Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

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

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

Legal note

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

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

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

Filesystems and swap

Lecture 5

Chapter 1

Block devices

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 The CHS (cylinder,head,sector) addressing is obsolete.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

Block devices

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

Filesystems

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

Filesystems

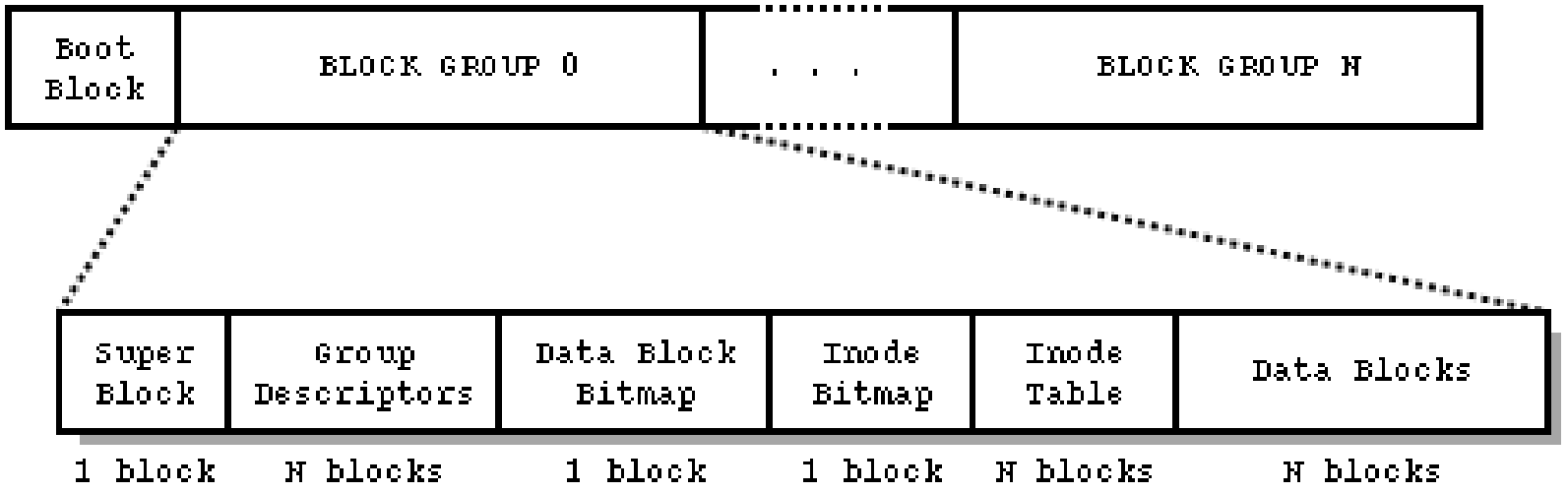
- Linux can support a large set of filesystems, including proprietary Microsoft (ntfs) and Apple (hfs+) FSs.
- There are a lot of filesystems native to Unix like OSes, e.g.: ext2/3/4, xfs, zfs, btrfs, reiserfs, jfs, f2fs, ...
- Each FS has a different set of features and a different way of implementing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

Filesystem features

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

The ext2 filesystem

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



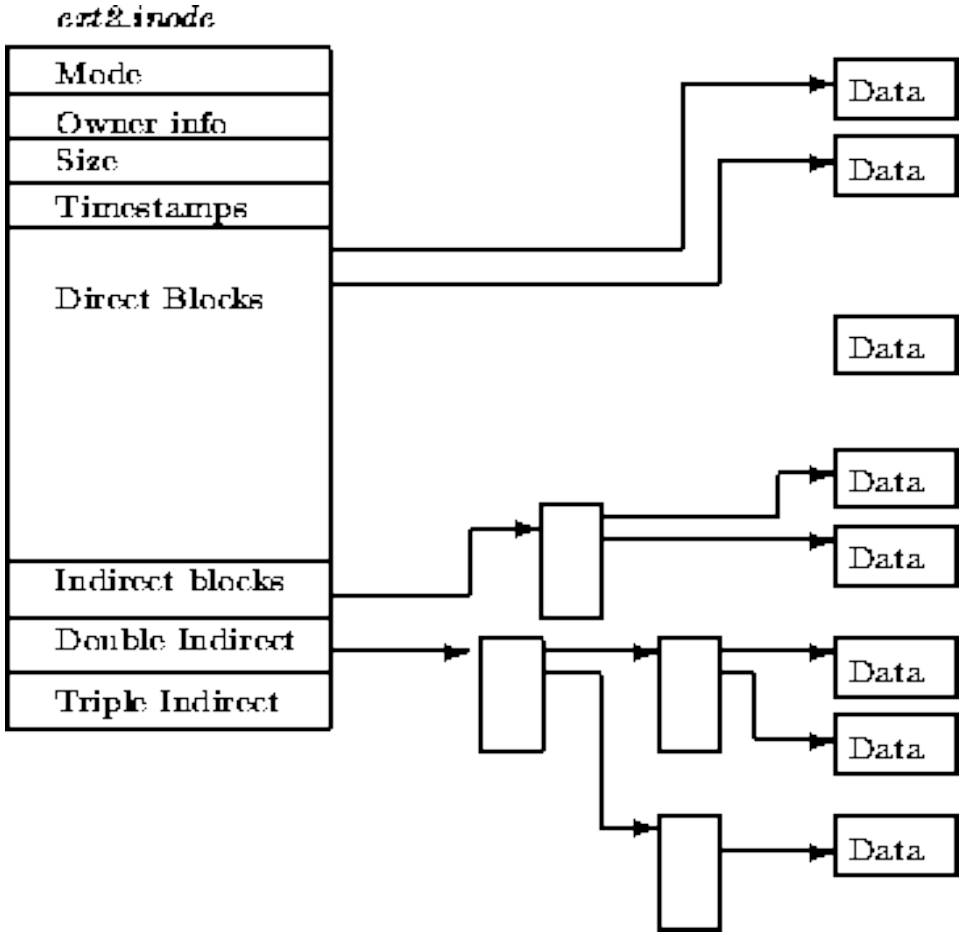
http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2.png

The ext2 filesystem

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

The ext2 filesystem

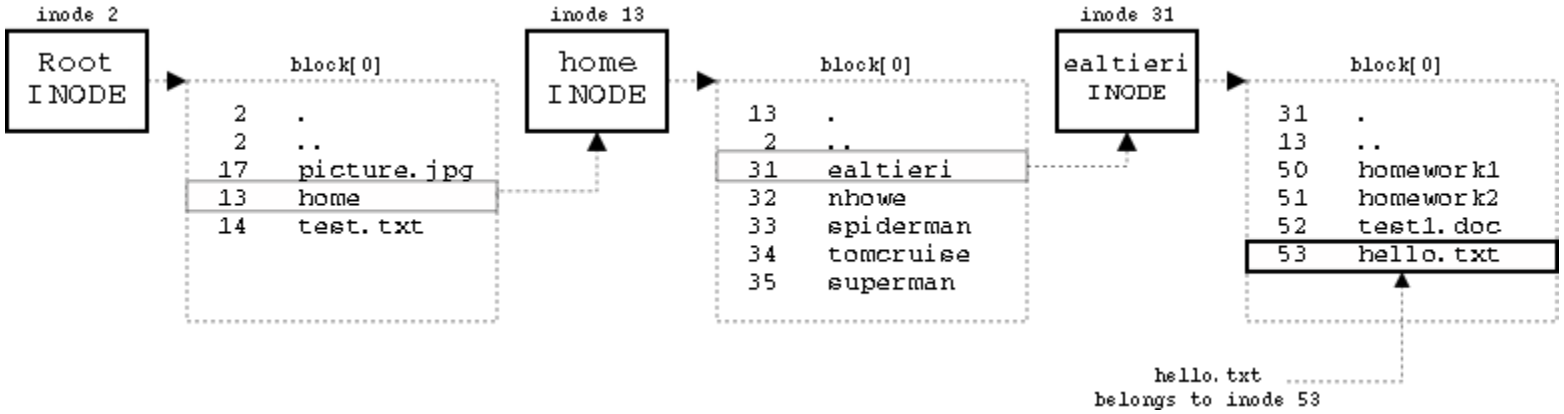
- In case the list of data blocks can't fit in a single inode, one to three levels of indirection blocks are inserted.



http://www.science.unitn.it/~fiorella/guidelinux/tlk/img84.gif

The ext2 filesystem

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



http://cs.smith.edu/~nhowe/262/oldlabs/img/ext2_locate.png

Links

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

Evolution of ext FS

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

Mounting

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

Mounting

Shell command line

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

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

Mounting

Shell command line

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

- Use the -o loop or -o loop=/dev/loop4 option to automatically assign or manually specify a loop device for image files before mounting. In this case the mount command will call the losetup utility to set up a loop device for convenience, and then mount the FS of the loopdevice on the given directory.
- In the given example the FS type will be probed for, but you could specify -t iso9660 or -t udf.
- You can also have raw hard disk images mounted this way, even in read-write mode. Some image formats require, that you specify an offset.
- You can use the losetup -d /dev/loop**N** to remove the Nth loopback block device after un-mounting it.

Creating filesystems

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

Shell command line

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

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

Managing filesystems

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

Shell command line

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

Managing filesystems

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

Managing filesystems

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

Shell command line

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

- Note the link counter (second column) in **ls -l** output. Use the **-s** switch to create a symbolic link instead of a hard link. Links can be deleted with **rm**, just like any file.
- You can use **sync** to flush all the write caches to disc. The umount command will call that automatically for you.
- A filesystem of tmpfs type is technically a RAM drive. It's not backed by storage blocks, but by memory pages. There is mkfs.tmpfs, rather use **mount -t tmpfs**.
- Use the **tune2fs** command to set FS parameters for ext2/3/4, like fsck scheduling, label, journaling, …
- Complete practice **RH124 CH 14.5.** and **14.8.**

Swap

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

Managing swap

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

Shell command line

```
# dd if=/dev/zero of=swap.file bs=1M count=1024
# mkswap swap.file
# swapon swap.file
```

- You can activate a swap space with **swapon filename**, deactivate with **swapoff filename.**
- You can check the amount of total and free swap space along with physical memory usage by issuing the **free** command.
- You can print the priority of each enabled swap space with **swapon -s**.
- Refer to **RH134 CH 9.3.** for more on swap.

/etc/fstab

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

/etc/fstab

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

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

Partitioning

Lecture 5

Chapter 2

Partitions

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

Partitions

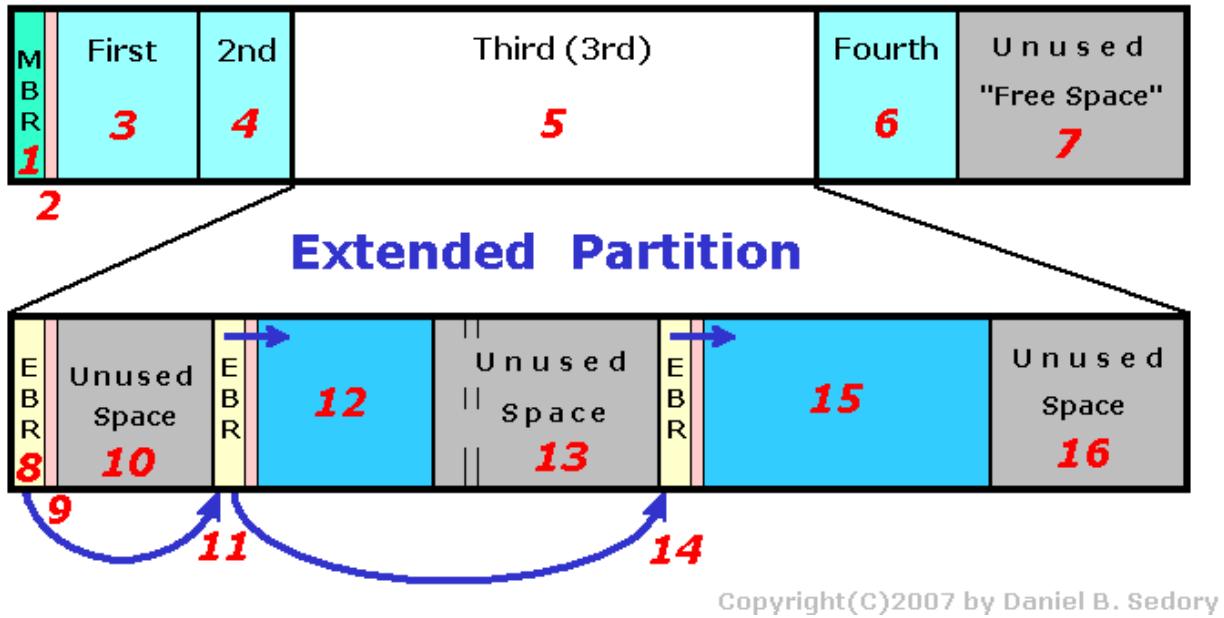
- One could have many reasons to partition a drive.
  - Multiboot: to install more than one OS, that have different native FSs. The filesystems need to be created on isolated block ranges with no overlap.
  - Backup schemes: If some part of the filesystem changes infrequently (e.g. system directories)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

MBR partitioning scheme

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

MBR partitioning scheme

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



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

MBR limitations

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

The GPT partitioning scheme

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

GPT benefits

- The primary header and primary table is repeated at the end of disk as a **secondary** instance for safety.
- There are no extended or logical partitions in this scheme. Practically any number of primary partitions are possible.
- The address limit of GPT is more than HUGE.
- All new OSs support GPT, but BIOS computers can't boot from a GPT disk directly.

Partiton numbering

- Partitions are numbered from 1 in Linux. The first logical partition (if present) always has the number of 5. For e.g.: /dev/sdc5 is the device file for the third SCSI disc's first logical partition.

Managing partitions

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

- Refer to **RH134 CH 9.1.** for more.
- Complete practices **RH134 CH 9.4.** and **9.5.**

Logical Volume Management

Lecture 5

Chapter 3

LVM

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

LVM

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

LVM

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

PVs

- You can create a PV with **pvcreate blockdev_1 blockdev_2 blockdev_n** command. Simply list 1 or more block devices after the command, to initialize them as PV.
- You can remove PV metadata from a block device with **pvremove blockdev_1 blockdev_n** if you do not want it to be involved in LVM any more.
- You can display PVs with **pvs**. Note the the **-o** option to select colums to display and **-O** for selecting the sort field.
- Use **pvdisplay** to display detailed PV information, including UUID, number of used and available extents.

VGs

- To create a VG from PVs use **vgcreate -s 2M vg_name pv_1 pv_2 pv_n** command. The first argument is the VG name, all the remaining arguments must be initialized PVs. All of them will be assigned to the newly created VG. Note the **-s** switch to specify the extent size. If omitted it will default to 4 MiB. It's conventional to name VGs with a vg_ prefix.
- Use **vgchange -s** to change the extent size later. Use **vgchange -a n VG_name** to deactivate a VG at runtime, while -a y switch will activate a whole VG at runtime.
- Use **vgs** to list or **vgdisplay** to detail existing volume groups.
- Use **vgreduce vg_name list_of_pvs** to remove one or more PVs from a VG. Use **vgextend vg_name list_of_pvs** to add free PVs to the existing volume group.

LVs

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

Managing LVs

- LVM devices are presented as /dev/dm-[number] files. There are two symlinks to each of the device nodes: /dev/mapper/vgname-lvname and /dev/vgname/lvname You can refer both when creating a filesystem or swap space on top of the LV, they represent the same LV.
- When shrinking, you have to reduce the FS first. In case of ext2/3/4 use **resize2fs /dev/vg_some/lv_any [size]** to resize the FS structure. Ext2/3/4 FS can only be shrunk offline, but can grow online. In case of growing you grow the LV first, then give the resize2fs command without size argument to fill up to the length of the LV.
- You can use **lsblk** also for listing LVs and child-parent relations.
- Refer to **RH134 CH 10.4.** for more.
- Complete practice **RH134 CH 10.6.**

LVM RAID

- When creating an lv with **lvcreate** you can specify raid personality with --type switch like --type raid1 or --type raid5. Use -i to specify number of stripes -I for stripe size, -m for number of mirrors (no. of disks -1).

Shell command line

```
# lvcreate -i2 -I64 -L 465G -n stripe_vol
vg_demo /dev/sdb1 /dev/sdc1
```

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

LVM cache

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

Shell command line

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

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

LVM thin volumes

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

Shell command line

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

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

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

LVM thin volumes

LVM snapshots

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

Shell command line

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

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

LVM snapshots

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

Shell command line

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

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

Recommended reading

Lecture 5

Chapter 4

Relevant chapters in RH:

- RH 124 CH14 (Linux filesystems)
- RH 134 CH9 (Partitions and file systems)
- RH 134 CH10 (LVM)

Read by next lecture:

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

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

Thank you for your attention!

Lecture 5, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK