Linux

LINUX FILE SYSTEM and PERMISSIONS

File systems overview

File system or **filesystem** (often abbreviated to **fs**) is a method and data structure that the operating system controls how data is stored and retrieved.

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more.

File systems types

A **disk file system** takes advantages of the ability of disk storage media to randomly address data in a short amount of time. Some disk file systems are journaling file systems or versioning file systems.

Examples: FAT (FAT12, FAT16, FAT32), exFAT, NTFS, APFS, ext2, ext3, ext4.

A **network file system** is a file system that acts as a client for a remote file access protocol. Programs using local interfaces can transparently create, manage and access hierarchical directories and files in remote network-connected computers.

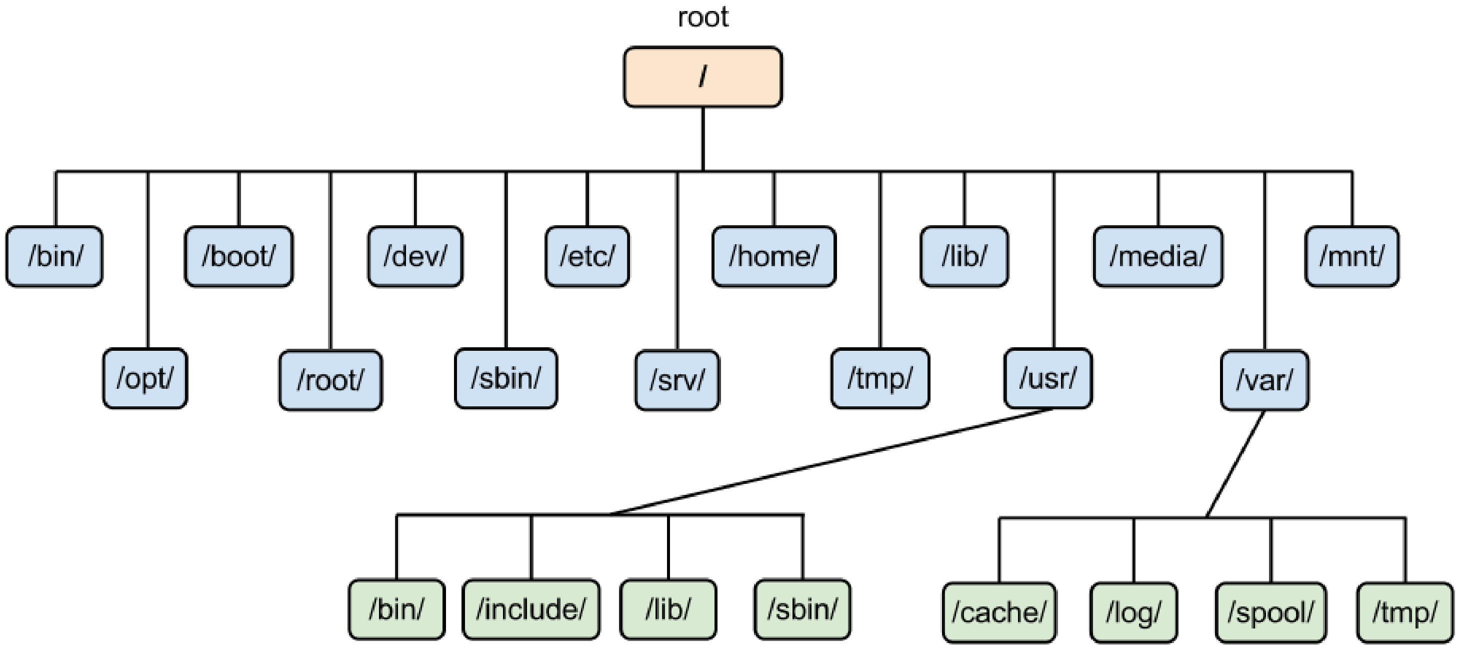
Examples: NFS, AFS, SMB protocols.

Special file systems

In the Linux kernel, **configfs** and **sysfs** provide files that can be used to query the kernel for information and configure entities in the kernel.

Procfs maps processes and, on Linux, other operating system structures into a filespace.

Linux file system hierarchy



Common directories

• /boot	bootloader files
• /bin	binary executable files
• /etc	configuration data
• /var	program data
• /tmp	temporary files
• /home | /root	home directories
• /usr	user applications files
• /dev	devices (virtual)
• /proc	processes (virtual)
• /run	runtime data (virtual)

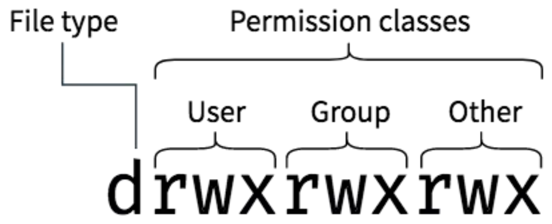
Navigation

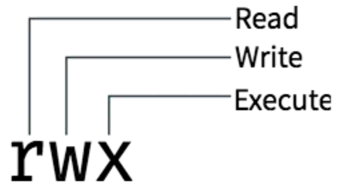
pwd	print current path
cd	change directory
ls	list files and directories
/	file system root
.	current directory pointer
..	parent directory pointer
~	home directory pointer

Viewing permissions and ownership

Use **ls -l** to display file system object permissions and ownership.

```
#ls -l
total 0
-rw-r--r--  1 Aleksei_Sokolov2  wheel  0 Sep  8 14:30 text
```





Effects of permissions on files and directories

Numeric	Symbolic	File actions	Directory actions
4	r (read)	Read contents	List contents
2	w (write)	Change contents	Modify object names
1	x (execute)	Run executable code	Navigate to directory

Special permissions

Setuid and **setgid** bits substitute the user / group executing a binary file with the file’s owner / group. They also enforce ownership of newly created files when applied to directories.

Sticky bit limits file deletion and name modification to respective owners / groups.

Managing ownership

Use **chown** to change the owner and group of a file system object.

Syntax:

```
chown <user>:<group> <object path>
```

Group can be omitted. To specify the group only, omit the user, but leave the semicolon.

Examples:

```
chown alice:employees file.txt
chown alice file.txt
chown :employees file.txt
```

Managing permissions

Use **chmod** to modify file system object permissions.

Syntax:

```
chmod <permissions> <object path>
```

Chmod supports numeric as well as target-operation-permission notation.
Targets are **u** for user, **g** for group, **o** for everyone and **a** for all of the above.
Supported operations are **+** for granting, **-** for revoking and **=** for setting explicitly.

Examples:

```
chmod 755 file.txt
chmod u+x file.txt
chmod a-rw file.txt
chmod ug=rx,o=r file.txt
```

Special attributes and access lists

Additional attributes may be set on files, e.g. to allow contents a file to be added upon, but not rewritten, or to prevent file renaming or deletion. These attributes are set with **chattr** and may be viewed with **lsattr**.

Linux uses simple POSIX permissions that are limited in flexibility. Fine-grained control might be achieved with **access control lists** (**ACLs**). ACLs may be defined by using **setfacl** and viewed with **getfacl**.

Default permissions and access masks

Default file system object permissions are defined by the programs that create them. For example, text editors create files to be readable and writeable, but not executable, by everyone; binary executables are created executable by the compilers.

To further limit the permissions that may be assigned to file system object by different programs, a **file permission creation mask**, or **umask**, is used. The mask defines which permissions can be set by a particular process and is assigned to said process. If any process attempts to set less restrictive permissions, these permission will not be applied.

Home work

1. Create new volume in VirtualBox, attach it to VM
2. Create two partitions. One of it should be formatted in ext3, second in BTRFS
3. Mount into your system as /homework/ext & /homework/btrfs
4. They should be mounted after reboot

THANK YOU