and outgoing access to your systems and local network, and are an essential security facility in modern networks, where intrusions and other kinds of attacks are a fact of life on any computer connected to the internet. You can control the level of trust afforded on traffic across particular interfaces, and/or with particular network addresses.

36.3 Learning Objectives:

- Understand what firewalls are and why they are necessary.
- Know what tools are available both at the command line and using graphical interfaces.
- Discuss about **firewalld** and the **firewall-cmd** programs.
- Know how to work with **zones**, **sources**, **services**, and **ports**.

36.4 What Is a Firewall?

Firewall: network security system that monitors and controls all network traffic. Applies **rules** on both incoming and outgoing network connections and packets and builds flexible barrier (i.e., firewalls) depending on the level of trust of a given connection.

Firewalls can be hardware or software based. They are found both in network routers, as well as in individual computers, or network nodes. Many firewalls also have routing capabilities.

36.5 Packet Filtering

Almost all firewalls based on **Packet Filtering**.

Information transmitted across networks in form of packets, and each one of these packets has:

- Header
- Payload
- Footer.

Header and footer contain information about destination and source addresses, what kind of packet it is, which protocol it obeys, various flags, which packet number this is in a stream, all sorts of other metadata about transmissions. Actual data in payload.

Packet filtering intercepts packets at one or more stages in network transmission, including application, transport, network, datalink.

Firewall establishes set of rules by which each packet may be:

- Accepted or rejected based on content, address, etc.
- Mangled in some way
- Redirected to another address
- Inspected for security reasons
- Etc.

Various utilities exist for establishing rules and actions to be taken as the result of packet filtering.

36.6 Firewall Generations

Early firewalls (dating back to the late 1980's) based on **packet filtering**: content of each network packet inspected and either dropped, rejected, or sent on. No consideration given about **connection state**: what stream of traffic the packet was part of.

Next generation of firewalls based on **stateful filters**, which also examine **connection state** of packet, to see if it is a new connection, part of an already existing one, or part of none. Denial of service attacks can bombard this kind of firewall to try and overwhelm it.

Third generation of firewalls called **Application Layer Firewalls**, aware of the kind of application and protocol the connection is using. Can block anything which should not be part of the normal flow.

36.7 Firewall Interfaces and Tools

Configuring your system's firewall can be done by:

- Using relatively low-level tools from the command line, combined with editing various configuration files in the `/etc` subdirectory tree:

  'iptables'

  'firewall-cmd'

  'ufw'

- Using robust graphical interfaces:

  'system-config-firewall'

  'firewall-config'

  'gufw'

  'yast'

Will work with lower-level tools for following reasons:

- Change less often than graphical ones
- Tend to have larger set of capabilities
- Tend to be quite different and each confined to GUI. Vary little from distribution to distribution, while only one family of distributions

Disadvantage: can seem more difficult to learn at first. In following, will concentrate on use of modern **firewalld** package, includes both **firewall-cmd** and **firewall-config**. FOr distributions which don't have it by default, can be installed from source rather easily, as will do if necessary in exercise.

36.8 Why We Are Not Working with iptables

More firewall installations today actually use **iptables** package on user side. This currently interfaces same kernel firewall implementation code as **firewalld**, which will be discussed more in detail.

Decided not to teach **iptables** because it requires much more time to get to useful functionality.

However, **iptables** discussed in detail in next course in Linux Foundation system administrator sequence: *LFS311 - Linux for System Engineers*/*LFS211 - Linux Networking and Administration*.

36.9 firewalld

firewalld: **Dynamic Firewall Manager**. Utilizes network/firewall **zones** which have defined levels of trust for network interfaces or connections. Supports both IPv4 and IPv6 protocols.

In addition, separates **runtime** and **permanent** (persistent) changes to configuration, and also includes interfaces for services/applications to add firewall rules.

Configuration files kept in `/etc/firewalld` and `/usr/lib/firewalld`. Files in `/etc/firewalld` override those in other directory and are the ones system administrators should work on.

Command line tool actually **firewall-cmd** which will be discussed. Run before getting any further:

```
$ firewall-cmd --help
Usage: firewall-cmd [OPTIONS...]
....
Status options
   --state                Return and print firewalld state
   --reload               Reload firewall and keep state information
   --complete-reload      Reload firewall and loose state information
   --runtime-to-permanent  Create permanent from runtime configuration
....
```

which runs about 200 lines, too long to be included here.

Note: will see that almost all options rather obvious, as well named. As a service, **firewalld** replaces older **iptables** Error to run both services, **firewalld** and **iptables**, at same time.

36.10 firewalld Service Status

firewalld: service which needs to be running to use and configure the firewall. Enabled/disabled, or started/stopped in usual way:

```
$ sudo systemctl [enable/disable] firewalld
$ sudo systemctl [start/stop] firewalld
```

Can show current state in either of the following ways:

```
$ sudo systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
    Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
    Active: active (running) since Tue 2015-04-28 12:00:59 CDT; 5min ago
 Main PID: 777 (firewalld)
    CGroup: /system.slice/firewalld.service
            777  /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
Apr 28 12:00:59 CenOS7 systemd[1]: Started firewalld - dynamic firewall daemon.
$ sudo firewall-cmd --state
running
```

Note: if you have more than one network interface when using IPv4, have to turn on **ip forwarding**. Can do this at runtime by doing either of:

```
$ sudo sysctl net.ipv4 ip_forward=1
$ echo 1 > /proc/sys/net/ipv4/ip_forward (needs to be run as root to get echo to work properly)
```

However, this is not persistent. To do that, have to add following line to `/etc/sysctl.conf`:

```
net.ipv4.ip_forward=1
```

and then reboot or type:

```
$ sudo sysctl -p
```

to read in new setting without rebooting.

36.11 Zones

firewalld works with **zones**, each of which has defined level of trust and certain known behavior for incoming/outgoing packets. Each interface belongs to particular zone (normally, it is **Network Manager** which informs **firewalld** which zone is applicable), but this can be changed with **firewallcmd** or the **firewall-config** GUI.

The zones:

- **drop**

  All incoming packets dropped with no reply. Only outgoing connections are permitted.

- **block**

  All incoming network connections rejected. Only permitted connections are those from within the system.

- **public**

  DO not trust any computers on the network; only certain consciously selected incoming connections are permitted.

- **external**

  Used when masquerading is being used, such as in routers. Trust levels are the same as in public.

- **dmz (Demilitarized Zone)**

  Used when access to some (but not all) services are to be allowed to the public. Only particular incoming connections are allowed.

- **work**

  Trust (but not completely) connected nodes to be not harmful. Only certain incoming connections are allowed.

- **home**

  Mostly trust the other network nodes, but till select which incoming connections are allowed.

- **internal**

  Similar to **work** zone.

- **trusted**

  All network connections are allowed.

On system installation, most, if not all Linux distributions, will select the **public** zone as default for all interfaces.

The differences between some of the zones mentioned not obvious, do not need to go into that much detail. Note: one should not use a more open zone than necessary.

36.12 Zone Management

Get the default zone:

```
$ sudo firewall-cmd --get-default-zone
public
```

Obtain a list of zones currently being used:

```
$ sudo firewall-cmd --get-active-zones
public
  interfaces: eno16777736
```

List all available zones:

```
$ sudo firewall-cmd --get-zone
block dmz drop external home internal public trusted work
```

To change the default zone to **trusted** and then change it back:

```
$ sudo firewall-cmd --set-default-zone=trusted
success
$ sudo firewall-cmd --set-default-zone=public
success
```

To assign an interface temporarily to a particular zone:

```
$ sudo firewall-cmd --zone=internal --change-interface=eno1
success
```

To assign an interface to a particular zone permanently:

```
$ sudo firewall-cmd --permanent --zone=internal --change-interface=eno1
success
```

which creates the file `/etc/firewalld/zones/internal.xml`.

To ascertain the zone associated with a particular interface:

```
$ sudo firewall-cmd --get-zone-of-interface=eno1
public
```

Finally, to get all details about a particular zone:

```
$ sudo firewall-cmd --permanent --zone=public --list-all
public (default, active)
  interfaces: eno16777736
  sources:
  services: dhcpv6-client ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
```

36.13 Source Management

Any zone can be bound not just to network interface, but also to particular network addresses. Packet associated with zone if:

- it comes from source address already bound to the zone; or if not,
- it comes from an interface bound to the zone.

Any packet not fitting the above criteria is assigned to the default zone (i.e., usually **public**).

To assign a source to a zone (permanently):

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=192.168.1.0/24
success
```

This says anyone with an IP address of `192.168.1.x` will be added to the **trusted** zone.

Note: can remove previously assigned source from zone by using `--remove-source` option, or change zone by using `--change-source`.

Can list the sources bound to a zone with:

```
$ sudo firewall-cmd --permanent --zone=trusted --list-sources
192.168.1.0/24
```

In both of above commands, if you leave out the `--permanent` option, you get only the current runtime behavior.

36.14 Service Management

So far, have assigned particular interfaces and/or addresses to zones, but haven't delineated what **services** and **ports** should be accessible within a zone.

To see all the services available:

```
$ sudo firewall-cmd --get-services
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns ftp \
high-availability http https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps \
```

```
libvirt libvirt-tls mdns mountd ms-wbt mysql nfs ntp openvpn pmcd pmproxy pmwebapi \
pmwebapis pop3s postgresql proxy-dhcp radius rpc-bind samba samba-client smtp ssh \
telnet tftp tftp-client transmission-client vnc-server wbem-https
```

or, to see those currently accessible in a particular zone:

```
$ sudo firewall-cmd --list-services --zone=public
dhcpv6-client ssh
```

To add a service to a zone:

```
$ sudo firewall-cmd --permanent --zone=home --add-service=dhcp
success
$ sudo firewall-cmd --reload
```

Second command, w ith `reload`, needed to make change effective. Also possible to add new services by editing files in `/etc/firewalld/services`.

36.15 Port Management

Port management very similar to service management:

```
$ sudo firewall-cmd --zone=home --add-port=21/tcp
success
$ sudo firewall-cmd --zone=home --list-ports
21/tcp
```

w here by looking at `/etc/services`, can ascertain that port 21 corresponds to **ftp**:

```
$ grep " 21/tcp" /etc/services
ftp                21/tcp
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 37 System Startup and Shutdown - Notes

37.3 Learning Objectives:

- Explain the boot process.
- Identify several types of boot loaders.
- Describe what the BIOS does.
- Identify the relevant configuration files.
- Describe how the system shuts down and reboots.

37.4 Boot Sequence

Basic steps in boot sequence:

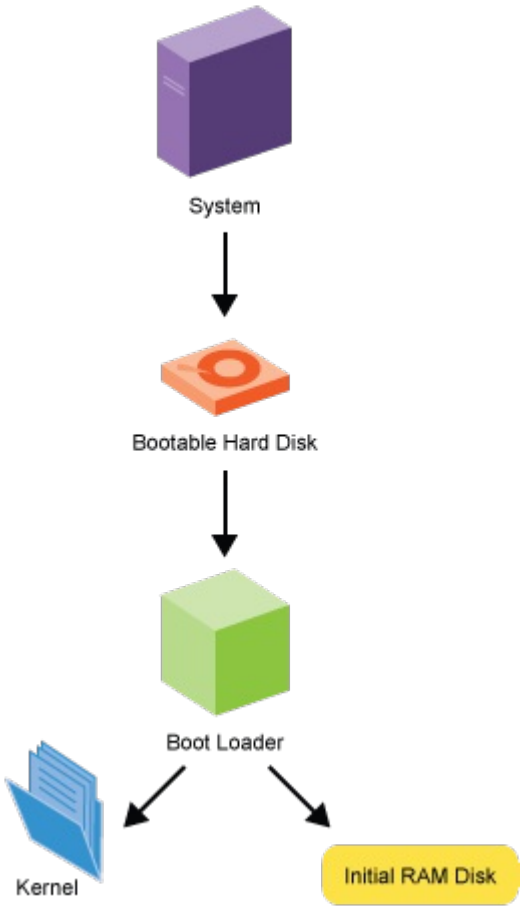
1. BIOS/UEFI locates and executes the boot program, or boot loader.
2. Boot loader loads the kernel.
3. Kernel starts the **init** process (`pid=1`).
4. **init** manages system initialization, using systemd or the older Upstart and SysVinit startup scripts.

When power applied to computer, computer can only perform operations the BIOS (**B**asic **I**nput **O**utput **S**ystem) orders it to do.

First, BIOS runs POST (**P**ower **O**n **S**elf **T**est), which checks memory and hardware, then searches specific location or device for boot program. Typically, boot program found in device's MBR (**M**aster **B**oot **R**ecord). Control of computer is then transferred to this boot program (usually GRUB).

Boot program then loads kernel into memory and executes it. On x86 platforms (and many others), kernel first has to decompress itself in place. It then performs hardware checks, gains access to important peripheral hardware, and eventually runs **init** process. This first process continues the system startup, managing either systemd or Upstart, or running appropriate init scripts is SysVinit is being used.

Newer computers are moving to UEFI, a replacement for BIOS, which performs many of the same functions.



The Boot Sequence

37.5 BIOS

On x86 architecture, BIOS contains all the code required to gain initial access to the keyboard, display screen, disk drives, serial communications, and a number of miscellaneous functions. Once full system running, most of these devices w ill have enhanced capabilities w hen complete and specialized device drivers can be loaded and take over.

BIOS typically placed in ROM chip that comes w ith computer (often called ROM BIOS). This ensures that BIOS w ill alw ays be available and w ill not be damaged by disk failures. This also makes it possible for computer to boot itself.

During boot process, BIOS loads boot loader from MBR.

37.6 Boot Loaders

Number of different boot loaders used in Linux:

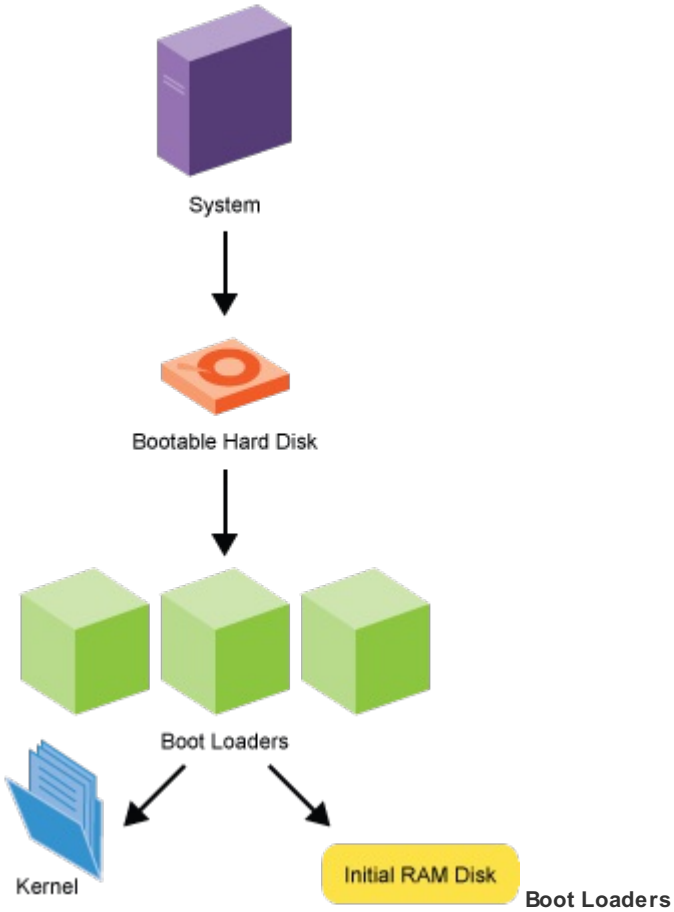
- **GRUB**
- **efibootmgr**
- **LILO**
- **Das U-Boot**

Virtually, all (non-embedded) modern Linux distributions use **GRUB** (**GR**and **U**nified **B**oot Loader). GRUB's features include ability to boot multiple operating systems, both graphical and text-based interface allow ing ease of use over serial cable, pow erful command line interface for interactive configuration, netw ork-based diskless booting, and other advanced features.

efibootmgr not actually a boot loader, but is a **boot manager**, used in conjunction w ith **GRUB** on multi-boot EFI systems.

The **Li**nux **Lo**ader (**LILO**) is older and obsolete.

Das U-Boot is the most popular loader for embedded Linux systems. Some other boot loaders, including **bareboot**. How ever, not really considering embedded space in this course.



Boot Loaders

37.7 Configuration Files in /etc

Earlier, discussed about w here Linux distributions cooperate, and hopefully follow agreed-upon standards to place certain kinds of files in standard places on system.

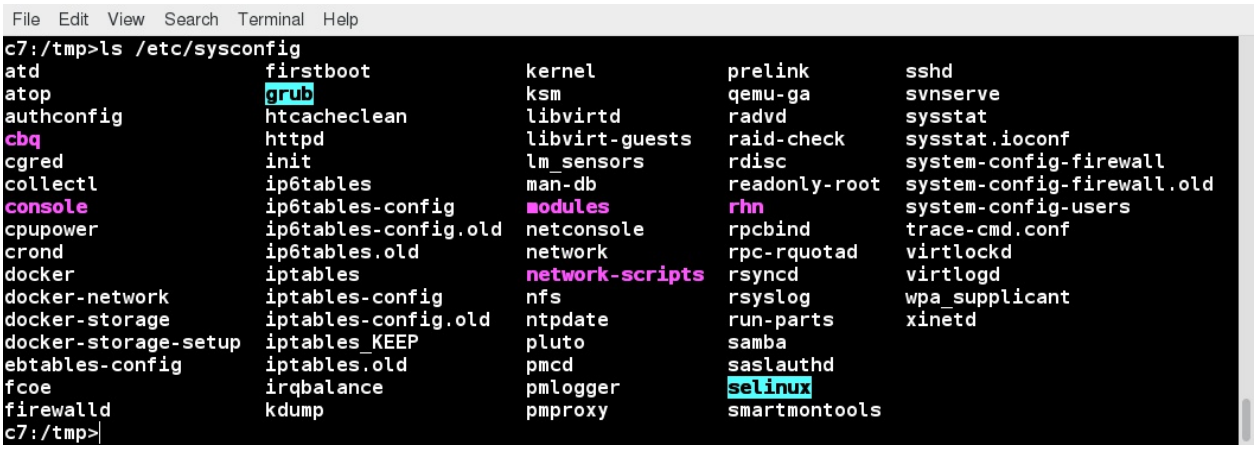
In particular, system-side configuration files generally placed in `/etc` and its subdirectories, w hile user-specific ones often placed in their individual home directories. Not completely true, though; eg, default configuration information might be stored in `/usr/lib/systemd`, but can be overridden by files in `/etc/systemd`.

FOr historical reasons, Linux distributions evolved their ow n rules about exactly w here to place some information in `/etc`. Eg., all Red Hat-derived systems make extensive use of `/etc/sysconfig`, w hile Debian-based systems have used `/etc/default`. Interestingly, RHEL 7 and SUSE use both.

37.8 /etc/sysconfig

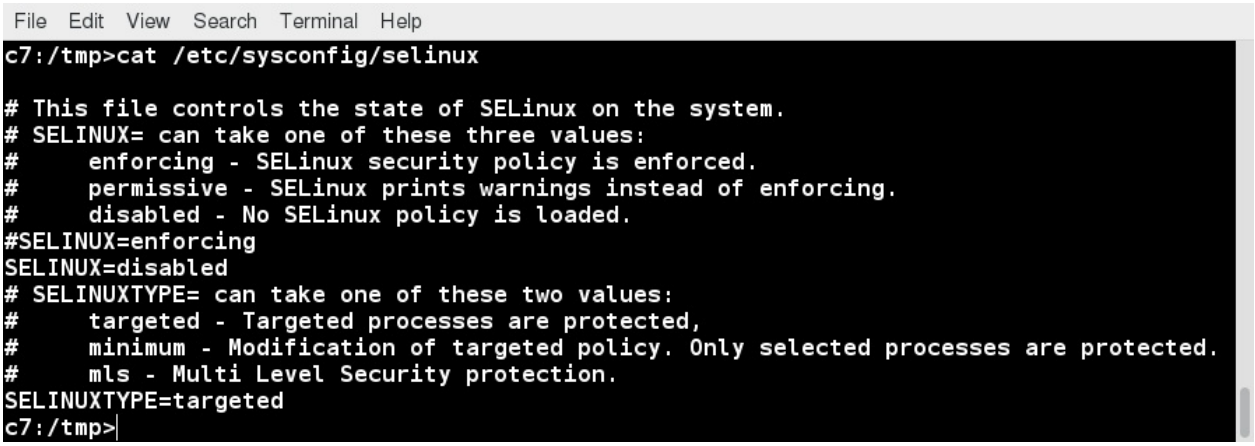
On RHEL 7 systems, files in `/etc/sysconfig` used w hen starting, stopping, configuring, or querying system services.

```
$ ls /etc/sysconfig
```



Can take a look at one file in screenshot here; this files reads and sets the **selinux** configuration at system startup.

```
$ cat /etc/sysconfig/selinux
```



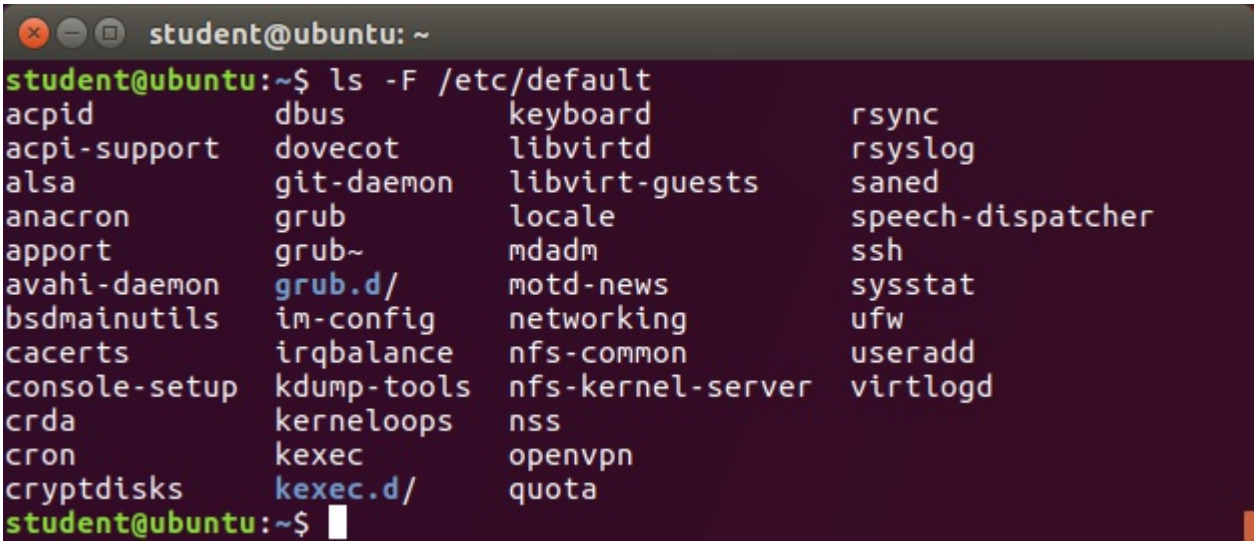
37.9 /etc/default

Screenshot shows `/etc/default` directory on Ubuntu 17.04 system.

Use of this directory similar to that of Red Hat's `/etc/sysconfig`:

- Files used to provide extra options when starting a service
- Typically contain code to set environment variables.

Eg., file `/etc/default/useradd` sets defaults that are used when new user accounts are being created. As noted, RHEL 7 also has this directory as its use if becoming more universal.

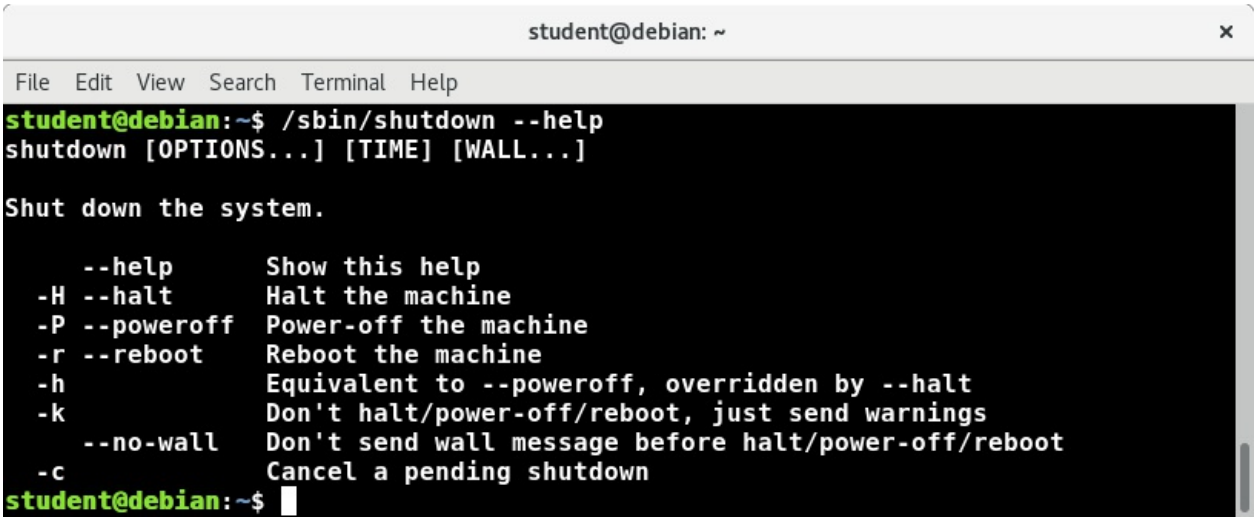


37.10 Shutting Down and Rebooting

shutdown used to bring system down in secure fashion, notifying all users that system is going down and then stopping it in a graceful and non-destructive way. After it is shutdown, system if either halted or rebooted. Can see some shutdown examples here:

```
$ sudo shutdown -h +1 "Power Failure imminent"
$ sudo shutdown -h now
$ sudo shutdown -r now
$ sudo shutdown now
```

Options can easily be listed by built-in help message.



With no options (eg. `shutdown now`), default behavior is to power off the system completely. Some distributions, such as Ubuntu, violate this and go to single user mode instead.

One common mistake is failing to include a time argument (such as `now` or some actual time). This is required.

There are also lagacy commands **reboot**, **halt**, **poweroff**, which many veteran users use frequently.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 38 GRUB - Notes

38.2 Introduction

A system that can not start up on its own is rather useless. Linux systems have a lot of flexibility in how they boot. Possible to choose between different kernel versions or options, or even different operating systems in multiple-boot scenarios. Most non-embedded systems use the Grand Unified Boot Loader to accomplish the first steps of a successful system initialization. GRUB also has interactive capabilities as well as a secure password facility.

38.3 Learning Objectives:

- Explain what the role of GRUB is.
- Understand the differences between the GRUB 1 and GRUB 2 versions.
- Explain the interactive selections you can make at boot.
- Install GRUB.
- Explain how the configuration files that GRUB needs are used and modified.

38.4 What is GRUB?

Virtually, all x86-based Linux systems (outside the embedded sphere) today use GRUB (**GR**and **U**nified **B**ootloader) to handle early phases of system startup. Other platforms may have other equivalents, such as ELILO used on EFI systems such as IA64 (Itanium), and Das U-BOOT used on many embedded configurations.

Some important features of GRUB are:

- Alternative operating systems can be chosen at boot time.
- Alternative kernels and/or initial ramdisks can be chose at boot time for a given operating system.
- Boot parameters can be easily changed at boot time without having to edit configuration files, etc. in advance.

GRUB Version 2 has replaced Version 1 on all major Linux distributions, except for RHEL 6-based ones. While the details are different between versions, basic philosophy is the same. Thus, will concentrate almost exclusively on Version 2.

At boot, basic configuration file is read, either `/boot/grub/grub.cfg` or `/boot/grub2/grub.cfg`, depending on distribution.

This file is auto-generated by **update-grub** (or **grub2-mkconfig** on RHEL 7) based on configuration files in the `/etc/grub.d` directory and on `/etc/default/grub` and should never be edited by hand. Usually, these utilities are run from other distribution-supplied scrips used for updating or compiling Linux kernels.

For either version, relevant essential configuration file contains some global parameters and then a stanza for each operating system or kernel configured.

38.5 Interactive Selections with GRUB at Boot

Upon system boot, after the initial POST and BIOS stages, GRUB will be entered and will display a menu. This may or may not have graphics in it (at least for the background splash screen).

Menu contains list of bootable images from one or more Linux distributions or operating systems. There may also be submenus with even more choices.

Using up and down arrows and the **Enter** key, can select the right boot option, or can wait for a configurable time period before the default choice is entered.

However, can do much more. After selecting an entry, can type **e** for edit, and then enter into interactive shell. In this shell, can alter the **stanza** in the configuration file that describes the particular boot option. Usually, do this to alter the **kernel command line**; eg., adding the word **single** at the end of the command line will cause system to boot in single user mode in order to take corrective action. Once desired change is made, can hit the right key to make the system boot.

At bottom of screen, will see displayed information on the exact key strokes, so there is no need to memorize.

Note: any changes made to configuration are **not persistent** and will be lost on next boot. So, for permanent change, need to change the actual files on the machine, using the right utilities.

Also possible to enter a pure shell, rather than edit a particular **stanza**. Can run a number of different commands and even try to re-install or repair GRUB. If there are serious problems, like not being able to find a configuration file, GRUB reverts back to this command line mode and you may be able to rescue the system without resorting to rescue media.

38.6 Installing GRUB

The word **installing** can have several different meanings with respect to GRUB:

1. Installing the **grub** program and associated utilities in their proper locations. In GRUB 1 there is actually a program called **grub**, but in GRUB 2 there are a bunch of utilities with names like **grub2-*** or **grub-***; how they are packaged is rather distribution dependent.
2. Installing the files GRUB needs to operate at boot time, under either `/boot/grub` or `/boot/grub2`. Separate than the files the Linux kernel needs (`vmlinuz-*`, `initramfs-*`) which will need to be in `/boot` directory as well.
3. Installing GRUB as the **boot loader** in system; usually, done at the front of the entire hard disk, but can also be done in a partition and accessed via **chainloading** from one GRUB to another.

If you do not install GRUB during a system installation, or need to re-install at some later point, exact procedure for doing so depends on GRUB version. For either version, relevant essential configuration file contains some global parameters, and then, a stanza for each operating system or kernel configured.

For Version 2, installation procedure can be as easy as:

```
$ sudo grub2-install /dev/sda
```

Please read **man** page carefully before running such command. Many options, and messing up GRUB can make system un-bootable. In particular, have to tell system where to find `/boot` directory, what partition it resides in.

Note: on EFI multi-boot systems, may have to also run **efibootmgr** as things can be more complex. See **man** page.

38.7 GRUB Device Nomenclature

In both GRUB versions, the first hard drive is denoted as `hd0`, second is `hd1`, etc. However, in Version 1, partitions start counting from 0, and in Version 2 from 1:

- `sda1` is `(hd0,1)` in **GRUB 2**, but `(hd0,0)` in **GRUB 1**.
- `sdc4` is `(hd2,4)` in **GRUB 2**, but `(hd2,3)` in **GRUB 1**.

There is no need to emphasize that getting confused about this can be rather destructive. Both versions of GRUB sometimes use `sda#` notation, and sometimes use `(hd0,#)`, so can get further confusing.

Within configuration file, each stanza has to specify what the root partition is; not the same as what is meant when talking about

root directory of system. In this context, means the partition contains the kernel itself (in the `/boot` directory). Eg., very common to have `/boot` in its own partition, such as `/dev/sda1`.

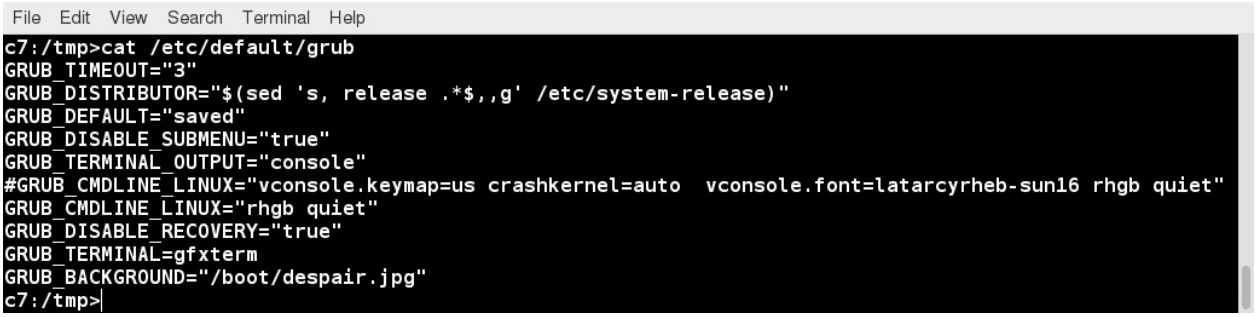
Also fine to do `kernel (hd0,0)/vmlinuz....` instead, and leave out `root` line.

Quick look at `grub.cfg` should make it clear.

38.8 GRUB Configuration File

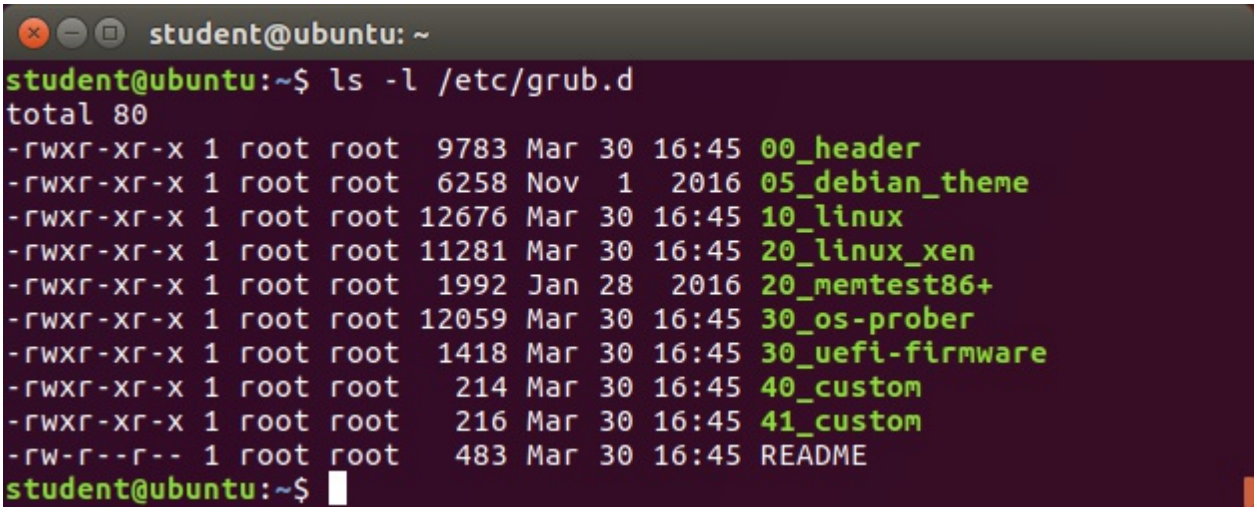
Remember: should not edit `grub.cfg` directly. Two locations in `/etc` directory that should require attention and are used to reconstruct `grub.cfg` whenever system altered with new kernels, or relevant updating program (such as **update-grub** or **grubby**) manually run.

First location: `/etc/default/grub`. Screenshot shows how this looks like on RHEL 7 system. Red Hat has really slimmed down this file compared to other distributions.



Second location: `/etc/grub.d`. Screenshot shows how it looks on Ubuntu 17.04.

Each of the two files (`/etc/default/grub` and `/etc/grub.d`) run in ascending order when configuration file updated. Won't discuss them here, as they are self-documenting and recommend to look at them.



##

Back to top

Chapter 39 init: SystemV, Upstart, systemd - Notes

39.3 Learning Objectives:

- Understand the importance of the **init** process
- Understand how **systemd** (and **Upstart**) arose and how they work.
- Use **systemctl** to configure and control **systemd**.
- Explain how the traditional **SysVinit** method works and how it incorporates **runlevels** and what happens in each one.
- Use **chkconfig** and **service** (and alternative utilities) to start and stop services or make them persistent across reboots when using **SysVinit**.

39.4 The init Process

`/sbin/init` (usually called **init**): first user-level process (or task) run on system, continues to run until system is shutdown. Traditionally, has been considered **parent** of all user processes, although technically that is not true, as some processes are started directly by kernel.

init coordinates later stages of boot process, configures all aspects of the environment, and start processes needed for logging into the system. **init** also works closely with kernel in cleaning up after processes when they terminate.

Traditionally, nearly all distributions based **init** process on UNIX's venerable **SysVinit**. However, this scheme was developed decades ago under rather different circumstances:

- Target was multi-user mainframe systems (and not personal computers, laptops, and other devices)
- Target was single processor system
- Startup (and shutdown) time was not an important matter; it was far less important than getting things right.

Startup viewed as serial process, divided into series of sequential stages. Each stage required completion before the next could proceed. Thus, startup did not easily take advantage of parallel processing that could be done on multiple processors or cores.

Secondly, shutdown/reboot was seen as a relatively rare event, and exactly how long it took was not considered important.

Modern systems have required newer methods with enhanced capabilities.

39.5 Startup Alternatives

To deal with intrinsic limitations in SysVinit, new methods of controlling system startup developed. While there are others, two main schemes adopted by Enterprise distributors:

1. **Upstart**: developed by Ubuntu, first included in 6.10 release in 2006, made default in 9.10 release in 2009. Also adopted in Fedora 9 (in 2008), in RHEL 6 and its clones (CentOS, Scienctific Linux, Oracle Linux), and in openSUSE was offered as an option since version 11.3. Has also been used in various embedded and mobile devices.
2. **systemd**: more recent, Fedora was fist major distribution to adopt in 2011. Standard since RHEL 7 and Ubuntu 16.04.

RHEL 7 based on systemd and every other major Linux distribution has adopted it, made it the default or announced plans to do so. Main systemd developers closely tied to Linux kernel community. Even Ubuntu phased out Upstart in its favor.

Migrations to newer schemes complicated, bugs and missing features can be very disabling, so there have been essential required compatibility layers. Thus, SysVinit utilities and methods will persist for long time, even if under the hood things are quite

different.

History of all this and controversies quite complicated. Colorful personalities have ensured not all discussion is of technical nature In our context, w ill not look through this lens.

Next, going to concentrate on systemd and SysVinit w ith briefer section on Upstart. (even though RHEL 6 and some other distributions had used Upstart, has been thoroughly hidden behind compatibility layer using normal SysVinit utilities.)

39.6 systemd

systemd system and session manager for Linux rapidly taking root in all major distributions. Features include follow ing:

- Is compatible w ith SysVinit scripts
- Boots faster than previous systems
- Provides aggressive parallelization capabilities
- Uses socket and D-Bus activation for starting services
- Replaces shell scripts w ith programs
- Offers on-demand starting of daemons
- Keeps track of processes using cgroups
- Supports creating snapshots and restoring of the system state
- Maintains mount and automount points
- Implements an elaborate transactional dependency-based service control logic
- Can w ork as a drop-in replacement for SysVinit.

Instead of bash scripts, systemd uses `.service` files. In addition, systemd sorts all daemons into their ow n Linux cgroups (control groups).

systemd backw ard compatible w ith SysVinit and the concept of runlevels supported via runlevel **targets**. **telinit** program emulated to w ork w ith runlevels.

39.7 systemd Configuration Files

Although systemd prefers to use set of new standardized configuration files, can also use distribution-dependent legacy configuration files as fall-back.

Example of new configuration file: `/etc/hostname`, w hich w ould replace `/etc/sysconfig/network` in Red Hat, `/etc/hostname` in SUSE, and `/etc/hostname` (adopted as the standard) in Debian.

Other files might include:

- `/etcvconsole.conf` : default keyboard mapping and console font
- `/etc/sysctl.d/*.conf` : drop-in directory for kernel **sysctl** parameters
- `/etc/os-release` distribution ID file

systemd backw ard compatible w ith SysVinit, so using old commands w ill generally w ork. Supports the concept of runlevels, supported via runlevel *targets*, and **telimit** is emulated to w ork w ith runlevels.

39.8 systemctl

systemctl: main utility for managing services. Basic syntax:

```
$ systemctl [options] command [name]
```

Below, can see some examples of how you can use **systemctl**:

- To show the status of everything that systemd controls:

```
$ systemctl
```

- To show all available services:

```
$ systemctl list-units -t service --all
```

- To show only active services:

```
$ systemctl list-units -t service
```

- To start (activate) one or more units:

```
$ sudo systemctl start foo
$ sudo systemctl start foo.service
$ sudo systemctl start /path/to/foo.service
```

where a unit can be a service or a socket.

- To stop (deactivate) a service:

```
$ sudo systemctl stop foo.service
```

- To enable/disable a service:

```
$ sudo systemctl enable sshd.service
$ sudo systemctl disable sshd.service
```

Equivalent of `chkconfig on/off` and doesn't actually start the service.

Note: Some **systemctl** commands in the above examples can be run as non-root user, others require running as root or with **sudo**.

For an excellent summary fo how to go from **SysVinit** to **systemd**, see the SysVinit to Systemd Cheatsheet.

39.10 A Word About SysVinit

SysVinit was standard method for starting and shutting down systems, and for managing services for many years.

However, has been replaced on all major Linux distributions by systemd. Still value in covering how it works, as it is being replaced, as compatibility layers have been put into place to ensure the older methods can still be used. Furthermore, sometimes third party software has not been updated to use **systemctl** methods of systemd (a prominent example: VMWare, as of this writing).

Besides discussion of runlevels and other ingredients, **chkconfig** and **service** commands will also be explained. Eventually, will

probably move to drop discussion of SysVinit, as well as Upstart.

36.11 SysVinit Runlevels

As a SysVinit system starts, passes through sequence of **runlevels** which define different system states; numbered from 0 to 6.

Runlevel 0 reserved for **system halt** state, runlevel 1 for **single-user mode**, and runlevel 6 for **system reboot**. Other runlevels used to define what services running for a normal system, and different distributions define them somewhat differently. Eg., on Red Hat-based systems, runlevel 2 defined as running system without networking or **X**, runlevel 3 includes networking, and runlevel 5 includes networking and **X**. Table below summarizes SysVinit rundown levels.

System Runlevels

Runlevel	Meaning
S, s	Same as 1
0	Shutdown system and turn power off
1	Single User Mode
2	Multiple user, no NFS, only text login
3	Multiple user, with NFS and network, only text login
4	Not used
5	Multiple user, with NFS and network, graphical login with X
6	Reboot

Current runlevel can be simply displayed with **runlevel** command:

```
$ runlevel
N 5
```

where first character is the previous level, `N` means unknown.

telinit can be used to change runlevel of system. Eg., to go from runlevel 3 to runlevel 5, type:

```
$ sudo /sbin/telinit 5
```

39.12 SysVinit and /etc/inittab

When **init** process started, first thing it does is to read `/etc/inittab`. Historically, this file told **init** which scripts to run to bring system up each runlevel, and was done with series of lines, one for each runlevel:

```
id:runlevel(s):action:process
```

where:

- `id` : a unique 1-4 character identification for the entry
- `runlevel(s)` : zero or more single character or digit identifiers indicating which runlevel the action will be taken for
- `action` : describes the action to be taken
- `process` : specifies the process to be executed.

However, In more recent systems such as RHEL 6 which hide upstart behind compatibility layer, the only uncommented line and the only thing being set in this file is the default runlevel with the line:

```
id:5:initdefault
```

This is the level to stop at when booting the system. However, if another value specified on kernel command line, init ignores default. (This is done by simply appending right integer to kernel command line.) Default level is usually 5 for a full multi-user, networked graphical system, or 3 for a server without a graphical interface.

Some recent systemd-based distributions (including RHEL 7) do not use this file at all; all lines in it are comments, but it is kept around to avoid breaking old scripts.

39.13 SysVinit Startup Scripts

Traditional method is to first run the `rc.sysinit` script, which performs numerous functions, such as starting LVM, mounting filesystems, etc. This script resides in `/etc` directory, but more likely you will find it in `/etc/rc.d` with symbolic link to `/etc`.

Next, `rc` script (in same directory) is run with desired runlevel as argument. This causes system to to to `rc.d/rc[0-6].d` directory and run all the scripts in there as in:

```
$ ls -lF /etc/rc.d/rc5.d
total 0
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 K05pcmd -> ../init.d/pmcd*
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 K05pmie -> ../init.d/pmie*
....
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 S10network -> ../init.d/network*
lrwxrwxrwx. 1 root root 14 Sep 3 10.05 S19vmware -> ../init.d/vmware*
```

The `rc.local` script may be used to start system-specific applications.

Note:

- All the actual scripts are in `/etc/init.d`. Each runlevel directory just links back to them.
- **Start** scripts start with **S** in the name
- **Kill** scripts start with **K** in the name.

The existence or non-existence of a script's symbolic link in a runlevel directory determines whether or not the script is executed at that runlevel.

The number following the **K** or **S** in each script's name determines the order in which the scripts are invoked. the script name is also the name of the service.

Controlling which initialization scripts are run on entry to each runlevel involves managing the symbolic links. While it is possible to manage these links manually, there are utilities such as **chconfig** whicha re designed to do this consistently and more easily.

39.14 chkconfig

chkconfig: used to query and configure what runlevels the various sytem services are to run in. Can see some **chkconfig** examples below :

- Check particular service to see if it is set up to run in the current runlevel:

```
$ chkconfig some_service
```

This will return true if the service is configured to be running, false otherwise. Should now that even if it is configured to be running, might currently be stopped.

- See what services are configured to run in each of the runlevels:

```
$ chkconfig --list [service name]
```

- Turn on a certain service next time the system boots:

```
$ sudo chkconfig some_service on
```

- Do not turn a certain service on next time the system boots:

```
$ chkconfig some_service off
```

- You should not that the on and off do not affect the current state by starting or stopping a service. You would have to do that with:

```
$ sudo service some_service [stop | start]
```

Not difficult to add your own services and write your own startup scripts. One only has to place a script in `/etc/init.d` which has to have certain features in it (just some lines at the top!) and then use `chkconfig --add` to enable or `chkconfig --del` to disable use of the on and off instructions etc.

How does **chkconfig** actually determine which number should appear after the `S` or `K` in a symbolic link, and how does it know which runlevels to set on or off and what state to set the symbolic links in? The information is in the scripts themselves, which contain a line near the top like:

```
# chkconfig: 2345 10 90
```

The first argument after the **chkconfig:** is there to define which runlevels to have the service on by default. In the above example that means levels 2, 3, 4, and 5. The second and third numbers are the numerical prefixes in the start and stop scripts, so in the above they start with **S10** and **K90**.

39.15 service

Every operating system has **services** which are usually started on system initialization and often remain running until shutdown. Such services may be started, stopped, or restarted at any time, generally requiring root privilege. On a Linux system using or emulating SysVinit, the services are those in the `.etc.init.d` directory.

You can see the current status of a particular service by doing:

```
$ sudo service network status
Configured devices:
lo eth0 eth1 eth2 wlan0
Currently active devices:
lo eth0

$ sudo service vsftpd status
vsftpd (pid 5284) is running...
```

service takes a number of options, which vary according to the particular service; for example:

```
$ sudo service network
Usage: /etc/init.d/network {start|stop|restart|reload|status}

$ sudo service iptables
Usage: /etc/init.d/iptables {start|stop|restart|condrestart|status|panic|save}
```

All **service** really does is change directory to `/etc/init.d` and run the appropriate script in that directory with the supplied options.

Can see the status of all the services on the system with:

```
$ sudo service --status-all
acpid (pid 4170) is running...
anacron (pid 4540) is running...
atd (pid 4553) is running...
....
smartd (pid 4614) is running...
smbd is stopped
......
```

Starting and stopping services with **service** is only effective during the current operation of the system; all changes are lost upon reboot. To cause a particular service to be turned on or not during system initialization, on Red Hat-based systems one uses **chkconfig** as described earlier.

39.16 chkconfig and service on Debian-based systems

On Debian-based systems, including Ubuntu, the utilities will work only if you have installed the **sysvinit-utils** and **chkconfig** packages, as in:

```
$ sudo apt install sysvinit-utils chkconfig
```

However, revent versions of Ubuntu no longer package **chkconfig**; will have to use **update-rc.d** utility about to be described.

As alternative, can use the more native commands on these systems. Eg., equivalent of **service** utility:

```
$ sudo invoke-rc.d cups [ status | start | stop ]
$ sudo status cups
```

to check or change the **cups** situation. **status** command is more rence and preferred over **invoke-rc.d**. Likewise, equivalent of **chkconfig** would be:

```
$ sudo update-rc.d cups [ defaults | purge ]
$ sudo sysv-rc-conf cups [ on | off ]
```

Will have to consult **man** pages for full documentation.

39.17 Upstart

Upstart: **event-driven**, rather than being a set of serial precedures. Event notifications are sent to init process to tell it to execute certain commands at the right time after prerequisites have been fulfilled. Because Upstart is being superseded by systemd, w on't spend as much time on it or do exercises w ith it. Upstart configuration files include:

```
/etc/init/rcS.conf
/etc/rc.sysinit
/etc/inittab
/etc/init/rc.conf
/etc/rc[0-5].d
/etc/init/start-ttys.conf
```

When kernel starts init process, this causes `rcS.conf` script to be executed. This, in turn, causes `rc-sysinit.conf` to be run.

`rc-sysinit.conf` w ill do a number of tasks, including starting the LVM, mounting filesystems, and then executing all of the runlevel scripts for the default runlevel specified in `/etc/inittab`.

This is accomplished by executing `rc.conf` and parring it the runlevel. These runlevel scripts bring up the services on the system. Finally, additional scripts such as `prefdm.conf` (for runlevel 5 only) are executed.

As mentioned previously, `/etc/inittab` is deprecated, and is not used only for setting up the default runlevel via the `initdefault` line. Other configuration is done via Upstart jobs in the `/etc/init` directory. Generally, Upstart events w ill be found in the `/etc/event.d` directory.

Eg., number of active `tty` consoles is now set by the `ACTIVE_CONSOLES` variable in `/etc/sysconfig/init` w hich is read by the `/etc/init/start-ttys.conf` job. Default value is `ACTIVE_CONSOLES=/dev/tty[1-6]`, w hich starts a getty process on `tty1` through `tty6`.

While Upstart w as included in RHEL 6, RHEL 7 uses systemd. Ubuntu is currently the only major Linux distribution using Upstart, but has announced plans to move to systemd.

39.18 Upstart utilities

Using **initctl** you can view, start, stop jobs in much the same w ay that **service** does. Syntax is:

```
$ initctl options command
```

w here `options` can have the follow ing values:

- **start**: Starts a job
- **stop**: Stops a job
- **restart**: Restarts a job
- **reload**: Sends **HUP** signal to a job
- **status**: Queries status of a job
- **list**: Lists know n jobs
- **emit**: Emits an event

Good summary of how to use **initctl** and many other features of Upstart can be found online: Upstart Intro, Cookbook & Best Practices

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Table 4.2. SysVinit to systemd Cheatsheet (abbreviated version)

SysVinit Command	systemd Command	Notes
`service foo start`	`systemctl start foo.service`	Start a service; no effect on reboot
`service foo stop`	`systemctl stop foo.service`	Stop a service; no effect on reboot.
`service foo restart`	`systemctl restart foo.service`	Stop and then restart
`service foo reload`	`systemctl reload foo.service`	When supported, reloads relevant configuration files without interrupting pending operations.
`service foo condrestart`	`systemctl condrestart foo.service`	Restarts if the service is already running.
`service foo status`	`systemctl status foo.service`	Tells whether a service is currently running.
`ls /etc/rc.d/init.d/`	`systemctl list-unit-files --type=service`	Lists the services that can be started or stopped.
`ls /etc/rc.d/init.d/`	`ls /lib/systemd/system/*.service`	Lists all services and other units. (preferred)

chkconfig foo on	**systemctl enable foo.service**	Turns on at next reboot or other trigger.
chkconfig foo off	**systemctl disable foo.service**	Turns off at next reboot or other trigger.
chkconfig foo	**systemctl is-enabled foo.service**	Checks whether a service is or is not configured to start.
chkconfig --list	**systemctl list-unit-files --type=service**	Prints a table listing which runlevel each service is configured to be on or off.
chkconfig --list	**ls /etc/systemd/system/*.wants/**	Preferred command for same operation
chkconfig foo --list	**ls /etc/systemd/system/ *.wants/foo.service**	Lists which runlevels this service is configured to be on or off
chkconfig foo --add	**systemctl daemon-reload**	Reloads when a new service is created or any configuration is modified.

Chapter 40 Backup and Recovery Methods - Notes

40.3 Learning Objectives:

- Identify and prioritize data that needs backup.
- Employ different kinds of backup methods, depending on the situation.
- Establish efficient backup and restore strategies.
- Use different backup utilities, such as **cpio**, **tar**, **gzip**, **bzip2**, **xz**, **dd**, **rsync**, **dump**, **restore**, and **mt**.
- Describe the two most well-known backup programs, Amanda and Bacula.

40.4 Why Backups?

Whether you are administering only one personal system or a network of many machines, system backups are very important. Some of the reasons are:

- **Data is valuable**

  On-disk data is an important work product, and is therefore a commodity that we wish to protect. Recreating lost data costs time and money. Some data may even be unique, and there may be no way to recreate it.

- **Hardware fails**

  While storage media reliability has increased, so has drive capacity. Even if the failure rate per byte decreases, unpredictable failures still occur. May be pessimistic to say there are only two kinds of drives: those that have failed, and those that will fail, but it is essentially true. Using **RAID** helps, but backups are still needed.

- **Software fails**

  No software is perfect. Bugs may destroy or corrupt data. Even stable programs long in use can have problems.

- **People make mistakes**

  Everyone has heard **OOPS!** (or something much worse) coming from the next cubicle (or from their own mouth) at one time or another. Sometimes, just a simple typing error can cause large scale destruction of files and data.

- **Malicious people can cause deliberate damage**

  Could be the canonical disgruntled employee or an external hacker with a point to make. Security concerns and backup capabilities are very strongly related.

- **Unexplained events happen**

  Files can just disappear without you know how, who, or even when it occured.

- **Rewinds can be useful**

  Sometimes, restoring to an earlier **snapshot** or all or part of the system may be required.

40.5 What Needs Backup?

Some data is critical for backup, some less critical, and some never needs saving. In priority order:

1. **Definitely**
   - Business-related date
   - System configuration files
   - User files (usually under `/home`)
2. **Maybe**
   - Spooling directories (for printing, mail, etc.)
   - Logging files (found in `/var/log`, and elsewhere)
3. **Probably not**
   - Software that can easily be re-installed; on a well-managed system, this should be almost everything
   - The `/tmp` directory, because its contents are indeed supposed to be temporary
4. **Definitely not**
   - Pseudo-filesystems such as `/proc`, `/dev`, and `/sys`
   - Any swap partitions or files

Obviously, files essential to your organization require backup. Configuration files may change frequently, and along with individual user's files, require backup as well.

Logging files can be important if you have to investigate your system's history, which can be particularly important for detecting intrusions and other security violations.

Don't have to back up anything that can easily be re-installed. Also, the **swap** partitions (or files) and `/proc` filesystems are generally not useful for necessary to backup, since the data in these areas is basically temporary (just like in the `/tmp` directory).

40.6 Backup vs. Archive

All backup media have finite lifetime before becoming unreadable. Conventional estimates are listed below:

- Magnetic tapes: 10-30 years
- CDs and DVDs: 3-10 years
- Hard Disks: 2-5 years

Lifetime is very sensitive to:

- Environmental conditions (temperature, humidity, etc.)
- Quality of media
- Haivng working software that can read data on current operating systems and hardware.

Lifetime is sufficient for backup, but nor for permanent digital archiving.

For lifetimes longer than the usual backup timescales, data can be preserved using multiple copies, plus copying over to newer media from time to time.

FOr very long times (i.e., many decades, centuries, etc.), standard methods do not work easily, as everything can go obsolete, hardware, software, and document format, media.

None of the inexpensive digital formats can actually compete with paper and film for long periods of time (if they are properly stored and continuously cared for - like wine).

This is a problem serious people think about and there should be good solutions available before all is lost.

40.7 Tape Drives

Tape drives not as common as they used to be. Relatively slow and permit only sequential access. On any modern setup, rarely used for primary backup. Sometimes used for off-site storage for archival purposes for long time reference. However, magnetic

tape drives always have only finite lifetime without physical degradation and loss of data.

Modern tape drives usually of the LTO (Linear Tape Open) variety, whose first versions appeared in the late 1990s as an open standards alternative; early formats were mostly proprietary. Early versions held up to 100 GB; newer versions can hold 2.5 TB or more in a cartridge of the same size.

Day to day backups are usually done with some form of NAS (Network Attached Storage) or with cloud-based solutions, making new tape-based installations less and less attractive. However, they can still be found and system administrators may be required to deal with them.

In what follows, will try not to focus on particular physical forms for the backup media, and will speak more abstractly.

40.8 Backup Methods

Should never have all backups residing in same physical location as systems being protected. Otherwise, fire or other physical damage could lead to a total loss. In the past, this usually meant physically transporting magnetic tapes to a secure location. Today, this is more likely to mean transferring backups files over the Internet to alternative physical locations. Obviously, has to be done in secure way, using encryption and other security precautions as appropriate.

Several different kinds of backup methods can be used, often in concert with each other:

- **Full**

  Back up all files on the system.

- **Incremental**

  Backup all files that have changed since the last incremental or full backup.

- **Differential**

  Backup all files that have changed since the last full backup.

- **Multiple level incremental**

  Backup all files that have changed since the previous backup at the same or a previous level.

- **User**

  Backup only files in a specific user's directory.

40.9 Backup Strategies

Should note that backup methods useless without associated **restore** methods. Have to take into account the robustness, clarity, ease of both directions when selecting strategies.

Simplest backup scheme: do a full backup of everything once, and then perform incremental backups of everything that subsequently changes. While full backups can take a lot of time, restoring from incremental backups can be more difficult and time consuming. Thus, can use a mix of both to optimize time and effort.

Example of one useful strategy involving tapes (can easily substitute other media in description):

1. Use tape 1 for a full backup on Friday.
2. Use tapes 2-5 for incremental backups on Monday-Thursday.
3. Use tape 6 for full backup on second Friday.
4. Use tapes 2-5 for incremental backups on second Monday-Thursday.

5. Do not overwrite tape 1 until completion of full backup on tape 6.
6. After full backup to tape 6, move tape 1 to external location for disaster recovery.
7. For next full backup (next Friday) get tape 1 and exchange for tape 6.

A good rule of thumb is to have at least two weeks of backups available.

40.10 Backup utilities

A number of programs are used for backup purposes:

- **cpio**
- **tar**
- **gzip**, **bzip2**, **xz**

  cpio and **tar** create and extract **archives** of files. The archives are often compressed with **gzip**, **bzip2**, or **xz**. The archive file may be written to disk, magnetic tape, or any other device which can hold files. Archives are very useful for transferring files from one filesystem or machine to another.

- **dd**

  This powerful utility is often used to transfer raw data between media. It can copy entire partitions or entire disks.

- **rsync**

  This powerful utility can synchronize directory subtrees or entire filesystems across a network, or between different filesystem locations on a local machine.

- **dump and restore**

  These ancient utilities are designed specifically for backups. They read from the filesystem directly (which is more efficient). However, they must be restored only on the same filesystem type that they came from. There are newer alternatives.

- **mt**

  Useful for querying and positioning tapes before performing backups and restores.

40.11 Using tar for Backups

tar is easy to use:

- When creating **tar** archive, for each directory given as argument, all files and subdirectories will be included in archive
- When restoring, reconstitutes directories as necessary
- Even has `--newer` option that allows incremental backups
- Version of **tar** used in Linux can also handle backups that do not fit on one tape or whatever device being used.

Few examples of how to use **tar** for backups:

- Create an archive using `-c` or `--create`:

```
$ tar --create --file /dev/st0 /root
$ tar -cvf /dev/st0 /root
```

Can specify a device or file with `-f` or `--file` option.

- Create with multi-volume option, using `-M` or `--multi-volume` if backup won't fit on one device:

```
$ tar -cMf /dev/st0 /root
```

Will be prompted to put next tape when needed.

- Verify files with compare option, using `-d` or `--compare`:

```
$ tar --compare --verbose --file /dev/st0
$ tar -dvf /dev/st0
```

After making backup, can make sure that it is complete and correct using above verification option.

By default, **tar** will recursively include all subdirectories in archive.

When creating archive, **tar** prints message about removing leading slashes from absolute path name. While this allows for restoring files anywhere, default behavior can be modified.

Most **tar** options can be given in short form with one dash, or long form with two: `-c` is completely equivalent to `--create`.

Note: can combine options (when using short notation) so no need to type every dash.

Furthermore, single-dashed **tar** options can be used with or without dashes; i.e., `tar cvf file.tar dir1` has same result as `tar -cvf file.tar dir1`.

40.12 Using tar for Restoring Files

`-x` or `--extract` option extracts files from archive, all by default. Can narrow the file extraction list by specifying ony particular files. If directory specified, all included files and subdirectories are also extracted.

`-p` or `--same-permissions` options ensures files are restored with their original permissions.

`-t` or `--list` option lists, but does not extract, the files in the archive.

Examples:

- Extract from an archive:

```
$ tar --extract --same-permissions --verbose --file /dev/st0
$ tar -xpvf /dev/st0
$ tar xpvf /dev/st0
```

- Specify only specific files to restore:

```
$ tar xvf /dev/st0 somefile
```

- List the contents of a **tar** backup:

```
$ tar --list --file /dev/st0
$ tar -tf /dev/st0
```

40.13 Incremental Backups with tar

Can do incremental backup with **tar** using the `-N` (or the equivalent `--newer`), or the `--after-date` options. Either option requires specifying either a date or a qualified (reference) file name:

```
$ tar --create --newer '2011-12-1' -vf backup1.tar /var/tmp
$ tar --create --after-date '2011-12-1' -vzf backup1.tar /var/tmp
```

Either form creates backup archive of all files in `/var/tmp` which were modified after December 1, 2011.

Because **tar** only looks at file's date, does not consider any other changes to the file, such as permissions or file name. To include files with these changes in incremental backup, use **find** and create a list of files to be backed up.

Note: when followed by an option like `--newer`, must use the dash in options like `-vzf`, or **tar** will get confused. This kind of option specification confusion sometimes occurs with old UNIX utilities like **ps** and **tar** with complicated histories involving different families of UNIX.

40.14 Archive Compression Methods

Often desired to compress files to save disk space and/or network transmission time, especially since modern machines will often find the compress -> transmit -> decompress cycle faster than just transmitting (or copying) an uncompressed file.

Number of commonly used compression schemes available in Linux. In order of increasing compression efficiency (which comes at the cost of longer compression times):

- **gzip**

  Uses Lempel-Zip Coding (LZ277) and produces `.gz` files.

- **bzip2**

  Uses Burrow-Wheeler block sorting text compression algorithm and Huffman coding, and produces `.bz2` files.

- **xz**

  Produces `.xz` files and also support legacy `.lzma` format.

Decompression times do not vary as much as compression times do. For day-to-day use on smaller files, one usually just uses **gzip**, as it is extremely fast, but, for larger files and for archiving purposes, the other two methods are often used. E.g., The Linux Kernel Archives now offers kernels only in the **xz** format.

The `.zip` format is rarely used in Linux, except to extract legacy archives from other operating systems.

Compression utilities very easily (and often) used in combination with **tar**:

```
$ tar zcvf source.tar.gz source
$ tar jcvf source.tar.bz2 source
$ tar Jcvf source.tar.xz source
```

for producing a compressed archive. Note: first command has the exact same effect as doing:

```
$ tar cvf source.tar source; gzip -v source.tar
```

but is more efficient because:

1. There is no intermediate file storage.
2. Archiving and compression happen simultaneously in the pipeline.

For decompression:

```
$ tar xzvf source.tar.gz
$ tar xjvf source.tar.bz2
$ tar xJvf source.tar.xz
```

or even simpler:

```
$ tar xvf source.tar.gz
```

as modern versions of **tar** can sense the method of compression and take care of it automatically.

Obviously, not worth using these methods on archives whose component files are already compressed, such as `.jpg` images, or `.pdf` files.

40.15 dd

dd: one of the original UNIX utilities, extremely versatile. Without options, does very low-level raw copying of files or even whole disks. Capable of doing many kind of data conversions during the copy (such as changing byte order) and has many options to control offsets, number of bytes, block size, etc.

dd often used to read fixed amounts of data from special device nodes such as `/dev/zero` pr `/dev/random`. Basic syntax:

```
$ dd if=input-file of=output-file options
```

If the input or output files are not specified, the default is to use `stdin` and `stdout`. Doing:

```
$ dd --help
```

will yield a really long list of options, some frequently used, and some only very rarely.

40.16 dd Examples

Some examples of using **dd**:

- Create a 10MB file filled with zeros:

```
$ dd if=/dev/zero of=outfile bs=1M count=10
```

- Backup an entire hard drive to another (raw copy):

```
$ dd if=/dev/sda of=/dev/sdb
```

- Create an image of a hard disk (which could later be transferred to another hard disk):

```
$ dd if=/dev/sda of=sdadisk.img
```

- Backup a partition:

```
$ dd if=/dev/sda1 of=partition1.img
```

- Use **dd** in a pipeline:

```
$ dd if=ndata conv=swab count=1024 | uniq > ofile
```

40.17 rsync

rsync (**r**emote **sync**hronize): used to transfer files across network (or between different locations on the same machine) as in:

```
$ rsync [options] source destination
```

Source and destination can take the form of `target:path` where `target` can be in the form of `[user@]host`. `user@` part is optional and used if the remote user is different from the local user. Thus, these are all possible **rsync** commands:

```
$ rsync file.tar someone@backup.mydomain:/usr/local
$ rsync -r a-machine:/usr/local b-machine:/usr/
$ rsync -r --dry-run /usr/local /BACKUP/usr
```

Have to be very careful with **rsync** about exact location specifications (especially if you use the `--delete` option), so it is highly recommended to use the `--dry-run` option first, and then repeat if the projected action looks correct.

rsync is very clever; checks local files against remote files in small chunks, and very efficient in that when copying one directory to a similar directory, only the differences are copied over the network. This synchronizes the second directory with the first directory. One often uses the `-r` option which causes **rsync** to recursively walk down directory tree copying all files and directories below the one lsited as the `sourcefile`. Thus, very useful way to back up project directory might be similar to:

```
$ rsync -r project-X archive-machine:archives/project-X
```

Simple (and very effective and very fast) backup strategy: simply duplicate directories or partitions across a network with **rsync** commands and do so frequently.

40.18 cpio

cpio (**c**o**p**y **i**n and **o**ut): general file archiver utility that has been around since the earliest days of UNIX, originally designed for tape backups. Even though newer archiving programs (like **tar**, which is not exactly young) have been deployed to do many of the tasks that were once in the domain of **cpio**, it still survives.

E.g., have already seen the use fo **rpm2cpio** to convert RPM packages into **cpio** archives and then extract them. Also, the Linux

kernel uses a version of **cpio** internally to deal with **initramfs** and **initrd** initial ram filesystems and disks during boot. One reason **cpio** lives on is that it is lighter than **tar** and other successors, even if it is somewhat less robust.

Examples of using **cpio**:

- Create an archive, use `-o` or `--create`:

```
$ ls | cpio --create -O /dev/st0
```

- Extract from an archive, use `-i` or `--extract`:

```
$ cpio --i somefile -I /dev/st0
```

- List contents of an archive, use `-t` or `--list`:

```
$ cpio -t -I /dev/st0
```

Can specify the input (`-I` device) or use redirection on the command line.

The `-o` or `--create` option tells **cpio** to copy files out to an archive. **cpio** reads a list of file names (one per line) from standard input and writes the archives to standard output.

The `-i` or `--extract` option tells **cpio** to copy files from an archive, reading the archive from standard input. If you list files names as patterns (such as `*.c`) on the command line, only files in the archive that match the patterns are copied from the archive. If no patterns give, all files extracted.

The `-t` or `--list` options tells **cpio** to list archive contents. Adding the `-v` or `--verbose` option generates long listing.

40.19 dump and restore

dump and **restore** utilities: been around since earliest days of UNIX, not originally designed for modern hardware, filesystems, storage capabilities.

Unlike **cpio** and **tar**, these utilities directly read and write filesystem, more efficient and allows backup files to be created without affecting time stamps.

Positive features:

- Can perform full or incremental backups
- Understand specific filesystem format and hot to read and write it
- Efficient when creating full backups, head movement reduced
- Can specify output tape size and density, block size and count, or even both
- Can dump to any valid device or file; defaults to `/dev/tape`
- Parameters in `/etc/fstab` control what gets dumped and when.

Negative features:

- Multiple filesystem passes are required for backups
- Only works with ext2, ext3, and ext4 filesystems. (Other filesystems may have their own utilities, such as **xfsdump**)
- Cannot be run safely on mounted filesystems.

Lack of filesystem flexibility: rather strong limitation, both because of multiplicity of filesystem tapes in use in Linux, and because

modern trend is to abstract details such as exactly how data is stored.

dump and **restore** sometimes used by higher-level backup program suites, such as Amanda. Some familiarity with these legacy tools is useful.

40.20 dump Options

dump has a number of options, including letting you set parameters. Some of these include:

- `-0-9`

  Dump level. Level 0 is full backup and higher numbers are incremental backups.

- `-B records`

  Records per volume.

- `-b blocksize`

  KB per record.

- `-f file`

  Output device or file.

- `-u`

  Update `/etc/dumpdates`

- `-w`
  Print most recent dump date of each filesystem in `/etc/dumpdates`.

(Note: some systems, such as Debian-based ones, may use `/var/lib/dumpdates` instead.)

Parameter values need not be placed immediately after the option, but must be given in the same order as the options that specify them.

40.21 restore

restore used to read archives, tapes, or files which were created by **dump**. E.g., to restore all files that were dumped, relative to current directory:

```
$ sudo restore -rvf /tmp/boot_backup
```

Useful options to **restore**:

- `-r`

  Restores everything. The dumped material is read and the complete contents are loaded into the current directory.

- `-t`

  Files and directories specified are listed on standard output if they reside on the backup. If no file argument is specified, the root directory on the backup is listed. No actual restore is performed with this option.

- `-x`

  The files and directories named are extracted from the backup. If the named file matches a directory on the backup, the directory is recursively extracted. If no argument is listed, then all contents of the backup are extracted.

- `-i`

  This mode allows the interactive restoration of files form the backup. After reading in the directory information from the backup, **restore** provides a shell-like interface that allows the user to move around the directory tree selecting files to be extracted.

You cannot use `-r` and `-s` at the same time.

40.22 mt

mt utility is used to control magnetic tape devices. Most often used to query or position the tape before or between backups and restores. By default, **mt** uses tape drive defined in `TAPE` environment variable. Can also use the `-f device` option to specify tape.

Note: only root user can use **mt**. Syntax:

```
mt [-h] [-f device] operation [count] [arguments...]
```

where:

- `-h` : is for displaying usage (help)
- `-f device` : is for specifying the tape device
- `operation` : is one of the tape operations
- `count` : is used for some operations which can repeat (default is 1)
- `arguments` : are used for some operations.

Some examples of using **mt**:

- Show satus information about the tape device:

```
$ mt status
```

- Rewind the tape:

```
$ mt rewind
```

- Erase the tape:

```
$ mt erase
```

- Move forward to the end of the current archive:

```
$ mt fsf
```

40.23 Backup Programs

No shortage of available backup program suites available for Linux, including proprietary applications for those supplied by storage vendors, as well as open-source applications. Several that are particularly well known:

1. **Amanda** (**A**dvanced **M**aryland **A**utomatic **N**etwork **D**isk **A**rchiver) uses native utilities (including **tar** and **dump**) but is far more robust and controllable. **Amanda** is generally available on Enterprise **Linux** systems through usual repositories. Complete information can be found on the Amanda website.
2. **Bacula** is designed for automatic backup on heterogeneous networks. Can be rather complicated to use and recommended (by its authors) only to experienced administrators. Generally available on Enterprise Linux systems through usual repositories. Complete information can be found at the Bacula website.
3. **Clonezilla** is a very robust disk cloning program, which can make images of disks and deploy them, either to restore a backup, or to be used for ghosting, to provide an image that can be used to install many machines. Complete information can be found at the Clonezilla website.
   Program comes in two versions: Clonezilla live, which is good for single machine backup and recovery, and Clonezilla SE, server **e**dition, which can clone to many computers at the same time. Clonezilla not very hard to use, and extremely flexible, supporting many operating systems (not just Linux), filesystem types, and boot loaders.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 41 Linux Security Modules - Notes

41.3 Learning Objectives:

- Understand how the Linux Security Module framework works and how it is deployed.
- List the various LSM implementations available.
- Delineate the main features of SELinux.
- Explain the different modes and policies available.
- Grasp the importance of contexts and how to get and set them.
- Know how to use the important SELinux utility programs.
- Gain some familiarity with AppArmor.

41.4 What Are Linux Security Modules?

A modern computer system must be made secure, but needs vary according to sensitivity of data, number of users with accounts, exposure to outside networks, legal requirements, and other factors. Resposibility for enabling good security controls falls both on application designers and the Linux kernel developers and maintainers. Users should have to follow good procedures as well, but on a well run system, non-privileged users should have very limited ability to expose system to security violations. In this section, concerned with how Linux kernel enhances security through use of Linux Security Modules framework, particularly with deployment of SELinux. Idea is to implement **mandatory access controls** over variety of requests made to kernel, but to do so in a way that:

1. Minimizes changes to the kernel
2. Minimizes overhead on the kernel
3. Permits flexibility and choice between different implementations, each of which is presented as a self-contained **LSM** (**L**inux **S**ecurity **M**odule).

Basic idea to **hook system calls**; insert code wherever an application requests transition to kernel (system) mode in order to accomplish work that requires enhanced abilities; this code makes sure permissions are valid, malicious intent is protected against, etc. Does this by invoking security-related functional steps before and/or after a system call is fulfilled by kernel.

41.5 LSM Choices

For a long time, only enhanced security model implemented was **SELinux**. When project was first floated upstream in 2001 to be included directly in kernel, there were objections about using only one approach to enhanced security.

As a result, LSM approach was adopted, where alternative modules to SELinux could be used as they were developed and was incorporated into Linux kernel in 2003.

Current LSM implementations:

- SELinux
- AppArmor
- Smack
- Tomoyo

Only one LSM can be used at a time, as they potentially modify the same parts of the Linux kernel.

Will concentrate primarily on SELinux and secondarily on AppArmor in order of usage volume.

41.6 SELinux Overview

SELinux originally developed by United States NSA (**N**ational **S**ecurity **A**dministration)
and has been integral in RHEL for a very long time, which has brought it a large usage base.

Operationally, SELinux is a set of security rules that are used to determine which processes can access which files, directories, ports, and other items on the system.

Works with three conceptual quantities:

1. **Contexts**: Are labels to files, processes, and ports. Examples of contexts are SELinux user, role, and type.

2. **Rules**: Describe access control in terms of contexts, processes, files, ports, users, etc.

3. **Policies**: Are a set of **rules** that describe what system-wide access control decisions should be made by SELinux.

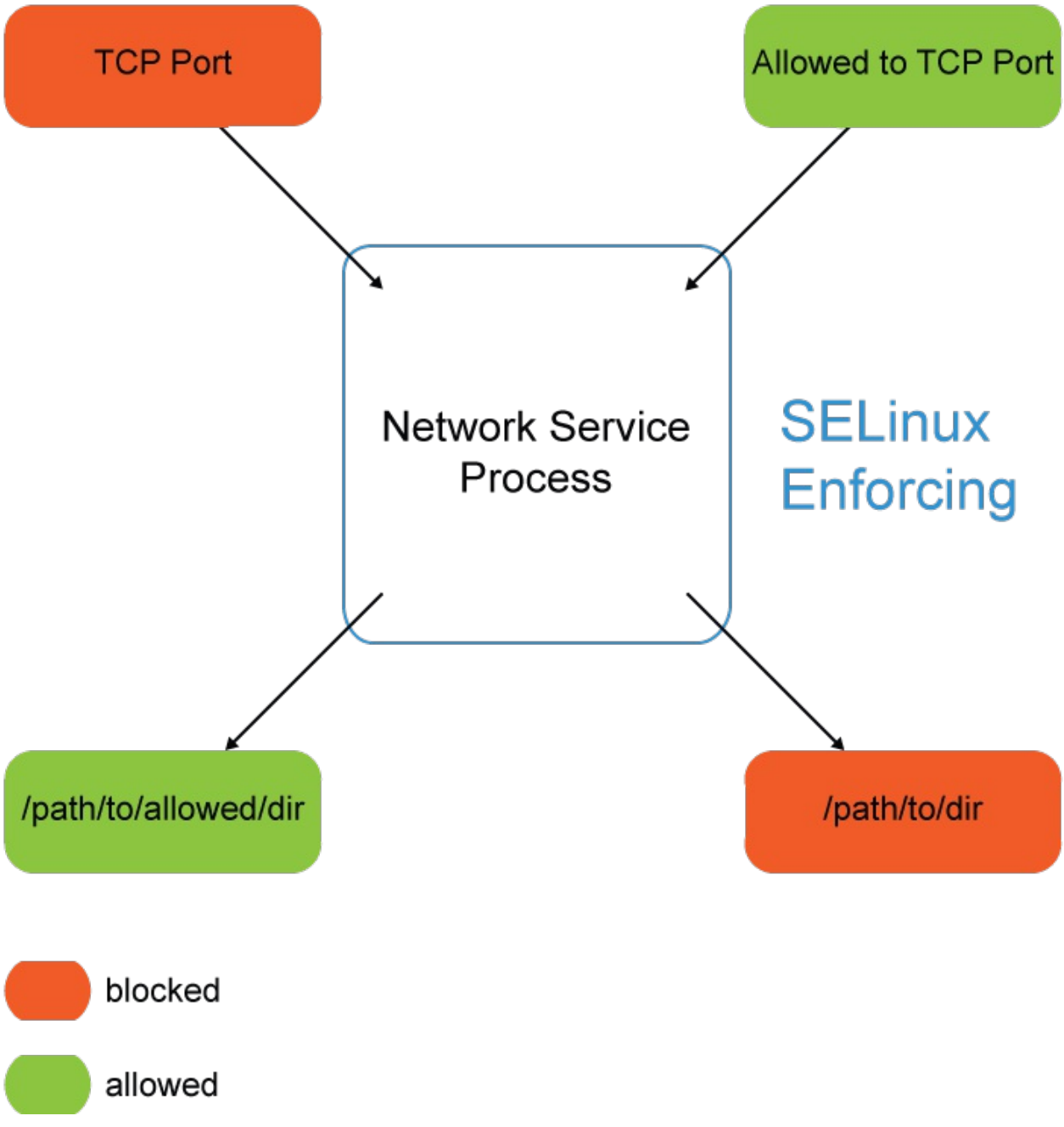
   A SELinux context is a name used by a rule to define how users, processes, files, and ports interact with each other. As the default policy is to deny any access, rules are used to describe allowed actions on the system.

41.7 SELinux Modes

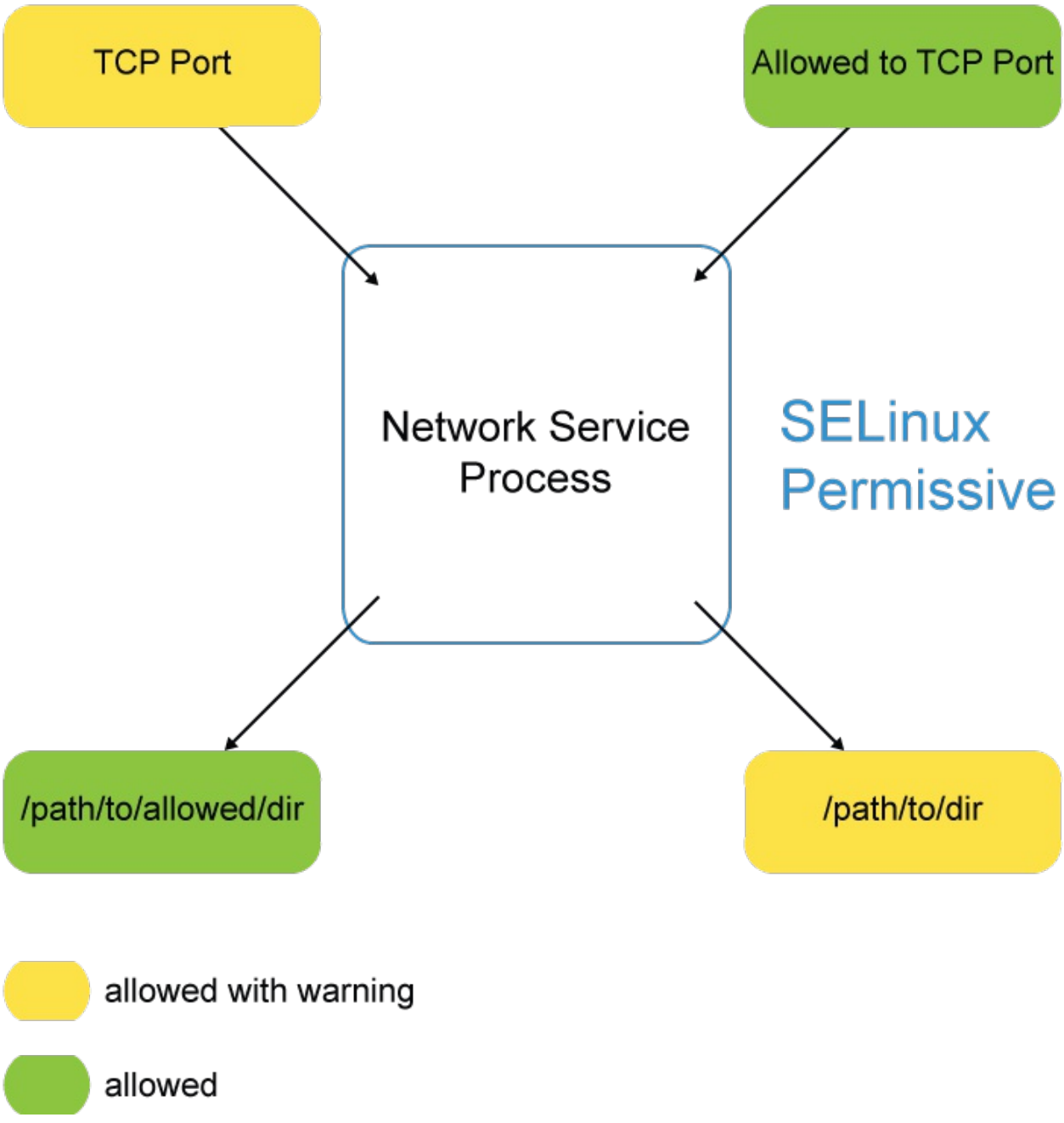
SELinux can be run under one of the three following modes:

- **Enforcing**: All SELinux code is operative and access is denied according to policy. All violations are audited and logged.
- **Permissive**: Enables SELinux code, but only audits and warns about operations that would be denied in enforcing mode.
- **Disabled**: Completely disables SELinux kernel and application code, leaving the system without any of its protections.

These modes are selected (and explained) in a file (usually `/etc/selinux/config`) whose location varies by distribution (it is often either at `/etc/sysconfig/selinux` or linked from there). The file is well self-documented. The **sestatus** utility can display the current mode and policy.



SELinux Enforcing Mode



SELinux Permissive Mode

To examine or set current mode, one can use **getenforce** and **setenforce**:

```
$ getenforce
Disabled
$ sudo setenforce Permissive
$ getenforce
Permissive
```

setenforce can be used to switch between **enforcing** and **permissive** modes on the fly while system is in operation. However, changing in or out of the **disabled** mode cannot be done this way. While **setenforce** allows you to switch between **permissive** and **enforcing** modes, it does not allow you to disable SELinux completely. There are at least two different ways to disable SELinux:

- **Configuration file**

Edit the SELinux configuration file (usually `/etc/selinux/config`) and set `SELINUX=disabled`. This is the default method and should be used to permanently disable SELinux.

- **Kernel parameter**

  Add `selinux=0` to the kernel parameter list when rebooting.

However, important to note that disabling SELinux on systems in which SELinux will be re-enabled is not recommended. Preferable to use the **permissive** mode instead of disabling SELinux, so as to avoid relabeling the entire filesystem, which can be time-consuming.

41.8 SELinux Policies

The same configuration file, usually `/etc/sysconfig/selinux`, also sets the **SELinux policy**. Multiple policies are allowed, but only one can be active at a time. Changing the policy may require a reboot of the system and a time-consuming re-labeling of filesystem contents. Each policy has files which must be installed under `/etc/selinux/[SELINUXTYPE]`.

Most common policies:

- **targeted**

  The **default** policy in which SELinux is more restricted to targeted processes. User processes and **init** processes are not targeted. SELinux enforces memory restrictions for **all** processes, which reduces the vulnerability to buffer overflow attacks.

- **minimum**

  A modification of the targeted policy where only selected processes are protected.

- **MLS**

  The Multi-Level Security policy is much more restrictive; all processes are placed in fine-grained security domains with particular policies.

41.9 Context Utilities

As mentioned earlier, contexts are labels applied to files, directories, ports, and processes. Those labels are used to describe access rules. There are four SELinux contexts:

- **User**
- **Role**
- **Type**
- **Level**.

However, will focus on **type**, which is most commonly utilized context. Label naming convention determines that type context labels should end with `_t` as in `kernel_t`.

Use `-Z` option to see context:

```
$ ls -Z
$ ps auZ
```

Use **chcon** command to change context:

```
$ chcon -t etc_t somefile
$ chcon --reference somefile so
```

41.10 SELinux and Standard Command Line Tools]

Many standard command line commands, such as **ls** and **ps**, were extended to support SELinux, and corresponding sections were added to their **man** pages explaining the details. Often the parameter `z` is passed to standard command line tools as in:

```
$ ps axZ
LABEL PID TTY STAT TIME COMMAND
system_u:system_r:init_t:s0 1 ? Ss 0:04 /usr/lib/systemd/systemd --switched-root
...
system_u:system_r:kernel_t:s0 2 ? S 0:00 [kthreadd]
...
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2305 ? D 0:00 sshd:
peter@pts/0
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2306 pts/0 Ss 0:00 -bash
...
system_u:system_r:httpd_t:s0 7490 ? Ss 0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0 7491 ? S 0:00 /usr/sbin/httpd -DFOREGROUND
...
$ ls -Z /home/ /tmp/
/home/:
drwx------. peter peter unconfined_u:object_r:user_home_dir_t:s0 peter
/tmp/:
-rwx------. root root system_u:object_r:initrc_tmp_t:s0 ks-script-c4ENhg
drwx------. root root system_u:object_r:tmp_t:s0 systemd-private-0ofSvO
-rw-------. root root system_u:object_r:initrc_tmp_t:s0 yum.log
```

Other tools that were extended to support SELinux include **cp**, **mv**, and **mkdir**.

Note: if you have disabled SELinux, no useful information is displayed in the related fields from these utilities.

41.11 SELinux Context Inheritance and Preservation

Newly created files inherit the context from their parent directory, but when moving files, it is the context of the source directory which may be preserved, which can cause problems.

Continuing the previous example, can see the context of `tmpfile` not changed by moving the file from `/tmp` to `/home/peter`:

```
$ cd /tmp/
$ touch tmpfile
$ ls -Z tmpfile
-rw-rw-r--. peter peter unconfined_u:object_r:user_tmp_t:s0 tmpfile
$ cd
$ touch homefile
$ ls -Z homefile
-rw-rw-r--. peter peter unconfined_u:object_r:user_home_t:s0 homefile
$ mv /tmp/tmpfile .
$ ls -Z
-rw-rw-r--. peter peter unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. peter peter unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

Classical example in which moving files creates a SELinux issue is moving files to the `DocumentRoot` directory of the **httpd** server. On SELinux-enabled systems, the web server can only access files with the correct context labels. Creating a file in `/tmp`, and then moving it to the `DocumentRoot` directory, will make the file unaccessible to the **httpd** server until the SELinux context of the file is adjusted.

41.12 restorecon

restorecon resets files contexts, based on parent directory settings. In the following example, **restorecon** resets the default label recursively for all files at the home directory:

```
$ ls -Z
-rw-rw-r--. peter peter unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. peter peter unconfined_u:object_r:user_tmp_t:s0 tmpfile
$ restorecon -Rv /home/peter
restorecon reset /home/peter/tmpfile context \
unconfined_u:object_r:user_tmp_t:s0->unconfined_u:object_r:user_home_t:s0
$ ls -Z
-rw-rw-r--. peter peter unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. peter peter unconfined_u:object_r:user_home_t:s0 tmpfile
```

Note: context for `tmpfile` has been reset to the default context for files created at the home directory. Type was changed from `user_tmp_t` to `user_home_t`.

41.13 semanage

Another issue is how to configure the default context for a newly created directory. **semanage fcontext** (provided by the **policycoreutils-python** package) can change and display the default context of files and directories. Note: **semanage fcontext** only changes the default settings; does not apply them to existing objects. This requires calling **restorecon** afterwards. For example:

```
[root@rhel7 /]# mkdir /virtualHosts
[root@rhel7 /]# ls -Z
...
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts
[root@rhel7 /]# semanage fcontext -a -t httpd_sys_content_t /virtualHosts
[root@rhel7 /]# ls -Z
...
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts
[root@rhel7 /]# restorecon -RFv /virtualHosts
restorecon reset /virtualHosts context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
[root@rhel7 /]# ls -Z
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 virtualHosts
```

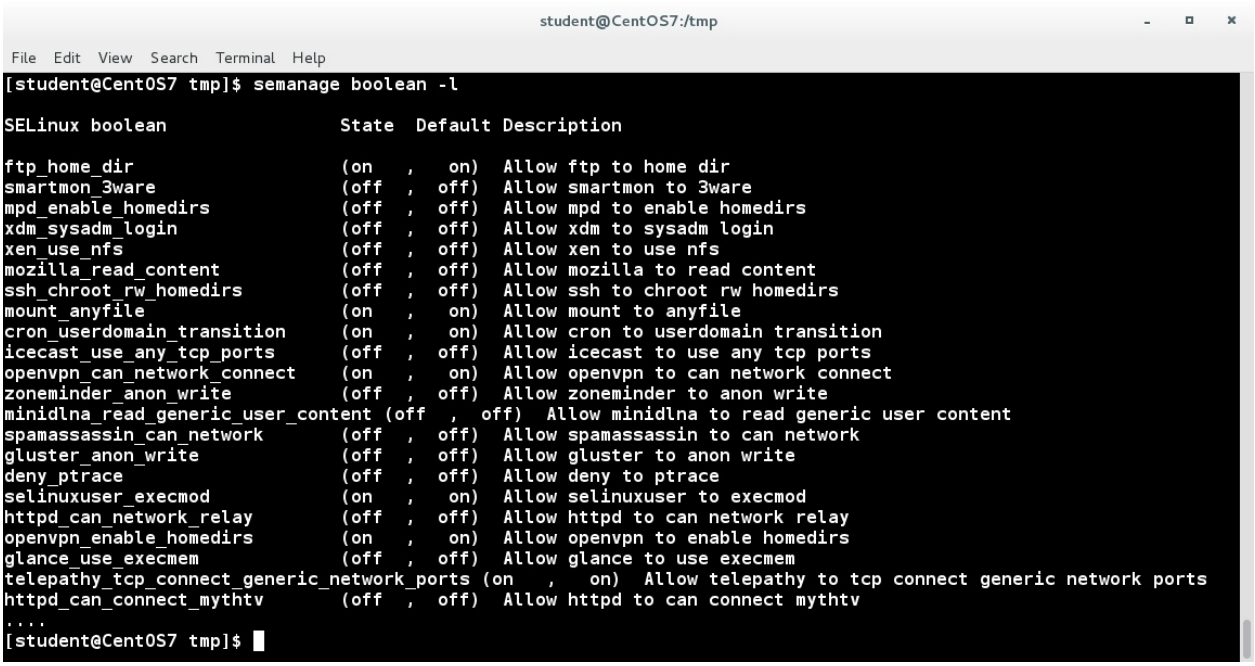
The context change from `default_t` to `httpd_sys_content_t` is thus only applied after the call to **restorecon**.

41.14 Using SELinux Booleans

SELinux policy behavior can be configured at runtime without rewriting policy. Accomplished by configuring SELinux Booleans, which are policy paramaters that can be enabled and disabled:

- **getsebool** - to see booleans
- **setsebool** - to set booleans
- **semanage booleans -i** - to see persistent boolean settings.

Can see what needs to be done to list all booleans of current policy, including current status and short description, below.



41.15 getsebool and setsebool

Alternative for displaying boolean information with simpler output: **getsebool -a**, which prints only the boolean name and its current status.

setsebool used for changing boolean status. Default behavior is to apply changes immediately that are not persistent across a reboot. However, `-P` parameter can be supplied in order to make changes persistent.

An example of non-persistent change using **setsebool**:

```
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> off
$ sudo setsebool ssh_chroot_rw_homedirs on
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> on
$ sudo reboot
...
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> off
```

An example of persistent change using **setsebool -P**:

```
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> off
$ sudo setsebool -P ssh_chroot_rw_homedirs on
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> on
$ sudo reboot
...
$ getsebool ssh_chroot_rw_homedirs
ssh_chroot_rw_homedirs --> on
```

41.16 Troubleshooting Tools

SELinux comes with a set of tools that collect issues at run time, log these issues, and propose solutions to prevent same issues from happening again. These utilities are provided by the **setroubleshoot-server** package. An example of their use:

```
[root@rhel7 ~]# echo 'File created at /root' > rootfile
[root@rhel7 ~]# mv rootfile /var/www/html/
[root@rhel7 ~]# wget -O - localhost/rootfile
--2014-11-21 13:42:04--  http://localhost/rootfile
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2014-11-21 13:42:04 ERROR 403: Forbidden.
[root@rhel7 ~]# tail /var/log/messages
Nov 21 13:42:04 rhel7 setroubleshoot: Plugin Exception restorecon
Nov 21 13:42:04 rhel7 setroubleshoot: SELinux is preventing /usr/sbin/httpd from getattr access on the file
.
For complete SELinux messages. run sealert -l d51d34f9-91d5-4219-ad1e-5531e61a2dc3
Nov 21 13:42:04 rhel7 python: SELinux is preventing /usr/sbin/httpd from getattr access on the file .
*****  Plugin catchall (100. confidence) suggests  **************************
If you believe that httpd should be allowed getattr access on the  file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing
# grep httpd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
[root@rhel7 ~]#  sealert -l d51d34f9-91d5-4219-ad1e-5531e61a2dc3
SELinux is preventing /usr/sbin/httpd from getattr access on the file .
*****  Plugin catchall (100. confidence) suggests  **************************
If you believe that httpd should be allowed getattr access on the  file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# grep httpd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
Additional Information:
Source Context system_u:system_r:httpd_t:s0
Target Context unconfined_u:object_r:admin_home_t:s0
Target Objects [ file ]
Source httpd
Source Path /usr/sbin/httpd
Port <Unknown>
Host rhel7
Source RPM Packages httpd-2.4.6-18.el7_0.x86_64
Target RPM Packages
Policy RPM selinux-policy-3.12.1-153.el7_0.11.noarch
Selinux Enabled True
Policy Type targeted
Enforcing Mode Enforcing
Host Name rhel7
```

```
Platform Linux rhel7 3.10.0-123.9.3.el7.x86_64 #1 SMP Thu
Oct 30 00:16:40 EDT 2014 x86_64 x86_64
Alert Count 2
First Seen 2014-11-21 12:34:13 CET
Last Seen 2014-11-21 13:42:04 CET
Local ID d51d34f9-91d5-4219-ad1e-5531e61a2dc3

Raw Audit Messages
type=AVC msg=audit(1416573724.395:1598): avc: denied { getattr } for pid=20180 comm="httpd"
path="/var/www/html/rootfile" dev="dm-0" ino=70624441 scontext=system_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file

type=SYSCALL msg=audit(1416573724.395:1598): arch=x86_64 syscall=lstat success=no exit=EACCES
a0=7f2896ed0578 a1=7fffcc64fb30 a2=7fffcc64fb30 a3=0 items=0 ppid=20178 pid=20180
auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none)
ses=4294967295 comm=httpd exe=/usr/sbin/httpd subj=system_u:system_r:httpd_t:s0 key=(null)

Hash: httpd,httpd_t,admin_home_t,file,getattr
```

Note: on RHEL 7, the suggestion is to run:

```
$ grep httpd /var/log/audit/audit.log |  audit2allow -M mypol
```

audit2allow: a tool that generates SELinux policy rules from logs of denied operations. **audit2why**: a similar tool, which translates SELinux audit messages into a description of why the access was denied.

Next example shows how to solve this issue using the **restorecon** tool which was described earlier. Feel free to try both approaches for fixing the SELinux issue.

```
[root@rhel7 ~]# restorecon -Rv /var/www/html/
restorecon reset rootfile context unconfined_u:object_r:admin_home_t:s0-
>unconfined_u:object_r:httpd_sys_content_t:s0

[root@rhel7 ~]# wget -q -O - localhost/rootfile
File created at /root
```

41.17 Additional Online Resources

This section has covered the basics and most common system administration tasks related to SELinux. There are freely available online resources for SELinux advanced topics, including:

- SELinux User's and Administrator's Guide
- Red Hat Enterprise Linux 6 Security-Enhanced Linux

41.18 AppArmor

AppArmor is an LSM alternative to SELinux. Support for it has been incorporated in the Linux kernel since 2006. It has been used by SUSE, Ubuntu, and other distributions.

AppArmor:

- Provides Mandatory Access Control (MAC)
- Allows administrators to associate a security profile to a program which restricts its capabilities
- Is considered easier (by some but not all) to use than SELinux
- Is considered filesystem-neutral (no security labels required).

AppArmor supplements the traditional UNIX Discretionary Access Control (DAC) model by providing Mandatory Access Control (MAC).

In addition to manually specifying profiles, AppArmor includes a learning mode, in which violations of the profile are logged, but

not prevented. This log can then be turned into a profile, based on the program's typical behavior.

41.19 Checking Status

Distributions that come with AppArmor tend to enable it and load it by default. Note: Linux kernel has to have it turned on as well, and, in most cases, only one LSM can run at a time.

Assuming you have the AppArmor kernel module available, on a systemd-equipped system you can do:

```
$ sudo systemctl [start|stop|restart|status] apparmor
```

to change or inquire about the current state of operation, or do:

```
$ sudo systemctl [enable|disable] apparmor
```

to cause to be loaded or not loaded at boot.

In order to see the current status, do:

```
$ sudo apparmor_status
apparmor module is loaded.
25 profiles are loaded.
25 profiles are in enforce mode.
    /sbin/dhclient
...
```

Profiles and **processes** are in either **enforce** or **complain** mode, directly analogous to SELinux's **enforcing** and **permissive** modes. Note that in the process, listing the PID is given:

```
$ ps aux | grep libvirtd
root      787  0.0  0.9  527200  35936 ?       Ssl  10:54  0:00  /usr/sbin/libvirtd
student  3346  0.0  0.0   13696   2204 pts/16  S+   11:42  0:00  grep --color=auto libvirtd
```

41.20 Modes and Profiles

Profiles restrict how executable programs, which have pathnames on your system, such as `/usr/bin/evince`, can be used.

Processes can be run in either of the two modes:

- **Enforce Mode**

  Applications are prevented from acting in ways which are restricted. Attempted violations are reported to the system logging files. This is the default mode. A profie can be set to this mode with **aa-enforce**.

- **Complain Mode**

  Policies are not enforced, but attempted policy violations are reported. This is also called the learning mode. A profile can be set to this mode with **aa-complain**

Linux distributions come with pre-packaged profiles, typically installed either when a given package is installed, or with an AppArmor package, such as **apparmor-profiles**. These profiles are stored in `/etc/apparmor.d`.

When installing new software, new profiles can be created specific to any executables in the package.

Exactly what AppArmor profiles are installed on your system depends on your selection of software packages. For example, on one particular Ubuntu system:

```
student@ubuntu: /etc/apparmor.d$ ls
abstractions     usr.lib.dovecot.anvil      usr.lib.telepathy
apache2.d        usr.lib.dovecot.auth       usr.sbin.avahi-daemon
bin.ping         usr.lib.dovecot.config     usr.sbin.cups-brows
...
```

Full documentation on what can go in these files can be obtained by doing **man apparmor.d**.

41.21 AppArmor Utilities

AppArmor has quite a few administrative utilities for monitoring and control. For example, on an OpenSUSE system:

```
$ rpm -qil apparmor-utils | grep bin
/usr/bin/aa-easyprof
/usr/sbin/aa-audit
/usr/sbin/aa-autodep
/usr/sbin/aa-cleanprof
/usr/sbin/aa-complain
/usr/sbin/aa-decode
/usr/sbin/aa-disable
/usr/sbin/aa-enforce
/usr/sbin/aa-exec
....
/usr/sbin/complain
/usr/sbin/decode
/usr/sbin/disable
/usr/sbin/enforce
....
```

Note: many of these utilities can be invoked with either their short or long names:

```
linux-llgn:/etc/apparmor.d # ls -l /usr/sbin/*complain
-rwxr-xr-x 1 root root 1442 Oct 25 07:37 /usr/sbin/aa-complain*
lrwxrwxrwx 1 root root   11 Nov 11 13:02 /usr/sbin/complain -> aa-complain*
linux-llgn:/etc/apparmor.d #
```

AppArmor Utilities

Program	Use
`apparmor_status`	Show status of all profiles and processes with profiles
`apparmor_notify`	Show a summary for AppArmor log messages
`complain`	Set a specified profile to complain mode
`enforce`	Set a specified profile to enforce mode
`disable`	Unload a specified profile from the current kernel and prevent from being loaded on system startup

`logprof`	Scan log files, and, if AppArmor events that are not covered by existing profiles have been recorded, suggest how to take into account, and, if approved, modify and reload
`easyprof`	Help setup a basic AppArmor profile for a program

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 42 Local System Security - Notes

42.2 Introduction

Essential task of any system administrator is to secure the system(s) against both internal and external threats. Work begins with designing proper security policy, crafted to defend against expected and unexpected threat vectors. In addition, good system hygiene has to be practiced; systems need to be maintained and upgraded in timely fashion, as well as physically protected from usurpers. Furthermore, sensible policies have to guarantee only appropriate users have potentially dangerous privileges, and only those which are absolutely needed.

42.3 Learning Objectives:

- Assess system security risks.
- Fashion and implement sound computer security policies and procedures.
- Efficiently protect BIOS and the boot loader with passwords.
- Use appropriate **mount** options, **setuid** and **setgid** to enhance security.

42.4 Local System Security

Computers are inherently insecure and need protection from people who would intrude in on or attack them. This can be done to simply harm the system, deny services, or steal information.

No computer can ever be absolutely secure. All we can do is slow down and/or discourage intruders so that they either go away and hunt for easier targets, or so we can catch them in the act and take appropriate action.

Security can be defined in terms of the system's ability to regularly do what it is supposed to do, integrity and correctness of the system, and ensuring that the system is only available to those authorized to use it.

The biggest problem with security is to find that appropriate mix of security and productivity; if security restrictions are tight, opaque, and difficult, especially with ineffective measures, users will circumvent procedures.

The four areas we need to protect include physical, local, remote, and personnel. In this section, we will not concentrate on network security, we will concentrate on local factors.

42.5 Creating a Security Policy

Important to create and publicize to you organization a clear security policy. It should:

- Be simple and easy to understand
- Get constantly updated
- Be in the form or a written document in addition to online documentation if needed
- Describe both policies and procedures
- Specify enforcement actions
- Specify actions to take in response to a security breach.

Policies should be generic and not hard to grasp as that makes them easier to follow. Must safeguard the data that needs protection, deny access to required services and protect user privacy.

These policies should be updated on a regular basis; policies need to change as requirements do. Having an out of date policy can be worse than having none.

42.6 What to Include in the Policy

A security policy should include methods of protecting information from being read or copied by unauthorized personnel. Should also include protection of information from being altered or deleted without the permission of the owner. All services should be protected so they are available and not degraded in any manner without authorization.

Essential aspects to cover:

- Confidentiality
- Data Integrity
- Availability
- Consistency
- Control
- Audit

Should make sure that data is correct and the system behaves as it is expected to do. There should be processes in effect to determine who is given access to your system. Human factor is the weakest link in the security chain; it required the most attention through constant auditing.

42.7 What Risks to Assess

Risk analysis is based on the following three questions:

- What do I want to protect?
- What am I protecting against?
- How much time, personnel, and money is needed to provide adequate protection?

Must know what you are protecting and what are you protecting against in order to determine how to protect your systems. This allows you to then plan policies and procedures to protect your systems.

This is the first step taken in constructing a computer security policy. It is a pre-requisite for planning and then enforcing policies and procedures to protect your systems.

42.8 Choosing a Security Philosophy

Two basic philosophies found in use in most computing environments:

- Anything not expressly permitted is denied.
- Anything not expressly forbidden is permitted.

You should decide which philosophy is best for your organization.

First choice is tighter: user is allowed to do only what is clearly and explicitly specified as permissible without privilege. This is the most commonly used philosophy.

Second choice builds a more liberal environment where users are allowed to do anything except what is expressly forbidden. Implies a high degree of assumed trust and is less often deployed for obvious reasons.

42.9 Some General Security Guidelines

Some general guidelines to remember when developing security philosphies:

1. **The human factor is the weakest link**

   You must educate your users and keep them happy. The largest percentage of break-ins are internal and are not often malicious.

2. **No computing environment is invulnerable**

   The only secure system is one that is not connected to anything, locked in a secure room, and turned off.

3. **Paranoia is a good thing**

   Be suspicious, be vigilant, and persevere when securing a computer. It is an ongoing process which must be constantly paid attention to. Check process, users, and look for anything out of the ordinary.

Users should never put the current directory in their path, i.e., do not do something in `~/.bashrc` like `Path=./:$PATH`

This is a significant security risk; a malicious person could substitute for a program with one of the same name that could do harmful things. Just think of a script names **ls** that contains just the line:

```
/bin/rm -rf $HOME
```

If you were to go to the directory that contains this file and type **ls**, you would wipe out your home directory!

42.10 Updates and Security

It is crucial to pay attention to your Linux distributor's updates and upgrades and apply them as soon as possible.

Any system which is not fully updated should be considered vulnerable.

Most attacks exploit known security holes and are deployed in the time period between revelation of a program and patches being applied. **Zero Day** exploits are actually much rarer, where an attacked uses a security hole that either has not been discovered yet or for which a fix has not been released.

System administrators are sometimes reluctant to apply such fixes immediately upon release, based on negative experiences with proprietary operating system vendors who can cause more problems with fixes than they solve. However, in Linux such security **regressions** are extremely rare, and the danger of delaying applying a security patch is probably never justifiable.

42.11 Hardware Accessibility and Vulnerability

Any time hardware is physically accessible security can be compromised by:

- Key logging: Recording the real time activity of a computer user including the keys they press. The captured data can either be stored locally or transmitted to remote machines.
- Network sniffing: Capturing and viewing the network packet level data on your network.
- Booting with a live or rescue disk.
- Remounting and modifying disk content.

Physical access to a system makes is possible for attackers to easily leverage several attack vectors, in a way that makes all operating system level recommendations irrelevant.

Thus, security policy should start with requirements on how to properly physical access to servers and workstations.

42.12 Hardware Access Guidelines

Necessary steps include:

- Locking down workstations and servers
- Protecting your network links against access by people you do not trust
- Protecting your keyboards where passwords are entered to ensure the keyboards cannot be tampered with
- Configuring password protection of the BIOS in such a way that the system cannot be booted with a live or rescue CD/DVD or USB key.

For single user computers and those in a home environment, some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it should not be there, or it should be better protected by following the above guidelines.

42.13 Protection of BIOS

BIOS is the lowest level of software that configures or manipulates your system. The boot loader access the BIOS to determine how to boot up the machine. The BIOS:

- Is the lowest level of security
- Should be protected by use of a password
- Should be updated and current.

Setting a BIOS password protects against unauthorized persons changing the boot options to gain access to your system. However, only matters if someone can gain physical access to machine, as it requires a local presence.

Also, generally recommended that BIOS be kept patched to the latest version of firmware. However, most BIOS updates have nothing to do with security, and system administrators have also been instructed to apply new BIOS only with care, as incompetent BIOS code has always been a plague, and unnecessary updates can render a system useless.

42.14 Protecting the Boot Loader with Passwords

Can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. This can work in conjunction with password protection for the BIOS.

Note: using a bootloader password alone will stop user from editing the bootloder configuration during boot process. Will not prevent a user from booting from an alternative boot media such as optical disks or pendrives. Thus, should be used with a BIOS password for full protection.

For older GRUB version 1, relatively easy to set password for **grub**, but for dominant GRUB version 2, things more complicated. However, have more flexibility and can do things like setting individual user-specific passwords (which can be the normal login ones).

Once again, should not edit `grub.cfg` directly. Rather, edit system configuration files in `/etc/grub.d` and then run **update-grub** or **grub2-mkconfig** and save the new configuration file.

One explanation of this can be found on the Grub2/Passwords webpage at Ubuntu.

42.15 Using Secure Mounting Options

When filesystem mounted, either at the command line with a **mount** command, or automatically by inclusion in `/etc/fstab`, various options can be specified to enhance security:

- `nodev`

  Do not interpret character or block special devices on the filesystem.

- `nosuid`

  The **set-user-identifier** or **set-group-identifier** bit are not to take effect (Will shortly discuss **setuid** and **setgid**)

- `noexec`

  Restrict direct execution of any binaries on the mounted filesystem.

- `ro`

  Mount filesystem in **read-only** mode as in:

```
$ mount -o ro,noexec,nodev, /dev/sda2 /mymountpt
```

or on `/etc/fstab`:

```
/dev/sda2 /mymountpt ext4 ro,noexec,nodev 0 0
```

42.16 setuid and setgid

Normally, programs run with privileges of user who is executing the program. This means that no matter who actually owns the binary executable that is running, the process still has constricted privileges.

Occasionally, may make sense to have normal users have expanded capabilities they would not normally have, such as ability to start or stop a network interface, or edit a file owned by superuser.

By setting the **setuid** (**set u**ser **ID**) flag on an executable file, one modified this normal behavior by giving the program the access rights of the **owner** rather than the **user** of the program.

Furthermore, one can also set the **setgid** bit for the process runs with the privileges of the group that owns the file rather than those of the one running it.

Should emphasize that this is generally a **bad idea** and is to be avoided in most circumstances. Often better to write a **daemon** program with lesser privileges for this kind of use. Some recent distributions have actually disabled this ability entirely.

By default, when a file is created in a directory, it is owned by the user and group of the user that created it. Using the **setgid** setting on the directory changes this so that files created in the directory are group owned by the group owner of the directory. Allows you to created a shared directory in which a group of users can share files.

42.17 Setting the setuid/setgid Bits

Simply done by:

```
$ chmod u+s somefile
$ chmod g+s somefile
```

where fist example does **setuid** and second the **setgid** operation.

For directories, setting group bits has very different effect; used to create a shared directory, as in:

```
$ chmod g+s somedir
```

Files created in this directory are group owned by the group owner of the directory.

Note: cannot effectively change the **setuid** on a shell script file; in fact, won't do anything unless you actually change the **setuid** bit *on the shell*, which would be a terrible security hole. Can only do this on executable binary programs.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter

Chapter 43 Basic Troubleshooting - Notes

43.3 Learning Objectives:

- Troubleshoot your system, following a number of steps iteratively until solutions are found.
- Check your network and file integrity for possible issues.
- Resolve problems when there is system boot failure.
- Repair and recover corrupted filesystems.
- Understand how **rescue and recovery** media can be used for troubleshooting.

43.4 Troubleshooting Levels

There are three levels of troubleshooting:

1. **Beginner**: Can be taught very quickly.
2. **Experienced**: Comes after a few years of practice.
3. **Wizard**: Some people think you have to be born this way, but that is nonsense; all skills can be learned. Every organization should have at least one person at this level of expertise to use as the go-to person.

Even the best administered systems will develop problems. Troubleshooting can isolate whether the problems arise from software or hardware, as well as whether they are local to the system, or come from within the local network or the Internet.

Troubleshooting properly requires judgment and experience, and while it will always be somewhat of an art form, following good methodical procedures can really help isolate the sources of problems in reproducible fashion.

43.5 Basic Techniques

Troubleshooting involves taking number of steps which need to be repeated iteratively until solutions are found. Basic recipe:

1. Characterize the problem
2. Reproduce the problem
3. Always try the easy things first
4. Eliminate possible causes one at a time
5. Change only one thing at a time; if that doesn't fix the problem, change it back
6. Check the system logs (`/var/log/messages` , `/car/log/secure` , etc.) for further information.

43.6 Intuition and Experience

Sometimes, ruling philosophy and methodology requires following very established procedure; making leaps based on intuition discouraged. Motivation for using a checklist and uniform procedure is to avoid reliance on a wizard, to ensure any system administrator will be able to eventually solve a problem if they adhere to well known procedures. Otherwise, if the wizard leaves the organization, there is no one skilled enough to solve tough problems.

If, on the other hand, you elect to respect your intuition and check hunches, should make sure you can get sufficient data quickly enough to decide whether or not to continue or abandon an intuitive path, based on whether it looks like it will be productive.

While ignoring intuition can sometimes make solving a problem take longer, troubleshooter's previous track record is critical benchmark for evaluating whether to invest resources this way. In other word's, useful intuition is not magic, it is distilled

experience.

43.7 Things to Check: Networking

Following items need to be checked when there are issues with networking:

- **IP configuration**

  Use **ifconfig** or **ip** to see if the interface is up, and if so, if it is configured.

- **Network Driver**

  If the interface can't be brought up, maybe the correct device driver for the network card(s) is not loaded. Check with **lsmod** if the network driver is loaded as a kernel module, or by examining relevant pseudo-files in `/proc` and `/sys` such as `/proc/interrupts` or `/sys/class/net`.

- **Connectivity**

  Use **ping** to see if the network is visible, checking for response time and packet loss. **traceroute** can follow packets through the network, while **mtr** can do this in a continuous fashion. Use of these utilities can tell you if the problem is local or on the Internet.

- **Default gateway and routing configuration**

  Run **route -n** and see if the routing table makes sense.

- **Hostname resolution**

  Run **dig** or **host** on a URL and see if DNS is working properly.

Network problems can be cause either by software or hardware, and can be as simple as is the device driver loaded, or is the network cable connected. If the network is up and running but performance is terrible, it really falls under the banner of performance tuning, not troubleshooting. The problems may be external to the machine, or require adjustment of the various networking parameters including buffer sizes, etc.

43.8 Things to Check: File Integrity

There are a number of ways to check for corrupt configuration files and binaries. Packaging systems have methods of verifying file integrity and checking for changes, as discussed earlier in this course. For RPM-based systems:

```
$ rpm -V some_package
```

checks a single package, while

```
$ rpm -Va
```

checks all packages on the system.

On Debian-based systems, one can do integrity checking with:

```
$ debsums options some_package
```

which will check the checksums on the files in that package. However, not all packages maintain checksums, so this might not be completely useful. One can also take advantage of the `-V` or `--verify` options on recent versions of **dpkg**.

aide does **intrusion detection** and is another way to check for changes in files:

```
$ sudo aide --check
```

will run a scan on your files and compare them to the last scan. Will have to maintain the **aide** database after initializing it, of course.

43.9 Boot Process Failures

If system fails to boot properly or fully, being familiar with what happens at each stage is important in identifying particular sources of problems. Assuming you get through BIOS stage, may reach one of these unfortunate states:

- **No boot loader screen**

  Check for GRUB misconfiguration, or a corrupt boot sector. You may have to re-install the boot loader.

- **Kernel fails to load**

  If the kernel **panics** during the boot process, it is most likely a misconfigured or corrupt kernel, or incorrect parameters specified on the kernel command line in the GRUB configuration file. If the kernel has booted successfully in the past, it has either been corrupted, or the kernel command line in the GRUB configuration file has been altered in an unproductive way. Depending on which, you can reinstall the kernel, or enter in the interactive GRUB menu at boot and use very minimal command line parameters and try to fix that way. Or, you can try booting into a rescue image as described in the next chapter.

- **Kernel loads, but fails to mount the root filesystem**

  The main causes here are:

  i. Misconfigured GRUB configuration file
  ii. Misconfigured `/etc/fstab`
  iii. No support for the root filesystem type either built into the kernel or as a module in the **initramfs** initial ram disk or filesystem.

- **Failure during the init process**

  There are many things that can go wrong once **init** starts; look closely at the messages that are displayed before things stop. If things were working before, with a different kernel, this is a big clue. Look out for corrupted filesystems, errors in startup scripts, etc. Try booting into a lower runlevel, such as 3 (no graphics) or 1 (single user mode).

Filesystem Corruption and Recovery

If during the boot process, one or more filesystems fail to mount, **fsck** may be used to attempt repair. However, before doing that, one should check that `/etc/fstab` has not yet been misconfigured or corrupted. Note once again that you could have a problem with a filesystem type the kernel you are running does not understand.

If root filesystem has been mounted you can examine this file, but `/` may have been mounted as read-only, so to edit the file and fix it you can run:

```
$ sudo mount -o remount,rw /
```

to remount it with write permission.

If `/etc/fstab` seems to be correct, you can move to **fsck**. First, should try:

```
$ sudo mount -a
```

to try and mount all filesystems. If this does not succeed completely, can try to manually mount the ones with problems. Should first run **fsck** to just examine; afterwards, can run it again to have it try and fix any errors found.

##

Back to top

Previous Chapter - Table of Contents - Next Chapteri

Chapter 44 System Rescue - Notes

44.2 Introduction

Sooner or later, system likely to undergo significant failure, such as failing to boot properly, to mount filesystems, initiate a desktop display, etc. **System Rescue** media in the form of optical disks or portable USB drives can be used to fix situation. Booting into either **emergency** or **single user** mode can enable using full suite of Linux tools to repair system back to normal function.

44.3 Learning Objectives:

- Explain what forms system rescue media come in and how they are made available or can be prepared.
- Know how to enter **emergency mode** and what can be done there.
- Know how to enter **single user mode**, what can be done there, and how it differs from emergency mode.

44.4 Rescue Media and Troubleshooting

Will discuss rescue and recovery disks and other media in the next session; here, will discuss in terms of troubleshooting. These media can contain valuable tools for evaluating (and fixing) trouble systems.

Exact choices will vary from one Linux distribution to another, but when you boot from an install or live CD/DVD or USB drive, will be able to select an option with a name like `Rescue Installed System`.

The rescue image generally contains a limited but powerful set of utilities useful for fixing problems on a system:

- Disk Maintenance Utilities
- Networking Utilities
- Miscellaneous Utilities
- Logging files.

44.5 Common Utilities on Rescue/Recovery Media

Rescue disks contain many useful programs, including:

- Disk utilities for creating partitions, managing RAID devices, managing logical volume, and creating filesystems;

  fdisk, **mdadm**, **pvcreate**, **vgcreate**, **lvcreate**, **mkfs**, and many others.

- Networking utilities for network debugging and network connectivity:

  ifconfig, **route**, **traceroute**, **mtr**, **host**, **ftp**, **scp**, **ssh**

- Numerous other commands are also available:

  bash, **chroot**, **ps**, **kill**, **vi**, **dd**, **tar**, **cpio**, **gzip**, **rpm**, **mkdir**, **ls**, **cp**, **mv**, and **rm** to name a few.

44.6 Using Rescue/Recovery Media

Rescue image will ask a number of questions upon starting. One of these is whether or not to mount your filesystems (if it can).

If so, mounted at somew here, usually at `/mnt/sysimage`. Can move to that directory to get to files, or can change into that environment w ith:

```
$ sudo chroot /mnt/sysimage
```

For a netw ork-based rescue, may also be asked to mount `/mnt/source`.

May install softw are packages from inside **chroot**-ed environment. May also be able to install them from ourside the **chroot**-ed environment. Eg., on an **rpm**-based system, by using the `--root` option to specify the location of the root directory:

```
$ sudo rpm -ivh --force --root=/mnt/sysimage /mnt/sources/Packages/vsftpd-2*.rpm
```

44.7 Emergency Boot Media

Emergency boot media are useful w hen your system w on't boot due to some issue such as missing, misconfigured, or corrupted files or a misconfigured service.

Rescue media may also be useful if the root passw ord is somehow lost or scrambled and needs to be reset.

Most Linux distributions permit the install media (CD, DVD, USB) and/or Live media to serve a double purpose as a rescue disk, w hich is very convenient. There are also special-purpose rescue disks available.

Live media (in any format) provide a complete and bootable operating system w hich runs in memory, rather than loading from disk. Users can experience and evaluate an operating system and/or Linux distribution w ithout actually installing it, or making any changes to the existing operating system on the computer.

Live removable media are unique in that they can run on a computer lacking secondary storage, such as a hard disk drive, or w ith a corrupted hard disk drive or file system, allow ing users to rescue data.

44.8 Using Rescue Media

Whether using Live, install, or rescue media, procedures for entering into a special operating system for rescue and recovery are the same, and as pointed out, one medium serves all three purposes.

Rescue/recovery mode can be accessed from an option on the boot menu w hen the system starts from removable media. In many cases, may have to type `rescue` on a line like:

```
boot: Linux rescue
```

Can't tell all the possibilities, as each distribution has somew hat different, but easy to ascertain procedures.

Next, can expect to be asked some questions, such as w hich language to continue in, as w ell as make some distribution-dependent choices. Then, w ill be prompted to select w here a valid rescue image is located: CD/DVD. Hard Drive, NFS, FTP, or HTTP.

Selected location must contain a valid installation tree, and the installation tree must be for the same Linux version as the rescue disk, and if using removable media, the installation tree must be the same as the one from w hich the media w as created. If using a `boot.iso` image dow nloaded from the vendor, then w ill also need a netw ork-based install tree.

Additional questions are asked about mounting your filesystems. If they can be found, they are mounted under `mnt/sysimage`. Will

then be given a shell prompt and access to various utilities to make the appropriate fixes to your system.

`chroot` can be used to better access root (`/`) filesystem.

44.9 Rescue USB Key

Many distributions provide provide a `boot.iso` image file for download (name may differ). Can use **dd** to place this on a USB key drive as in:

```
$ dd if=boot.iso of=/dev/sdX
```

assuming system recognizes the removable drive as `/dev/sdX`. Be aware this will obliterate the previously existing contents on the drive!

Assuming system has the capability of booting from USB media and the BIOS is configured for it, can then boot from this USB drive. Will then function in the same fashion as a rescue CD or DVD. However, note that the install tree will not be present on the USB drive; therefore, this method requires a network-based install tree if one is needed.

Helpful utilities such as **livecd-tools** and **liveusb-creator** allow specification of either a local drive or the Internet as the location for obtaining an install image, and then do all the hard work of constructing a bootable image and burning it on the removable drive. Extremely convenient and works for virtually all Linux distributions.

44.10 Emergency Mode

In **emergency mode** you are booted into the most minimal environment possible. Root filesystem is mounted read-only, no **init** scripts are run and almost nothing is set up.

The main advantage of emergency mode over single-user mode (to be described next) is that if **init** is corrupted or not working, can still mount filesystems to recover data that might be lost during a re-installation.

To enter emergency mode, need to select an entry from the GRUB boot menu and then hit `e` for edit. Then add the word `emergency` to the kernel command line before telling the system to boot. Will be asked for the root password before getting a shell prompt.

Note: may also enter emergency mode when a boot fails for a variety of reasons, including a corrupted filesystem.

44.11 Single User Mode

If system boots, but does not allow you to log in when it has completed booting, try **single user mode**. In single user mode:

- **init** is started
- Services are not started
- Network is not activated
- All possible filesystems are mounted
- root access is granted without a password
- A system maintenance command line shell is launched.

In this mode, system boots to **runlevel** 1 (in SysVinit language). Because single user mode automatically tried to mount your filesystem, cannot use it when root filesystem cannot be mounted successfully, or if the **init** configuration is corrupted.

To boot into single user mode, use the same method as decribed for emergency mode with one exception, replace the keyword `emergency` with the keyword `single`.

##

Back to top

Previous Chapter - Table of Contents - Jump to First Chapter