12- File system and access rights

- **File properties under Linux vs. Other systems**

- **File types**

(`-`)	Regular files......(s) is unknow to me till now (eg. `/dev/gpmctl`)
(`l`)	Symbolic Links (eg. `/sbin/init.d/rc2.d`......all files)
(**d**)	Directories and sub-directories
(**b**)	Block Device Files (eg. `/dev/hda1`...)
(**c**)	Character Device Files (eg. `/dev/tty1`....)
(**p**)	FIFO Named pipes (eg. `/dev/log`, `/dev/xconsole`)
(**s**)	Sockets

Note. The file names and directory names that starts with a Dot (`.`) are hidden from display by certain programs like `ls` etc.

- **Files and directories access rights**
  Access rights are restrictions applied on the content each file or directory.
  It doesn't restrict deletion of the file or directory. Only their parent directories access rights controls that.

- **Changing the files access rights**

`chmod [ugoa][+=-][rwx stXugo]` or `[0000 to 7777]` *`file`*

examples:

```
chmod u+w,g-x,o=wx file1
chmod 750 file2
chmod 4755 program1            (SUID=ON)
chmod u+s,g+s,o+t program2     SUID=ON,SGID=ON,StickyBit=ON)
chmod -R u=rwX,g=rX,o=rX dir1  Recursive 755 for directories.
                               and 644 for files.
```

- **Directories access rights**

- The read(r) without the search(x) access rights for directories makes no sense and the read bit is ignored.

- Any file (belonging to the user or not) under a directory set to write access to everybody can be erased by anybody.

Extra access rights			***u**ser*			***g**roup*			***o**thers*		
SUID (s)	***SGID (s)***	***Sticky Bit(t)***	***r***	***w***	***x***	***r***	***w***	***x***	***r***	***w***	***x***
4	***2***	***1***	***4***	***2***	***1***	***4***	***2***	***1***	***4***	***2***	***1***

- **SUID and SGID for programs** (`-rwsrwsrwx`)(`-rwSrwSrwx`)
  - SUID=ON: Effective user is owner of the program(w/SUID) started
  - SGID=ON: Effective group is the group of the program(w/SGID) started

- **SGID for Directories**
  Forces the subdirectories and files created in it, to have the same group as the directory. Independent of the user's group creating it.

– **Sticky Bit for <u>Directories</u> :**

Sets the rights to erase files only to their owner even if the directory is set to write for everybody. The sticky bit on **/temp** prevents that users processes erase files belonging to other users.
Note 1: Normally any file (belonging to the user or not) under a directory set to write access to group or others can be erased by users.
Note2: The **owner of the directory** can still erase any file even if the sticky bit is set.

- **<u>Sticky Bit for programs</u>:**
  – Allows an already run program to get stored in the ram (buffers) till the system goes down.
  Advantage: Fast load of program.
  Disadvantage: Uses lots of RAM

- **Command :**

  `chmod o+t` (sets the sticky bit)
  result= (`-rwxrwxrwt`) or (`-rwxrwxrwT`)
  `chmod u+t` (sets the SUID)
  result= (`-rwsrwxrwt`) or (`-rwSrwxrwT`)
  `chmod u+t` (sets the SGID)
  result= (`-rwxrwsrwt`) or (`-rwxrwSrwT`)
  Note: When adding a sticky-bit to a file/dir with an **<u>x</u>** for Others, the sticky-bit is displayed a **<u>t</u>** otherwise as **<u>T</u>** if the x was not present.
  The same applies to SUID and SGID (`-rw`**<u>S</u>**`rw`**<u>S</u>rw<u>T</u>**)

- **Attributes(chattr & lsattr)** for ext2/ext3 only...so far

  - Setting a directory or file to `append-only' attribute.
  - Command : **`chattr +a`** *`filename`* or *`directoryname`*
  - User must not necessarily be root
  - A file with this attribute may be appended to, but may not be deleted,and the existing contents of the file may not be overwritten. If a directory has this attribute, any files or directories within it may be modified as normal, but no files may be deleted.

  - Setting a directory or file to `immutable' attribute.
  - Command : **`chattr +i`** *`filename`* or *`directoryname`*
  - User must be ´root´
  - A file or directory with this attribute may not be modified, deleted, renamed, or (hard) linked

- **Display Attributes of files and directories**
  To list the (special) attributes of files and dirs. use the command **`lsattr`**

- **Attributes list:**

A	Atime record is not modified. Prevents too much disk access for laptops. Still in testing mode
a	Sets it to append mode only (can not erase it) Only root can set this attribute
c	The kernel compresses this file before storing The kernel decompresses it when reading it. NOT Implemented yet by kernel
d	Will not be backed-up by the program "dump"
i	Cannot be modified Cannot be erased Cannot be renamed Cannot be made a link to Only root can change this attribute
s	When this file is erased, its used blocks are written with '0' to prevent recovery at a later date.
S	Any change to this file will be immediately written on the disk instead of in the file system buffer. (equivalent to '`sync`' mount option)
u	When this file is deleted, its content is saved. It can therefore be undeleted later. NOT implemented yet by kernel.

- **umask for new files an directories**

Alows to control which access rights will be given to newly created files or directories:

```
New files access rights          = 666 !| umask (!|=Logical NOR)
New directories access rights    = 777 !| umask
```

- Examples:

umask	New files (666 access rights)	New Directories (777)
022	`-rw-r--r--`	`-rwxr-xr-x`
135	`-rw-r---w-`	`-rw-r---w-`
216	`-r--rw----`	`-r-xrw---x`

- **Directories access rights**

- Read and Search combination (`r-x`)
- normally any file (belonging to the user or not) under a directory set to write access to group or others can be erased by users

chown : Change user and group ownership of a file or directory
IMPORTANT:
`root` is the only user allowed to change ownership(`chown`) of files or directories.

Syntax:
```
chown [options] [user][:group] filename
chown [options] [user][:group] dirname
```

eg.

	chown	*user*:*group*	*filename*	Change user and group ownership of file
or	"	*user*	"	Change user ownership of file
or	"	*user*.	"	Change user and his group ownership of file
or	"	*user*:	"	" " "
or	"	.*group*	"	Change group ownership of file

Important Options:(from man page)

`-R --recursive`	Recursive:affect all the files and directories inside directory trees
`--dereference`	affect the referent of each symbolic link, rather than the symbolic link itself.
`-h, --no-dereference`	affect symbolic links instead of any referenced file. (available only on systems that can change the ownership of a symlink)
`--from=`*`CURRENT_OWNER`*`:`*`CURRENT_GROUP`*	change the owner and/or group of each file only if its current owner and/or group match those specified here. Either may be omitted, in which case a match is not required for the omitted attribute.
`-f, --silent, --quiet`	suppress most error messages
`-c, --changes`	like verbose but report only when a change is made
`--reference=RFILE`	use `RFILE`'s owner and group rather than the specified `OWNER:GROUP` values.
`-v, --verbose`	output a diagnostic for every file processed

chgrp : Change group ownership of a file or directrory

syntax: `chgrp [`*`options`*`]` *`newgroup filename`*

eg. `chgrp -R ftp /srv/www`
Changes recursively all the files and directories inside the dir. `/srv/www` to be owned by group `ftp`

`chgrp -R --reference=/home/hans /srv/ftp`
Changes recursively the group ownership of all the files and directories contained in `/srv/ftp` to the group owning the directory `/home/hans`

options:

`-R, --recursive`	operate on files and directories recursively
`--dereference`	affect the referent of each symbolic link, rather than the symbolic link itself
`-h, --no-dereference`	affect symbolic links instead of any referenced file (available only on systems that can change the ownership of a symlink)
`-f, --silent, --quiet`	suppress most error messages
`--reference=RFILE`	use `RFILE`'s group rather than the specified `GROUP` value
`-v, --verbose`	output a diagnostic for every file processed
`-c, --changes`	like verbose but report only when a change is made

Übung(20):

(20) **chmod & chown**
`cd ~`
`ls -l`
YasT - Administration des Systems -> Benutzerverwaltung
otto, password otto 2x, F4,
rainer, Gruppe F3 nogroup, password rainer 2x, F4, F10 <Esc> <Esc>
neuer xterm:

```
xterm -sb -rightbar -T rainer -bg lightcyan -e su -l rainer &
```

password:

neuer xterm: **`xterm -sb -rightbar -T otto -e su -l otto &`**
password:
`cd ~` !!!
`mkdir dir_a` ! als otto: Verzechnis dir_a & Datei file_a kreieren
`cd dir_a`
`cp ../.bashrc file_a`
`ls -l`

<u>rainer:</u> ls -l /home/otto/ ! Fehlermeldung rainer darf dieses Verz. nicht ansehen
`ls -l`

<u>als otto</u>: **`cd /home`**
`chmod g+rx otto` ! die Gruppe darf es anschauen

<u>als rainer</u>: **`ls -l /home/otto/`** ! Fehlermeldung: rainer ist nicht in der Gruppe users

als *eigenen Benutzer*: **`ls -l /home/otto/`** -in der Gruppe users is alles ok
<u>als otto</u>: **`chmod o=rx,g=r otto`** ! = setzen, das x von g ist weg
`ls -l`
`chmod 755 otto` ! alle darfen das Verzeichnis otto lesen

<u>als rainer</u>: **`ls-l /home/otto`** ! naless i.o.
`cp ~/.bashrc /home/otto/dir_a/file_b` ! Fehler: rainer darf bei otto nicht schreiben

<u>als otto</u>: **`cd ~/dir_a`**
`chmod a+x file_a` ! die Datei file_a wird ausführbar=script (Farbe rot)
`ls -l`

(4=s set user ID on execution, 2=s set group ID on execution, 1=t set sticky bit)
sticky bit = angeheftet
`chmod 4755 file_a` ! -rwsr-xr-x
`chmod 6755 file_a` ! -rwsr-sr-x
`chmod 7755 file_a` ! -rwsr-sr-t
----

chown *Benutzer.Gruppe Datei*

als otto: **chown *eigenen Benutzer* file_a** ! Fehlermeldung
su + password
chown *eigenen Benutzer* file_a
ls -l
exit, bin wieder otto

als eigenen Benutzer*:*
rm /home/otto/dir_1/file_a ! Fehlermeldung

als otto:
c**hmod g+w ~/dir_a**

als *eigenen Benutzer:*
rm /home/otto/dir_1/file_a ! diesesmal ist alles i.o.

su + password ! die Gruppe ändern
cd /home
chown .users rainer
ls -l
chown .nogroup rainer
exit

chattr

NAME

chattr - change file attributes on a Linux second extended file system

SYNOPSIS

chattr [-RV] [-v version] [mode] files...

DESCRIPTION

chattr changes the file attributes on a Linux second extended file system.

The format of a symbolic mode is +-=[ASacdisu].

The operator `+' causes the selected attributes to be added to the existing attributes of the files; `-' causes them to be removed; and `=' causes them to be the only attributes that the files have.

The letters `ASacdisu' select the new attributes for the files: don't update atime (A), synchronous updates (S), append only (a), compressed (c), immutable (i), no dump (d), secure deletion (s), and undeletable (u).

OPTIONS

-R Recursively change attributes of directories and their contents. Symbolic links encountered during recursive directory traversals are ignored.

-V Be verbose with chattr's output and print the pro gram version.

-v version
Set the filesystem version.

ATTRIBUTES

When a file with the 'A' attribute set is modified, its atime record is not modified. This avoids a certain amount of disk I/O for laptop systems.

A file with the `a' attribute set can only be open in append mode for writing. Only the superuser can set or clear this attribute.
A file with the `c' attribute set is automatically com pressed on the disk by the kernel. A read from this file returns uncompressed data. A write to this file compresses data before storing them on the disk.

A file with the `d' attribute set is not candidate for backup when the dump(8) program is run.

A file with the `i' attribute cannot be modified: it can not be deleted or renamed, no link can be created to this file and no data can be written to the file. Only the superuser can set or clear this attribute.

```
       When  a  file  with  the `s' attribute set is deleted, its
       blocks are zeroed and written back to the disk.

       When a file with the `S' attribute set  is  modified,  the
       changes  are  written  synchronously  on the disk; this is
       equivalent to the `sync' mount option applied to a  subset
       of the files.

       When  a  file  with  the `u' attribute set is deleted, its
       contents are saved.  This allows the user to ask  for  its
       undeletion.


AUTHOR
       chattr  was  written  by Remy Card <card@masi.ibp.fr>, the
       developer and maintainer of the ext2 fs.

BUGS AND LIMITATIONS
       As of ext2 fs 0.5a, the `c' and `u' attribute are not hon
       oured by the kernel code.  As of the Linux 2.0 kernel, the
       'A' attribute is not yet supported  by  the  kernel  code.
       (The noatime code is still in testing.)

       These  attributes  will be implemented in a future ext2 fs
       version.

AVAILABILITY
       chattr is part of the e2fsprogs package and  is  available
       for  anonymous ftp from tsx-11.mit.edu in /pub/linux/pack
       ages/ext2fs.

SEE ALSO
       lsattr(1)
```

ACL (Access Control List)

The ACL is an extention of the regular file/directory access rights.
Access Control Lists are a feature of the Linux kernel and are currently supported by ReiserFS, Ext2, Ext3, JFS, and XFS.The regular UNIX access permissions for a specific file or directory stay active for the owner,group,others, while other sets of permissions are given by means of ACLs, to give special access rights to specific users or groups for the same file or directory. Think of file/directory access permissions like this:
Regular access rights are the rule and the ACLs are the exceptions to the rule.
Note: The ACL mechanism needs to be enabled for a filesystem before being able to change it or use it. It is done by adding the mount option acl.
eg. `mount -o defaults,acl,usr_xattr -t ext3 /dev/hda3 /home`
It can also be introduced as mount option in `/etc/fstab`

ACL Terms

Access ACL
The access rights of a file or a directory defined by its regular permissions and the Extended ACL.

Default ACL (only for directories)
The access rights from which the files and directories will inherit when created within the directory.

Minimum ACL or Regular permissions
The minimum ACL is in fact the regular access rights applied to a file or a directory for the owner, the owning group and others.

Extended ACLs

The extended ACLs are extra ACLs entries made to extend the regular permissions. Each ACL for an a specific file or directory consists of a set of entries defining each different permission for either a user or a group.
An ACL entry contains: - a type (user or group)
- a name of the user or the group concerned
- a set of permissions for the user or the group concerned

eg. `user:bob:r-x` (type=`user`, name=`bob`, permissions=`r-x`)
Note1: When we list the ACLs the regular permissions are also displayed.
For these permissions the qualifier for the group or users is empty.
Note2: There is no need to assign any ACL entry for 'other' since it is already assigned using the regular access permissions.

Listing the ACL for a file or directory

Syntax: `getfacl` *`filename`*
`getfacl` *`dirname`*

The regular access rights will be displayed in the form of:
eg.
```
user::rwx
group::r--
other::r--
```

The extended ACLs (if existing) will then be displayed in the form of:
eg.
```
user:marie:rwx
group:admin:r--
```

The `mask` value

The `mask` is the representation of the maximum access permissions possible for all the groups assigned in the ACL and the regular group access permissions of a file or directory. This `mask` does nothing else than informing us how the group permission will look when we do an `ls -l` of the file or directory.
In fact this `mask` is calculated by the ACL mechanism and is simply the regular group permissions and the Extended ACL group permissions bitwise 'OR'ed together.
This calculation of the mask can be avoided by adding the option `-n`.

Note: If the file or directory ACL has a `mask` calculated, then the group access rights shown by the command `ls -l` will be the `mask` value. It does not reflect the real permissions of the owning group. It only reflects the maximum possible permissions of the owning groups because it considers also the groups permissions of the extended ACL.

eg1. if regular is `rw-` and Extended ACL is `r--` then the mask is `rw-`
eg2: if regular is `r--` and Extended ACL is `rwx` then the mask is `rwx`

Adding/Changing an ACL for a file or directory

Syntax: `setfacl -m u:`*`username`*`:`*`perm`*`,g:`*`groupname`*`:`*`perm`*` filename/dir`

```
eg1:  setfacl -m u:marie:rw- /var/db/data1
eg2:  setfacl -m u:paul:rw-,g:admin:rw- /data/marlo/
```

Note1: The indication that a file or a directory has extended ACL assigned, is done by a '**+**' at the end of the normal access rights displayed by `ls -l` command.
eg: `ls -l /var/db/data1`

```
-rw-rwxr--+  1 joe  video  0 2004-11-02 22:43 /var/db/data
```

Note2: The regular permissions can be changed using the regular `chmod` command or the `setfacl`.
eg. `setfacl -m u::rw- `*`file`* does the same as `chmod u=rw `*`file`*

Examples:

Granting an additional user read access

```
      setfacl -m u:lisa:r file
```

Revoking write access from all groups and all named users (using the effective rights mask)

```
      setfacl -m m::rx file
```

Removing a named group entry from a file's ACL

```
      setfacl -x g:staff file
```

Copying the ACL of one file to another

```
      getfacl file1 | setfacl --set-file=- file2
```

Copying the access ACL into the Default ACL

```
      getfacl -a dir | setfacl -d -M- dir
```

Default ACL

Default ACLs can only be applied to directories. The default ACL defines the access permissions all files or subdirectories under this directory inherit when they are created. A default ACL affects subdirectories as well as files.

Basically, the permissions in a default ACL are handed down in two ways:

- A <u>subdirectory</u> inherits the default ACL of the parent directory
  - as its own access ACL.
  - as its own default ACL
- A <u>file</u> inherits the default ACL of the parent directory
  - as its own access ACL

Default ACL vs. `umask` value

Normally the `umask` value controls the permissions of a newly created file or subdirectory. If a Default ACL is defined for a parent directory, it is this Default ACL that controls the permissions of a newly created file or subdirectory. The `umask` value has no more effect on these permissions.

Assigning Default ACL to a Directory

The following command assigns the Default ACL to a directory:

```
setfacl -d -m group:djungle:r-x mydir
```

The option `-d` of the `setfacl` command prompts `setfacl` to perform the following modifications (option `-m`) in the Default ACL.
The result would be the following (by issuing the `getfacl` command)

`getfacl mydir`

```
# file: mydir
# owner: tux
# group: project3
user::rwx
user:jane:rwx

group::r-x
group:djungle:rwx

mask::rwx

other::---

default:user::rwx
default:group::r-x
default:group:djungle:r-x
default:mask::r-x
default:other::---
```

`getfacl` returns both the access ACL and the default ACL. All lines that begin with default are the default ACL. Although you merely executed the `setfacl` command with an entry for the `djungle` group for the default ACL, `setfacl` automatically copied all other entries from the access ACL to form a valid default ACL.

Default ACLs do not have a direct effect on access permissions of the directory, they only come into play when files or subdirectories are created within it.
When handing down the permissions, only the default ACL of the parent directory

is taken into consideration, not the access permissions of the parent directory.

Results of creating files or subdirectories within this directory:

Created subdirectories:

- Regular permissions and extended ACL inherit from the Default ACL permissions of the parent directory.
- Default ACL inherit from the Default ACL of the parent directory.

Created files:

- Regular permissions and extended ACL inherit from the Default ACL permissions of the parent directory.

Example:

```
mkdir acltest
ls -ld acltest
      drwxr-xr-x    2 michel   video     4096 2004-11-10 22:57 acltest
```

Setting the directory Default ACL

```
setfacl -d -m group:users:rwx acltest
ls -ld acltest
      drwxr-xr-x+   3 michel   video     4096 2004-11-10 23:03 acltest
getfacl acltest
# file: acltest
# owner: michel
# group: video
user::rwx
group::r-x
other::r-x
default:user::rwx
default:group::r-x
default:group:users:rwx
default:mask::rwx
default:other::r-x
```

Creating a subdirectory within it:

```
mkdir acltest/testdir
ls -ld acltest/testdir
      drwxrwxr-x+   2 michel video  4096 2004-11-10 23:02 acltest/testdir
getfacl acltest/testdir
# file: acltest/testdir
# owner: michel
# group: video
user::rwx
group::r-x
group:users:rwx
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:users:rwx
default:mask::rwx
default:other::r-x
```

Creating a file within it:

```
touch acltest/file1
ls -l acltest/file1
      -rw-rw-r--+   1 michel  video  0  2004-11-10 23:03 acltest/file1
getfacl acltest/file1

# file: acltest/file1
# owner: michel
# group: video
```

```
user::rw-
group::r-x          #effective:r--   ( Because it is a file the X is taken out)
group:users:rwx     #effective:rw-   “”      “”       “”       “”
mask::rw-
other::r--
```

Deleting all Extended ACL of a file or directory
Syntax: `setfacl -b` *`filename`* or `setfacl -b` *`dirname`*

Deleting a default ACL of a file or directory
Syntax: `setfacl -k` *`filename`* or `setfacl -k` *`dirname`*

Saving ACLs of a full directory and subdirectories on file.
Syntax: `getfacl -R` *`/dirname`* `>` *`ACLFile`*

Recovering ACLs from a file.
Syntax: `setfacl --restore=`*`ACLFile`*
Restores a permission backup created by above `getfacl -R` or similar.
All permissions of a complete directory subtree are restored using this mechanism.
If the input contains owner comments or group comments, and `setfacl` is run by root, then the owner and owning group of all files are restored as well.
This option cannot be mixed with other options except `--test`.

Order of which ACL is being applied
As a basic rule, the ACL entries are examined in the following sequence: **owner**, **named user**, **owning group** or **named group**, and **other**.

Do `man setfacl` or `man acl`
to get more options and informations on ACLs or visit:
`http://sdb.suse.de/en/sdb/html/81_acl.html`
`http://acl.bestbits.at/`