SysAdmin - Part 3 - Users management & Users rights

Michel FACERIAS

12 novembre 2024

Polytech Montpellier

Université de Montpellier

Table des matières

1 Users management

1.1 Understanding users

In Linux, the identity of users is defined in a **meta-database**. It is composed of :
— a **local part, which always exists** ;
— a **remote part, optionally provided** via the network.

In this part, we are going to describe the local part only.

1.1.1 Concept : Local user database

 The `/etc/passwd` file is a **textual database** of **information about users** who can log on to the system. The name of the file comes from one of its original functions, which was to contain the data used to verify passwords too.

It contains several fields :

1. the **user name** (login name) ;
2. the password, but in modern usage, this field contains `"x"` (or `"*"`, for disabled accounts) ;
3. a unique **user identifier number** (UID) ;
4. a unique main **group identifier number** (GID) ;
5. a mostly empty field (or a GECOS, see https://en.wikipedia.org/wiki/Gecos_field) ;
6. the path of the **user home directory** ;
7. the path of the **user shell**.

UID smaller than 1000 are for **system users** (services). Normal users starts at UID 1000.

To read more about :
— `/etc/passwd` file : https://en.wikipedia.org/wiki/Passwd
— UID and GID, you can go to https://en.wikipedia.org/wiki/User_identifier

1.1.2 Concept : Local groups database

 The `/etc/group` file is a **textual database** of **information about groups** that users belong to.

It contains several fields :

1. the **group name** ;
2. a password, but in modern usage, this field contains `"x"`
3. unique **group identifier** number (GID) ;
4. users belonging to this group.

1.1.3 Concept : Local password database

 `/etc/shadow` is used to **increase the security level of passwords**. Typically, that data is kept in files owned by and accessible only by the superuser and is hashed (now using SHA-512).

To see more : https://en.wikipedia.org/wiki/Passwd#Password_file

1.1.4 Concept : User identity

 In Linux, a user is defined by :
— his **UID** (known from `/etc/passwd`) ;
— the **GID** of his mandatory group (known from `/etc/passwd` too) ;
— other groups (known from `/etc/group`)

1.1.5 To do : Who am I ?

 You can use the `whoami` command to **know who you are** !

But the `id` command gives you **more information about a user**. Without any argument, it gives you information about the current user.

Let's have a look, here for the *test* user :

```
$ whoami
test

$ id
uid=1000(test) gid=1000(test) groupes=1000(test),24(cdrom),25(floppy),29(audio),30(dip),44(
    video),46(plugdev),108(netdev)
```

You can see :
— the UID (1000) and the user name (test) ;
— the GID (1000) and the main group name (test) ;
— the other GID (1000, 24, 25, ...) and group names (test, cdrom, floppy, ...) that this user belongs.

Now, we can try to get information about an other user, giving his name :

```
$ id root
uid=0(root) gid=0(root) groupes=0(root)
```

This user is the **super-user *root***. Its UID and GID are 0. In most cases, it **doesn't belong to any other group**, because it's not necessary !!!

Try on your own computer.

1.1.6 To do : Playing with users and groups

 Under your own identity :
— Try to list all users defined on your computer using `cat` command ;
— Try to list all groups defined on your computer using `cat` command ;
— Try to list all hashed passwords defined on your computer using `cat` command ;
— Explain why the last command hangs.

1.2 Managing identities

1.2.1 Concept : Managing users

 To manage users, you **need to edit all the database files** described before. But, off course, there are **specific command to handle user** creation and deletion without editing this files manually :
— `adduser` and `useradd`, to create user ;
— `deluser` and `userdel`, to delete users.

The **first ones are hight-level command**. They use the second, that are low-level, to do the job. **So prefer the first ones**.

⚠ Since these commands **alter identity files**, they can **only be used by *root* user** !

1.2.2 Concept : Managing groups

 To manage groups, you **need to edit the group database files** described before. But, of course, there are **specific command to handle groups** creation and deletion without editing this files manually :

— `addgroup` and `groupadd`, to create groups ;
— `delgroup` and `groupdel`, to delete groups.

The **first ones are hight-level command** too. They use the second, that are low-level, to do the job. **So prefer the first ones**.

⚠ Since these commands **alter identity files**, they can **only be used by *root* user** !

1.2.3 Concept : Managing group membership

 `addgroup` (or `adduser`) and `delgroup` (or `deluser`) can also be used to **manage the membership of users in groups**.

They just need **existing user and group names as argument**.

See `man addgroup` and `man delgroup` for more information.

1.2.4 Concept : Changing identity

 A user can change his identity. It can be done by using :

— the `su` command (*Substitute User*), to **become another user and stay that way** ;
— the `sudo` command (*Substitute User, then DO*), **to become another user just for running a command** and became himself again.

Both commands use **the new user name as an argument**.

The `sudo` command **just need the user to be declared in a specific database** (in `/etc/sudoers` file).

The `su` command **need to know the password of the user you want to became**.

Using `su -` is a relogging in as the new user identity, **as if he had logged in himself** (home folder and other environment stuff).

⚠ `sudo` is usually used to give *root* privilege to a normal user, by **limiting the commands they can use**. But, as this **list of command is rarely defined**, `sudo` is mistaken for `su`. If you want to **delegate the entire management of a computer to a normal user**, use `su`, and **train your users to understand that with great power comes great responsibility**.

2 User credentials

2.1 Understanding files credentials

2.1.1 Concept : File metadata

Each file has metadata information, rights mask at the begining (the last six characters only), user and group ownership :

```
$ ls -l
-rw-r--r-- 1 owkenobi owkenobi 0 may 04 12:34 the_file

--rights--   --user-- --group--
```

File rights use **flags** that belong to **four classes** :
— **s** class : for **Special** (we will see in an other chapter) ;
— **u** class : for **Users** whose the file belongs ;
— **g** class : for the **Group** which the files belongs ;
— **o** class : for **Other** users (not the user or group owner).

Each **3 flags** gives an atomic **right** :
— **r** right : for "**Read**" ;
— **w** right : for "**Write**" ;
— **x** right : for "**eXecute**".

Rights should be understood differently for files or folders :

Flag	File	Folder
`r`	should **read** the content	should **read** the list (do `ls` in it)
`w`	should **write** the content	should write the list (**create** file or folder in it)
`x`	should be **launch** (treated as a program)	should **be the current path** (do `cd` in)

Note that `r` is needed on the file too, if you want to execute as a program :
— r : to read the content of the program (executable code) ;
— x : to force the shell to use it as a program.

2.1.2 To do : Evaluate some file rights

You are going to evaluate some file rights. Open a console and look at the rights of the regular files in `/bin` folder :

```
$ ll /bin
total 16172
-rwxr-xr-x 1 root root 1234376 27 mars  20:40 bash
-rwxr-xr-x 1 root root  829136 14 mai    2021 btrfs
lrwxrwxrwx 1 root root       5 14 mai    2021 btrfsck -> btrfs
-rwxr-xr-x 1 root root  455344 14 mai    2021 btrfs-convert
-rwxr-xr-x 1 root root  426544 14 mai    2021 btrfs-find-root
-rwxr-xr-x 1 root root  447120 14 mai    2021 btrfs-image
-rwxr-xr-x 1 root root  430576 14 mai    2021 btrfs-map-logical
-rwxr-xr-x 1 root root  426480 14 mai    2021 btrfs-select-super
-rwxr-xr-x 1 root root  422384 14 mai    2021 btrfstune
-rwxr-xr-x 3 root root   38984 20 juil.  2020 bunzip2
-rwxr-xr-x 1 root root  715480 25 juil.  2021 busybox
...
```

2.1.3 Question : Dealing with files rights

 Answer the following questions :

1. Which user and group are owning `/bin` files ?
2. Who can execute this files/programs ?
3. Who has wrote "in" these files ?
4. Can some other user write files in `/bin` folder ?
5. Why `other` and `root` group member can't write "in" these files ?

2.1.4 Concept : Effective rights

 The ability to access a file is respectively evaluated :
— as the user if the user is the file owner ;
— as a member if the user is a member of the group which own the file ;
— as all other user.

2.2 Changing files credentials

2.2.1 Concept : Changing user and group owner

 You can use :
— `chown` to change the owning user (*CHange OWNer*) ;
— `chgrp` to change the owning group (*CHange GRouP*).
When you use one of these tools, you can use the *UID* or *GID* instead of *user* or *group* name.
Use `man chown` or `man chgrp` to view more details.

 You can use `chown` to change even owning user and group in an unique action using `user.group` as an argument (or UID.GID).

2.2.2 Concept : Changing files rights

 You can use `chmod` to change the rights flag (*CHange MODe*), according to :
— the symbolic mode : `chmod [u g o a] [+ - =] [r w x] file` ;
— the numeric mode `chmod right_mask file`.
To understand the numeric mode, you should know that each **3 flags** is assigned to a boolean weight :
— **R** : "**Read**" right, value $2^2 = 4$;
— **W** : "**Write**" right, value $2^1 = 2$;
— **X** : "**eXecute**" right value $2^0 = 1$.
Some speak about *octal mode*, because each class has a value from 0 to 7.
Use `man chmod` to see the details.

2.2.3 Concept : File creation

Everything I create **is mine and belong to my main group too**.

Rights are accorded using a **default value**.

This value is set using a **bitwise binary AND** applied from the inverted mask and the canonical numerical creation permissions.

`umask` command is used to :

— to **print** the current mask, **without arguments** (or -S for symbolic, but inverted) ;

— to **set** the future mask, with a **3 (or 4) digits** value as an argument.

2.2.4 To do : Playing with creation mask

First, print the current mask :

```
$ umask
0007
```

Regarding only `ugo` classes, numerical mask is **007**.

According to the rule :

— files will be created with **660** right, because of the canonical files creation right is **666** ;

— folders will be created with **770** right, because of the canonical folders creation right is **777**.

Let's demonstrate in detail for a file :

```
canonical          666                 rw- rw- rw-
umask              007                 --- --- ---
inverted umask     !007 = 770          rwx rwx ---
binary and         666 & 770 = 660     rw- rw- ---
```

Then, predict what will be the right on a folder, using a paperboard. And verify your predictions using the `mkdir` command.

At now, change the creation mask, using `umask 002`. Make predictions, and verify, creating a regular file and a folder.

Verify your predictions ...

2.2.5 Question : Who is behind the mask ?

Answer the following questions.

— Ubuntu uses 022 as a default `ugo` mask. What is the problem ?

— I set 007 as a default mask in all systems I manage. Why am I doing this ?

— What is wrong with this value ?

— What is the best solution ?