Users and permissions

As with any operating system we will be working with different users on the system. We've been using the user `student`. We can confirm this by running the command `whoami`. Other commands we can run are `who` or `w`. These will show us all of the users that are logged in on the system (mind that both commands have a different output!).

User management

/etc/passwd

As we've seen before all system configuration is done using config files. The same is true for user management. User configuration is stored in the file `/etc/passwd`. It contains a list of user accounts together with some metadata seperated by colons:

```bash
student@linux-ess:~$ tail /etc/passwd
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
student:x:1000:1000:student:/home/student:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
```

The first account in this file is the `root` account:

bash

```
student@linux-ess:~$ head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

We can see more info than just usernames. These lines contain the user identifier (`1000` for `student`), home folder location, default shell, ...

ℹ You can find more information about this file, the columns and the contents by running the command `man 5 passwd` .

Adding users (useradd)

To add users we can simply use the `useradd` command. Its important to note that we have to run this command with `sudo` rights. This is because it is a system command that affects the entire system. To add a new user with the username `teacher` we could run the following command:

bash

```
student@linux-ess:~$ sudo useradd -m -d /home/teacher -c "Teacher Account
[sudo] password for student:
student@linux-ess:~$ tail -3 /etc/passwd
student:x:1000:1000:student:/home/student:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
teacher:x:1001:1001:Teacher Account:/home/teacher:/bin/sh
```

The `-d` takes in an argument and uses this argument as a path to where the homefolder has to be created. The `m` option (`create home`) will make sure the home folder actually gets created with the correct permissions. Lastly the `-c` (`comment`) option sets extra metadata to the user.

ℹ Mind that the option -d is optional and if not specified it defaults to a directory with the same name as the username within the directory /home.

ℹ We can delete users by running the command `userdel -r teacher` . The option `-r` will also remove that user's homefolder.

As seen in the folder `/home` or in the file `/etc/passwd` the new user has been created:

bash

```bash
student@linux-ess:~$ tail -1 /etc/passwd
teacher:x:1001:1001:Teacher Account:/home/teacher:/bin/sh
student@linux-ess:~$ ls -l /home
total 8
drwxr-x--- 5 student student 4096 Nov  4 16:04 student
drwxr-x--- 2 teacher teacher 4096 Nov  7 20:28 teacher
```

Every user has a userid (the third field in `/etc/passwd`). To view the userid of a user you can use the `id` command:

bash

```bash
student@linux-ess:~$ id teacher
uid=1001(teacher) gid=1001(teacher) groups=1001(teacher)
```

Default values

The `useradd` command uses quite some default values. We can check these default values by running the following command:

```
student@linux-ess:~$ sudo useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

These settings are kept in the file `/etc/default/useradd` and can be changed at any time by altering the file or using `useradd -D [option]` .

> Notice that the default value for the shell is `/bin/sh` . It is good practice to alter this to `/bin/bash` so that every new user that will be created in the future gets the `bourne again shell` as default (eg. sudo useradd -D --shell /bin/bash).

/etc/skel

By default homefolders are created as a subdirectory of the `/home` directory. These folders aren't created from scratch. The contents of the folder `/etc/skel` are copied within each newly created homefolder. This means that if we alter any contents in this `skel` folder, this wil actually be copied to any new user that will be created in the future:

```bash
student@linux-ess:~$ ls -a /etc/skel
.  ..  .bash_logout  .bashrc  .profile
student@linux-ess:~$ sudo ls -a /home/teacher/
.  ..  .bash_logout  .bashrc  .profile
```

Default profile files

.bashrc: executed everytime a new shell (bash) is started

.bash_logout: clears the screen when the user logs out

.profile: executed when user logs in

Editing users (usermod & userdel)

To edit a user's account configuration, we can use the `usermod` command. This command has several options that we can use to edit specific settings. For example to edit the comment field of a user we can run the following command:

```bash
student@linux-ess:~$ grep student /etc/passwd
student:x:1000:1000:student:/home/student:/bin/bash
student@linux-ess:~$ sudo usermod -c "Student Account" student
student@linux-ess:~$ grep student /etc/passwd
student:x:1000:1000:Student Account:/home/student:/bin/bash
```

To edit the default shell for a specific user we could do the following:

```bash
student@linux-ess:~$ tail -3 /etc/passwd
student:x:1000:1000:Student Account:/home/student:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
teacher:x:1001:1001:Teacher Account:/home/teacher:/bin/sh
student@linux-ess:~$ sudo usermod -s /bin/bash teacher
student@linux-ess:~$ tail -3 /etc/passwd
student:x:1000:1000:Student Account:/home/student:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
teacher:x:1001:1001:Teacher Account:/home/teacher:/bin/bash
```

View the manpage of `usermod` for all possible options.

If we want to create a user that cannot be logged into, we can change his shell to `/bin/false` or `/sbin/nologin` . The difference between these two is that `/sbin/nologin` gives a polite message that you cannot log in to this account before exiting. The option `/bin/false` directly exits without prompting anything.

Setting user passwords

If we want to change our own password we can use the `passwd` command (without sudo!):

bash

```
student@linux-ess:~$ passwd
Changing password for student.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

ℹ Note that your password has to be long and difficult enough, otherwise the new password will not be accepted.

ℹ Note that if we use `sudo passwd` that we are changing the password of the user root and not our own password!

As seen in the previous commands we created a new user with the username `teacher` but we never gave it a password. To do this we can run the `passwd` command with `sudo` rights and with a username as argument. This forces setting a new password for that specific user:

bash

```
student@linux-ess:~$ sudo passwd teacher
New password:
```

```
Retype new password:
passwd: password updated successfully
```

The password gets stored in the file `/etc/shadow` for security reasons. Regular users cannot view the contents of this file:

```bash
student@linux-ess:~$ tail -1 /etc/shadow
tail: cannot open '/etc/shadow' for reading: Permission denied
student@linux-ess:~$ sudo tail -1 /etc/shadow
teacher:$y$j9T$Vtf.U//c4/N/CB8LzHfnl0$5iCgijrpqXfaA3v18w/nAL2rl8BmiBYX5rn
```

As seen in the example above the password isn't shown in plaintext. This is because Linux systems use hashing algoritms to encrypt the passwords. The algoritm used here is called *sha512*.

Substitute user (su)

You can log out a user using `logout` or `exit`. Of course you can log back in as a different user after logging out. With the command su (substitute user) you can temporarily work as another user:

```bash
student@linux-ess:~$ whoami; pwd
student
/home/student
student@linux-ess:~$ su - teacher
Password:
teacher@linux-ess:~$ whoami; pwd
teacher
/home/teacher
teacher@linux-ess:~$ exit
logout
student@linux-ess:~$ whoami; pwd
student
/home/student
```

Note that root can become whoever he wants without the need to know that user's password:

```bash
student@linux-ess:~$ whoami; pwd
student
/home/student
student@linux-ess:~$ sudo su - teacher
teacher@linux-ess:~$ whoami; pwd
teacher
/home/teacher
teacher@linux-ess:~$ exit
logout
```

Note that to become root (without knowing his password) we can also use the `sudo su` command:

```bash
student@linux-ess:~$ whoami; pwd
student
/home/student
student@linux-ess:~$ sudo su - root
```

```
root@linux-ess:~# whoami
root
root@linux-ess:~# pwd
/root
root@linux-ess:~# exit
logout
```

We want to be mindful of commands that we run as the `root` user. This user can have permissions on all the files, folders and services on our system. This means that running commands as `root` can have a huge impact when being exploited.

Group management

View groupinfo of a user (id, groups)

To view the groups a user is member of we can use the commands `id` and `groups`:

```bash
student@linux-ess:~$ id student
uid=1000(student) gid=1000(student) groups=1000(student),4(adm),24(cdrom)
student@linux-ess:~$ groups student
student : student adm cdrom sudo dip plugdev lxd
```

/etc/group

The file where the group info of the users is stored is `/etc/group`. We could also make changes to this file instead of using commands.

```bash
student@linux-ess:~$ grep student /etc/group
adm:x:4:syslog,student
cdrom:x:24:student
```

```
sudo:x:27:student
dip:x:30:student
plugdev:x:46:student
lxd:x:110:student
student:x:1000:
```

Adding groups (groupadd)

If there's a need for new groups we can do it with the `groupadd` command:

```bash
student@linux-ess:~$ sudo groupadd ict
student@linux-ess:~$ tail -3 /etc/group
student:x:1000:
teacher:x:1001:
ict:x:1002:
```

We can delete groups using the command `groupdel` .

Editing groups (groupmod)

If we need to change a group we can use `groupmod` :

```bash
student@linux-ess:~$ sudo groupmod -n it ict
student@linux-ess:~$ tail -3 /etc/group
student:x:1000:
teacher:x:1001:
it:x:1002:
```

Edit group memberships (usermod)

If we want to add a user to a supplementary group we can also use the `usermod` command:

```bash
student@linux-ess:~$ sudo usermod -a -G it teacher
student@linux-ess:~$ id teacher
uid=1001(teacher) gid=1001(teacher) groups=1001(teacher),1002(it)
student@linux-ess:~$ groups teacher
teacher : teacher it
student@linux-ess:~$ grep it /etc/group
it:x:1002:teacher
```

❗ If we forget the `-a` (add) option the user will only be in the specified supplementary group and will be removed from all the groups he was in. This can be a serious problem if the user was the only one in the sudo group!

ℹ If we want to remove a user from a specific supplementary group we have to specify all the supplementary groups he must remain in (and don't use the -a option). In that case it will be easier to edit the group file `/etc/group` by hand.

ℹ The primary group of a user is specified in `/etc/passwd` and is the default group set on a new file or directory created by that user.

ℹ A user knows a change in group membership only when he logs in. So after a change a user has to login again to notice the difference.

In the example below we see that user student's `primary` group is changed to 'it' and this is shown by the id and grep of the passwd-file. However the groups-command does not show the change of group, also when we create a new file, the old primary group is still used. As explaned before, we need to log out and in again to get the changes to happen. This is also shown in the example.

bash

```
student@linux-ess:~$ grep student /etc/passwd
student:x:1000:1000:Student Account:/home/student:/bin/bash
student@linux-ess:~$ id student
uid=1000(student) gid=1000(student) groups=1000(student),4(adm),24(cdrom)
student@linux-ess:~$ groups
student : student adm cdrom sudo dip plugdev lxd
student@linux-ess:~$ ls -l
total 16
drwxr-xr-x 5 student student 4096 Nov  9 15:04 Documents
drwxr-xr-x 2 student student 4096 Oct 28 14:54 Downloads
drwxr-xr-x 6 student student 4096 Oct 23 12:22 linuscraft
drwx------ 5 student student 4096 Oct 19 15:07 snap
student@linux-ess:~$ sudo usermod -g it student
[sudo] password for student:
student@linux-ess:~$ grep student /etc/passwd
student:x:1000:1999:Student Account:/home/student:/bin/bash
student@linux-ess:~$ id student
uid=1000(student) gid=1999(it) groups=1999(it),4(adm),24(cdrom),27(sudo),
student@linux-ess:~$ groups student
student : student adm cdrom sudo dip plugdev lxd
student@linux-ess:~$ ls -l
total 16
drwxr-xr-x 5 student it 4096 Nov  9 15:04 Documents
drwxr-xr-x 2 student it 4096 Oct 28 14:54 Downloads
drwxr-xr-x 6 student it 4096 Oct 23 12:22 linuscraft
drwx------ 5 student it 4096 Oct 19 15:07 snap
student@linux-ess:~$ touch test
student@linux-ess:~$ ls -l
total 16
drwxr-xr-x 5 student it      4096 Nov  9 15:04 Documents
drwxr-xr-x 2 student it      4096 Oct 28 14:54 Downloads
drwxr-xr-x 6 student it      4096 Oct 23 12:22 linuscraft
drwx------ 5 student it      4096 Oct 19 15:07 snap
-rw-r--r-- 1 student student    0 Nov 18 14:02 test
student@linux-ess:~$ exit
```

Notice in the example above that when we changed the primary group, all the user student's files were also edited to the new primary group of this user. But because we didn't update the groups by loggin out and in again, our old primary group is used for the new file. In the Permissions part of this chapter we'll learn how to change the groupowner to correct this mistake. If we log in again, we see that our groups are updated.

```bash
ssh student@ip-address # Log in again
student@linux-ess:~$ touch test2
student@linux-ess:~$ ls -l
total 16
drwxr-xr-x 5 student it      4096 Nov  9 15:04 Documents
drwxr-xr-x 2 student it      4096 Oct 28 14:54 Downloads
drwxr-xr-x 6 student it      4096 Oct 23 12:22 linuscraft
drwx------ 5 student it      4096 Oct 19 15:07 snap
-rw-r--r-- 1 student student    0 Nov 18 14:02 test
-rw-r--r-- 1 student it         0 Nov 18 14:04 test2
student@linux-ess:~$ groups student
student : it adm cdrom sudo dip plugdev lxd
```

Permissions

From the very start, Unix (and thus Linux) was built as a multiuser operating system. Having multiple users on the same system means you need a way to keep users from accessing files from other users, and keep regular users from accessing files and programs that are intended to be used only by the system administrator. On the other hand users also need to be able to share files with others so they can collaborate effectively. As a system administrator being able to lock down or grant access to files is one the most important steps in keeping a system secure.

As you have seen in the previous section, your system already comes with a sensible set of file permissions. As a regular user you have full control in your own homefolder. You can create, edit and remove files and folders within your own home-directory `/home/student` . If you try to create or alter a file outside your homefolder you will get an error (permission denied). The only exception is the directory */tmp*. Eg: You are able to see the user database in `/etc/passwd` but cannot edit it as a regular user. Sometimes a regular user isn't even allowed to see the contents of a certain file or directory. Eg: `/etc/shadow` holds the passwords of users, that is why you are not even allowed to read it.

Permission errors are a common source of problem. Understanding and manipulating file permissions is a crucial step in becoming a competent Linux admin.

> Just because you can, doesn't mean you should. When troubleshooting permission errors always remember that permissions are your first line of defense against malicious actors. Always ask yourself why a program or a user should have access, and handle with caution. Don't just grant permissions to get rid of the error!

Octal notations

When looking at the extended info for a file using `ls` with the `-l` option (=long listing), you'll see three sets of three permissions applied to the file. The possible permissions are: read, write and execute. Permission to read a file's content, permission to change a file's content and permission to execute a file as a script or program. When a permission is not granted, you'll find a - in the position of the permission denied.

bash

```bash
student@linux-ess:~/course$ ls -l
total 4
-rw-rw-r-- 1 student student    0 okt  2 19:36 config
drwxrwxr-x 2 student student 4096 okt  2 19:36 folder
-rw-rw-r-- 1 student student    0 okt  2 19:36 rights.jpg
-rw-rw-r-- 1 student student    0 okt  2 19:36 test.txt
```

> The first character is a - (minus) for a regular file and a *d* for a directory.

> ℹ Directories in Linux have the same set of permissions. But because you need execute to access files in the directory, there is little you can do without it. The common permissions are rwx for a directory where you can do everything, r-x for a read-only directory, and ofcourse --- when you want to block access completely.

There are three sets because there are three different sets of people that permissions can be applied to. The first set describes the permissions for the owner of the file (the first name behind the permissions), the second applies to everyone that is a member of the group that owns the file (the second name). The last set is for everyone who doesn't fall under one of the first two categories. So in short: The three sets apply to userowner, groupowner and others, in that order.

When a user creates a file he automatically becomes the owner of that file. The group that owns the file is determined by the user's primary group. By default a user's primary group is a group with the same name as the username, which is why you'll often see owner and group owner have the same name (like *student student* in the above example). The `/dev` folder, that contains the files representing your hardware, is one of the places where you'll find files owned by the root-user with a different group as owner.

bash

```bash
student@linux-ess:~$ ls -l /dev/s[dr]?
brw-rw---- 1 root disk   8, 0 Nov 11 10:44 /dev/sda
brw-rw---- 1 root cdrom 11, 0 Nov 11 10:43 /dev/sr0
```

File permissions are written on disk as a field of bits in the file's properties. A bit is set to 1 when a permission is granted, 0 when it's not. So rwxrw-r-- becomes 111110100. Because humans are not very good at parsing binary sequences, they are represented as as octal numbers, numbers from 0 to 7 (000 to 111 in binary). To calculate the octal number remember that read is worth 4, write 2 and execute 1. Add those you need and you'll get the octal notation. The example above becomes 764 (rwx, 4+2+1=7, rw-, 4+2+0=6, r--, 4+0+0=4).

Changing permissions (chmod)

To change the permissions on a file you can use the `chmod` command.

To add or substract permissions of a file you can use the following method, where *rwx* stands for the rights you want to add or substract, and *ugo* stands for userowner, groupowner and others respectively. If you don't specify for who you want the change, it is changed on all three:

```bash
student@linux-ess:~$ touch test
student@linux-ess:~$ ls -l test
-rw-rw-r-- 1 student student 0 Nov 11 13:11 test
student@linux-ess:~$ chmod +x test
student@linux-ess:~$ ls -l test
-rwxrwxr-x 1 student student 0 okt  2 19:36 test
student@linux-ess:~$ chmod g-w test
student@linux-ess:~$ ls -l test
-rwxr-xr-x 1 student student 0 okt  2 19:36 test
student@linux-ess:~$ chmod go-rx test
student@linux-ess:~$ ls -l test
-rwx------ 1 student student 0 okt  2 19:36 test
```

When completely rewrite permissions you can also use chmod with an octal notation:

```bash
student@linux-ess:~$ touch config
student@linux-ess:~$ ls -l config
-rw-rw-r-- 1 student student    0 okt  2 19:36 config
student@linux-ess:~$ chmod 600 config
student@linux-ess:~$ ls -l config
-rw------- 1 student student    0 okt  2 19:36 config
student@linux-ess:~$ chmod 744 config
student@linux-ess:~$ ls -l config
-rwxr--r-- 1 student student    0 okt  2 19:36 config
```

The latter option is faster when you have to completely rewrite permissions (since you don't need to check the existing permissions, just overwrite). The former can be practical for making quick changes like making a file executable.

Changing ownership (chgrp, chown)

Besides changing the permissions of a file, you'll also need to change to whom these permissions apply to, by changing the user or group that owns the file. You will need sudo to assign files to different users/groups.

To change the group owner of a file or directory, you can use the `chgrp` command. To change this recursively use the `-R` option

bash

```
student@linux-ess:~$ ls -l course/
total 4
-rwxr--r-- 1 student student    0 okt  2 19:36 config
drwxrwxr-x 2 student student 4096 okt  2 19:36 folder
-rw-rw-r-- 1 student student    0 okt  2 19:36 rights.jpg
-rw-rw-rw- 1 student student    0 okt  2 19:36 test.txt
student@linux-ess:~$ sudo chgrp -R it course/
student@linux-ess:~$ ls -l course/
total 4
-rwxr--r-- 1 student it    0 okt  2 19:36 config
drwxrwxr-x 2 student it 4096 okt  2 19:36 folder
-rw-rw-r-- 1 student it    0 okt  2 19:36 rights.jpg
-rw-rw-rw- 1 student it    0 okt  2 19:36 test.txt
student@linux-ess:~$ ls -ld course/
drwxrwxr-x 3 student it 4096 okt  2 19:36 course/
```

The `chown` command is more versatile, it allows you to change owner and/or group. It has the same -R option to change an entire file-tree.

bash

```
student@linux-ess:~/course$ sudo chown teacher rights.jpg #changes the ow
student@linux-ess:~/course$ ls -l rights.jpg
-rw-rw-r-- 1 teacher it    0 okt  2 19:36 rights.jpg
```

```
student@linux-ess:~/course$ sudo chown teacher:root config folder #change
student@linux-ess:~/course$ ls -l
total 4
-rwxr--r-- 1 teacher root    0 okt  2 19:36 config
drwxrwxr-x 2 teacher root 4096 okt  2 19:36 folder
-rw-rw-r-- 1 teacher it        0 okt  2 19:36 rights.jpg
-rw-rw-rw- 1 student it        0 okt  2 19:36 test.txt
student@linux-ess:~/course$ sudo chown :it config #changes the group
student@linux-ess:~/course$ ls -l config
-rwxr--r-- 1 teacher    it        0 okt  2 19:36 config
```

Default permissions (umask)

A last thing we need to look at are the default permissions. What permissions are applied when you create new files and folders?

The maximum permissions for new files is 666, so -rw-rw-rw-. New files are never created with execute permissions. This is enforced by the kernel. It is of course possible to add the execute bit after file creation using `chmod`, but it always requires a conscious decission for security reasons. Folders don't have this limitation as a folder without an execute bit set is quite useless.

```bash
student@linux-ess:~/course$ touch file
student@linux-ess:~/course$ mkdir folder
student@linux-ess:~/course$ ls -l
total 4
-rw-rw-r-- 1 student student    0 okt 15 15:56 file
drwxrwxr-x 2 student student 4096 okt 15 15:56 folder
```

As you can see, we don't get the expected -rw-rw-rw- for the file, nor drwxrwxrwx for the folder. This is because most distributions are more strict than the Linux kernel allows. Using the kernel default would mean created files and folders are writable by every user on the system. They are however readable by `other` so beware of this with sensitive files.

The exact configuration of permissions for new files and folders is set by the `umask` . This is a value that defines the 'mask' that is applied for all newly created files and folders. To see the current mask, use the command `umask` .

bash

```
student@linux-ess:~/course$ umask
0002 #ignore the first 0 for now
```

So how does this work? We know the default is 666: if you substract the umask from that you get 664 or rw-rw-r-- (for folders we start with 777, so end with 775). A umask of 000 allows everything, a umask of 777 will make a new file have no permissions. The numbers still work the same (4 for read, 2 for write, 1 for execute) but this time you are using them to mask certain permission bits, or put more simply: deny certain permissions on new files and folders.

You can change the umask by using the same umask command.

bash

```
student@linux-ess:~/course$ umask 000
student@linux-ess:~/course$ touch newfile1
student@linux-ess:~/course$ ls -l newfile1
-rw-rw-rw- 1 student student 0 okt 15 16:23 newfile1
student@linux-ess:~/course$ umask 026 #if you want to mask multiple bits,
student@linux-ess:~/course$ touch newfile2
student@linux-ess:~/course$ ls -l newfile2
-rw-r----- 1 student student 0 okt 15 16:24 newfile2
student@linux-ess:~/course$ umask 077
student@linux-ess:~/course$ mkdir newfolder1
student@linux-ess:~/course$ ls -ld newfolder1
drwx------ 2 student student 4096 okt 15 16:25 newfolder1
```

ℹ Setting the umask using the command changes the umask for your current terminal session. Exiting the terminal will reset it to the default value. To make it permanent add `umask <your umask>` to your user's `.bashrc` file in your home directory.

bash

```
student@linux-ess:~/course$ touch file
student@linux-ess:~/course$ sudo touch file2
student@linux-ess:~/course$ ls -l
-rw-rw-r-- 1 student student 0 okt 15 16:44 file
-rw-r--r-- 1 root    root    0 okt 15 16:44 file2
```

One last thing to be aware of: If you look at the above files, you'll see that files created by the root user have a different umask set. This is explained by looking at the system-wide umask setting, found in the `/etc/login.defs` file.

bash

```
student@linux-ess:~/course$ nano /etc/login.defs
UMASK           022 #line 151
...
USERGROUPS_ENAB yes #line 230
```

To change the setting system-wide you can change the value for umask here. The default umask specified is the one the root user uses (no write for anybody but the owner). The reason files created by regular users get an extra w for the group, is the option on line 230. This option specifies that for any non-root user that has the same user-id as group-id (so the primary group is unchanged) the group umask-bits gets changed to the user umask-bits, explaining the 002 umask seen above.

You can also use the letter notation with umask:

bash

```
student@linux-ess:~$ umask
0002
student@linux-ess:~$ umask -S
u=rwx,g=rwx,o=rx
student@linux-ess:~$ touch file3
student@linux-ess:~$ ls -l file3
-rw-rw-r-- 1 student student 0 Nov 11 14:01 file3
```

```
student@linux-ess:~$ umask u=rwx,g=rx,o=
student@linux-ess:~$ umask
0027
student@linux-ess:~$ umask -S
u=rwx,g=rx,o=
student@linux-ess:~$ touch file4
student@linux-ess:~$ ls -l file4
-rw-r----- 1 student student 0 Nov 11 14:03 file4
```

Working together in a team (setgid)

If we want to be able to work together it is key that all users are able to change each others files. The solution to give all users the same primary group is a security issue because this also changes the rights on their homefolders:

bash

```
student@linux-ess:~$ sudo useradd -m -g it -s /bin/bash liam
student@linux-ess:~$ sudo passwd liam
student@linux-ess:~$ sudo useradd -m -g it -s /bin/bash jacob
student@linux-ess:~$ sudo passwd jacob
student@linux-ess:~$ ls -ld /home/liam  /home/jacob
drwxr-x--- 2 jacob it 4096 Nov 26 15:39 /home/jacob
drwxr-x--- 2 liam  it 4096 Nov 26 15:32 /home/liam
student@linux-ess:~$ sudo ls -l /home/jacob/.profile
-rw-r--r-- 1 jacob it 807 Jan  6  2022 /home/jacob/.profile
student@linux-ess:~$ su - jacob
Password:
jacob@linux-ess:~$ head -3 /home/liam/.profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
jacob@linux-ess:~$ exit
```

As you can see the user Jacob can view the files from the homefolder of the user Liam.

We will give the users their default primary group:

```bash
student@linux-ess:~$ sudo groupadd liam
student@linux-ess:~$ sudo groupadd jacob
student@linux-ess:~$ sudo usermod -g liam liam
student@linux-ess:~$ sudo usermod -g jacob jacob
student@linux-ess:~$ ls -ld /home/liam  /home/jacob
drwxr-x--- 2 jacob jacob 4096 Nov 26 15:39 /home/jacob
drwxr-x--- 2 liam  liam  4096 Nov 26 15:32 /home/liam
```

We will create a group for the two users to give them rights on a shared folder that we will create in a few. We'll also apply the users to the group:

```bash
student@linux-ess:~$ sudo groupadd ict
student@linux-ess:~$ sudo usermod -aG ict liam
student@linux-ess:~$ sudo usermod -aG ict jacob
student@linux-ess:~$ grep ict /etc/group
ict:x:1005:liam,jacob
```

We will make a directory that will be shared between the two users, and we will give it the necessary permissions:

```bash
student@linux-ess:~$ sudo mkdir -p /shares/ict
student@linux-ess:~$ ls -ld /shares/ict
drwxr-xr-x 2 root root 4096 Nov 26 15:58 /shares/ict
student@linux-ess:~$ sudo chgrp ict /shares/ict
student@linux-ess:~$ sudo chmod g+w /shares/ict
student@linux-ess:~$ ls -ld /shares/ict
drwxrwxr-x 2 root ict 4096 Nov 26 15:58 /shares/ict
```

As you can see members of the group ict have the permission to enter the share and to create/delete files and folders within the share

But there's a problem when the users create a file within the share:

```bash
student@linux-ess:~$ su - jacob
Password:
jacob@linux-ess:~$ cd /shares/ict/
jacob@linux-ess:/shares/ict$ touch testfile
jacob@linux-ess:/shares/ict$ mkdir testdir
jacob@linux-ess:/shares/ict$ ls -l
total 4
drwxrwxr-x 2 jacob jacob 4096 Nov 26 16:12 testdir
-rw-rw-r-- 1 jacob jacob    0 Nov 26 16:11 testfile
jacob@linux-ess:/shares/ict$ exit
logout
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict/
liam@linux-ess:/shares/ict$ ls
testdir  testfile
liam@linux-ess:/shares/ict$ echo hallo >> testfile
-bash: testfile: Permission denied
liam@linux-ess:/shares/ict$ ls -ld testdir
drwxrwxr-x 2 jacob jacob 4096 Nov 27 20:44 testdir
liam@linux-ess:/shares/ict$ cd testdir
liam@linux-ess:/shares/ict/testdir$ touch testfile2
touch: cannot touch 'testfile2': Permission denied
liam@linux-ess:/shares/ict$ exit
```

As you can see they cannot work together on the same files

A solution is the use of the special bit, named setgid. We can give it to the share with the command chmod g+s (to remove we would use g-s):

```bash
student@linux-ess:~$ ls -ld /shares/ict/
drwxrwxr-x 3 root ict 4096 Nov 26 16:12 /shares/ict/
student@linux-ess:~$ sudo chmod g+s /shares/ict
```

```
student@linux-ess:~$ ls -ld /shares/ict/
drwxrwsr-x 3 root ict 4096 Nov 26 16:12 /shares/ict/
```

As you can see in the permissions of the groupowner it now ends with a letter *s*. A lowercase *s* means that there is an *x* underneath, an uppercase *s* means that there is no *x* underneath.

The special bit setgid says that files and folders that will be created in this folder will have the same groupowner as this folder has:

bash

```
student@linux-ess:~$ ls -l /shares/ict/
total 4
drwxrwxr-x 2 jacob jacob 4096 Nov 26 16:12 testdir
-rw-rw-r-- 1 jacob jacob    0 Nov 26 16:11 testfile
student@linux-ess:~$ su - jacob
Password:
jacob@linux-ess:~$ cd /shares/ict/
jacob@linux-ess:/shares/ict$ echo "This is Jacob's text" > testfile2
jacob@linux-ess:/shares/ict$ mkdir testdir2
jacob@linux-ess:/shares/ict$ ls -l
total 12
drwxrwxr-x 2 jacob jacob 4096 Nov 26 16:12 testdir
drwxrwsr-x 2 jacob ict   4096 Nov 26 16:26 testdir2
-rw-rw-r-- 1 jacob jacob    0 Nov 26 16:11 testfile
-rw-rw-r-- 1 jacob ict     21 Nov 26 16:26 testfile2
jacob@linux-ess:/shares/ict$ exit
logout
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict/
liam@linux-ess:/shares/ict$ echo "This is Liam's text" >> testfile2
liam@linux-ess:/shares/ict$ cat testfile2
This is Jacob's text
This is Liam's text
liam@linux-ess:/shares/ict$ exit
```

As you can see now both users can work together with eachother's files.

The sticky bit

The setgid bit solves the main problem of making files in a shared folder created by one user accessible to other users in the same non-primary group. However this opens up another problem: Every user with access to the share can now delete or rename the files other users place in this folder. In some cases, like a shared project people are working on, this is fine. But in cases you don't want to allow this, another special permission bit is used: the sticky bit. Setting this bit on a folder will disallow any user (except the root user) from renaming or removing files or subfolders he does not own inside that folder.

This principle is also used in the system's ´/tmp´ folder, disallowing users to remove another user's temporary files.

bash

```bash
student@linux-ess:~$ ls -ld /tmp/
drwxrwxrwt 24 root root 4096 nov 27 13:50 /tmp/
```

Notice the t that replaced the x in the 'other' set of permissions when the sticky bit is set.

Just like with other permission bits you use chmod to add or remove the sticky bit.

bash

```bash
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict/
liam@linux-ess:/shares/ict$ ls -l
total 8
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testdir
drwxrwsr-x 2 jacob ict   4096 nov 27 15:03 testdir2
```

```
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testfile
-rw-rw-r-- 1 jacob ict      4 nov 27 15:03 testfile2
liam@linux-ess:/shares/ict$ rm testfile2     #Liam can remove Jacob's fil
liam@linux-ess:/shares/ict$ ls -l
total 4
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testdir
drwxrwsr-x 2 jacob ict   4096 nov 27 15:03 testdir2
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testfile
liam@linux-ess:/shares/ict$ exit
logout
student@linux-ess:~$ sudo chmod +t /shares/ict/
student@linux-ess:~$ ls -ld /shares/ict/
drwxrwsr-t 3 root ict 4096 nov 27 15:05 /shares/ict/
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict
liam@linux-ess:/shares/ict$ rm -rf testdir2/  #Liam can no longer delete
rm: cannot remove 'testdir2/': Operation not permitted
liam@linux-ess:/shares/ict$ exit
logout
student@linux-ess:~$ sudo chmod -t /shares/ict/
student@linux-ess:~$ ls -ld /shares/ict/
drwxrwsr-x 3 root ict 4096 nov 27 15:05 /shares/ict/
```

Within the special permissions field, the sticky bit is the rightmost bit, with a value of one. You can use this if you want to completely rewrite the permissions of a folder using octal notation. Just add a one before the standard mode.

```bash
student@linux-ess:~$ cd /shares/
student@linux-ess:/shares$ sudo mkdir ict2
student@linux-ess:/shares$ sudo chown :ict ict2/
student@linux-ess:/shares$ ls -ld ict2/
drwxr-xr-x 2 root ict 4096 nov 27 15:16 ict2/
student@linux-ess:/shares$ sudo chmod 1775 ict2/
student@linux-ess:/shares$ ls -ld ict2/
drwxrwxr-t 2 root ict 4096 nov 27 15:16 ict2/
```

```
student@linux-ess:/shares$ sudo mkdir ict3
student@linux-ess:/shares$ sudo chown :ict ict3/
student@linux-ess:/shares$ ls -ld
drwxr-xr-x 5 root root 4096 nov 27 15:20 .
student@linux-ess:/shares$ sudo chmod 3770 ict3/
student@linux-ess:/shares$ ls -ld
drwxr-xr-x 5 root root 4096 nov 27 15:20 .
student@linux-ess:/shares$ ls -ld ict3/
drwxrws--T 2 root ict 4096 nov 27 15:20 ict3/ #both setgid and sticky bit
```

As you can see above: To combine it with the setgid bit, the second bit in the triplet with a value of two, you add both values together.

To unset the sticky bit use a zero. A three-digit mode will also remove the sticky bit. Notice that the setgid bit is kept. To remove all special permissions add another zero in front.

```bash
student@linux-ess:/shares$ ls -ld ict3/
drwxrws--T 2 root ict 4096 nov 27 15:20 ict3/
student@linux-ess:/shares$ sudo chmod 0777 ict3/
student@linux-ess:/shares$ ls -ld ict3/
drwxrwsrwx 2 root ict 4096 nov 27 15:20 ict3/
student@linux-ess:/shares$ sudo chmod 00775 ict3/
student@linux-ess:/shares$ ls -ld ict3/
drwxrwxr-x 2 root ict 4096 nov 27 15:20 ict3/
```

If you want the users of the group ict to work together on each others files and you want that they can't remove each other files you have to set the kernel parameter fs.protected_regular=1.

```bash
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict/
liam@linux-ess:/shares/ict$ ls -l
total 8
```

```
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testdir
drwxrwsr-x 2 jacob ict   4096 nov 27 15:03 testdir2
-rw-rw-r-- 1 jacob jacob    0 nov 27 14:59 testfile
-rw-rw-r-- 1 jacob ict      4 nov 27 15:03 testfile2
liam@linux-ess:/shares/ict$ echo text from liam > testfile2     #Liam can
-bash: testfile2 Permission denied
liam@linux-ess:/shares/ict$ exit
student@linux-ess:~$ sudo sysctl -a | grep regular
fs.protected_regular = 2
student@linux-ess:~$ sudo sysctl fs.protected_regular=1
fs.protected_regular = 1
student@linux-ess:~$ su - liam
Password:
liam@linux-ess:~$ cd /shares/ict/
liam@linux-ess:/shares/ict$ echo text from liam > testfile2     #Liam can
liam@linux-ess:~$ cat testfile2
text from liam
```

If we want to keep the kernel parameter in the future we have to change this parameter in the file _/usr/lib/sysctl.d/99-protect-links.conf

As you may have noticed there is a third bit we haven't talked about. setuid, the first bit in the field. This allows executable files to run with the permissions of the owner of the file, not the one executing it. This is used by the *passwd* command to allow users to change their own password for example, as a normal user has no access to the /etc/shadow-file. Setting the setuid bit can have serious security risks, and is almost always a very bad idea. So you should probably ignore this knowledge

Access control lists

The ACL feature was created to give users the ability to selectively share files and folders with other users and groups. Before ACL's are usable we need to install the needed package:

bash

```
student@linux-ess:~$ sudo apt install acl
```

When installed, it needs to be turned on when the filesystem is mounted. In our Ubuntu installation ACL's are loaded by default. To add ACL's to a file or folder, use the `setfacl` command. ACL's can be viewed with the `getfacl` command.

To add ACL's you need to be the owner of the file or folder, if you are added by an ACL you will not be able to add ACL's yourself.
ACL permissions have precedence over the regular file permissions.
All ACL permissions are cumulative, this means if we are in 2 groups that are added to a file with ACL's. One with r-- rights and one with rwx rights, we will have rwx rights.

With the `setfacl` command, we'll be able to modify (-m) or delete (-x) ACL's.

bash

```
student@linux-ess:~$ touch memo
student@linux-ess:~$ ls -l memo
-rw-rw-r-- 1 student student 0 Nov 11 14:15 memo
student@linux-ess:~$ getfacl memo
# file: memo
# owner: student
# group: student
user::rw-
group::rw-
other::r--

student@linux-ess:~$ setfacl -m u:teacher:rw memo
student@linux-ess:~$ setfacl -m g:it:rw memo
student@linux-ess:~$ ls -l memo
-rw-rw-r--+ 1 student student 0 Nov 11 14:15 memo
```

With `getfacl` we can check the existing ACL's on a file or folder. Note that we can also see that ACL's are set by a + in the `ls -l` command

bash

```
student@linux-ess:~$ getfacl memo
# file: memo
# owner: student
# group: student
user::rw-
user:teacher:rw-
group::rw-
group:it:rw-
mask::rw-
other::r--
```

For teacher to be able to access the student's file memo in it's homefolder we need to edit some permissions. A possible solution would be: `setfacl -m u:teacher:rx /home/student` . Now, teacher can enter and look in the homefolder of student.

In previous example, we also see a mask option, this option decides the maximum permission and also has precedence over the regular file permissions except for the user owner. We can also add this parameter as follows:

bash

```
student@linux-ess:~$ setfacl -m m:r memo
student@linux-ess:~$ getfacl memo
# file: memo
# owner: student
# group: student
user::rw-
user:teacher:rw-                #effective:r--
group::rw-
group:it:rw-                    #effective:r--
mask::r--
other::r--
```

```
student@linux-ess:~$ ls -l memo
-rw-r--r--+ 1 student student 0 Nov 11 14:15 memo
```

If we want to remove an ACL entry from the file we can use the -x option:

```bash
student@linux-ess:~$ setfacl -x g:it memo
student@linux-ess:~$ getfacl memo
# file: memo
# owner: student
# group: student
user::rw-
user:teacher:rw-                #effective:r--
group::rw-
mask::r--
other::r--
```

If we want to remove all ACL entries from the file we can use the -b option:

```bash
student@linux-ess:~$ setfacl -b memo
student@linux-ess:~$ getfacl memo
# file: memo
# owner: student
# group: student
user::rw-
group::rw-
other::r--
```

We can also add default ACL’s by adding the d: parameter or adding the -d option. The default part makes sure new files and folders get the same ACL’s as their parent directory. Note that this only applies if the user creating the file or folder has the permissions to do so!

bash

```
student@linux-ess:~$ mkdir memos
student@linux-ess:~$ ls -ld memos
drwxrwxr-x 2 student student 4096 Nov 11 14:48 memos
student@linux-ess:~$ getfacl memos
# file: memos
# owner: student
# group: student
user::rwx
group::rwx
other::r-x
student@linux-ess:~$ setfacl -m g:it:rwx memos         # we need to add t
student@linux-ess:~$ setfacl -m d:g:it:rwx memos       # or setfacl -d -m
student@linux-ess:~$ getfacl memos
# file: memos
# owner: student
# group: student
user::rwx
group::rwx
group:it:rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:it:rwx
default:mask::rwx
default:other::r-x

student@linux-ess:~$ mkdir memos/January
student@linux-ess:~$ getfacl memos/January/
# file: memos/January/
# owner: student
# group: student
user::rwx
group::rwx
group:it:rwx
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
```

```
default:group:it:rwx
default:mask::rwx
default:other::r-x

student@linux-ess:~$ touch memos/January/22
student@linux-ess:~$ getfacl memos/January/22
# file: memos/January/22
# owner: student
# group: student
user::rw-
group::r-x                    #effective:r--
group:it:rwx                  #effective:rw-
mask::rw-
other::r--
```

Previous
8 Filters & file operations

Next
Lab