Technische Universität Berlin

Linux III – Users & permissions

Training for Afghan System Administrators

Linux as a multi-user system

Linux – like UNIX which was used on mainframe computers in big organizations - has always been a multi-user system. A multi-user system means that different users can use the system and its resources simultaneously.

Every user that should be able to use the computer needs an account on the system. All user accounts, together with some information about the users (like their full name, their home directory, and some more attributes) are stored in **/etc/passwd** and the encrypted passwords for all users (plus some other password-related configurations) are stored in **/etc/shadow**. These two files work together, and every user has an entry in each of the two files. When a user tries to log in, the system will compare the name and password with the information stored in these files and will then either grant or deny access.

To make administration easier, users of the same type (or access level) can be assigned to groups. Information about the groups and group members are saved **/etc/group**. On many systems, if you create a new user account a group of the same name will be created as well.

The file /etc/shadow

The file /etc/shadow is used to store the passwords and other password related information. Each line represents one user.

- stefan:$ljb1iUaqmB2TvlRdRdI8DugqECln671DwjREhm/wZEjDALuqRHPyKBW4/BDJ70:15300:0:99999:7:::

The order of the fields (separated by ":") is as follows:

- Username
- Encrypted password
- Days since Epoch last password change
- Days until change allowed
- Days before change required
- Days warning for expiration
- Days before account inactive
- Days since Epoch when account expires
- Reserved for future use

Username, password and days since last password change are always filled in, the rest of the fields are optional.

Epoch is a date often found in IT and is 01.01.1970

The file /etc/passwd

The syntax for the file /etc/passwd is:

- username:password field:uid:gid:GECOS field:path to home directory:login shell

Example:

- stefan:x:1001:1001:Stefan Heil, Room FH-518, 03031473821:/home/stefan:/bin/bash

The “x” in the password field means that the password is stored in the file /etc/shadow. If you need to (temporarily) deactivate the account, you can put an “*” instead and the user will not be able to log in.

The GECOS field is a list of comma separated user details, such as full name, room / building number, phone number, other contact details.

The login shell field specifies which shell will be used for logins. Besides the bash shell, there are many others (sh, ash, zsh, ksh, etc...), but bash is what you will see on most systems

The file /etc/shadow

The file /etc/shadow is used to store the passwords and other password related information. Each line represents one user.

- stefan:$ljb1iUaqmB2TvlRdRdI8DugqECln671DwjREhm/wZEjDALuqRHPyKBW4/BDJ70:15300:0:99999:7:::

The order of the fields (separated by ":") is as follows:

- Username
- Encrypted password
- Days since Epoch last password change
- Days until change allowed
- Days before change required
- Days warning for expiration
- Days before account inactive
- Days since Epoch when account expires
- Reserved for future use

Username, password and days since last password change are always filled in, the rest of the fields are optional.

Epoch is a date often found in IT and is 01.01.1970

Examples: adding, removing, modifying users and groups

- useradd -m -c “Stefan Heil” -G GROUPNAME -s /bin/bash stefan # adds user stefan, creates a default home dir, adds a full name, and adds him to group GROUPNAME
- usermod -G GROUPNAME stefan # adds stefan to group GROUPNAME
- usermod -L stefan # deactivates stefan's account
- userdel -r -f stefan # deletes user stefan and his home dir (even if there are files from other users in the home dir)
- groupadd NEWGROUP -g 1234 -p GROUP_PW # adds a new group NEWGROUP with a GID of 1234 and sets a group password of GROUP_PW
- groupmod GROUPNAME -p # changes the password of the group GROUPNAME
- gpasswd -r GROUPNAME # disables password for GROUPNAME
- groupdel GROUPNAME # deletes the group GROUPNAME

Changing passwords and password settings

Users can change their own password with the ***passwd*** utility. Only root can change the passwords of other users.

- Examples:
  - passwd # to change your own passwd
  - passwd USERNAME # to change another users password (works only as root!)

Root can use the ***chage*** utility to change all user's password settings, such as expiration date, warning time before expiration, days between mandatory password changes, inactivity times, etc.. Regular users can use the chage utility to view their password settings and policies

- Examples:
  - chage # shows all password policies and dates
  - chage -M 30 USERNAME # USERNAME must change pw every 30 days
  - chage -E "2015-12-31" USERNAME # deactivate USERNAME's account on 31122015

Keeping /etc/passwd & /etc/shadow consistent

Some examples:

- pwconv:: convert from using /etc/passwd for user passwords to using /etc/shadow
- pwunconv: convert from using /etc/shadow to using /etc/passwd
- grpconv / grpunconv: same as above, but for group passwords
- pwck: consistency check for /etc/passwd and /etc/shadow
- grpck: same as above but for /etc/group and /etc/gshadow

The need for a filesystem permission model

To to keep some order on a multi-user system, users have their own home directories (e.g. /home/USER1, /home/USER2) where personal files and configurations are stored. Inside these directories, users can usually read and write all files, because they are the owners of the files. However, there are thousands of directories and millions of other files spread throughout the filesystem. In order to define what each user is allowed to do with these files, a permission model is needed.

For example, while non-root users are allowed to *read* and *execute* many files, they are usually not allowed to *write* files outside of their home directory (except in /tmp). This makes sure that users can't delete or modify important system files. For security reasons, some files and directories are not even readable for other users.

Read, write and execute

The read permission grants the ability to read a file. When set for a directory, this permission grants the ability to read the names of files in the directory, but not to find out any further information about them such as contents, file type, size, ownership, permissions.

The write permission grants the ability to modify a file. When set for a directory, this permission grants the ability to modify entries in the directory. This includes creating files, deleting files, and renaming files.

The execute permission grants the ability to execute a file. This permission must be set for executable programs, including shell scripts, in order to allow the operating system to run them. When set for a directory, this permission grants the ability to access file contents and meta-information if its name is known, but not list files inside the directory, unless read is set also.

Filesystem permission model – who can do what?

Each file /directory has certain permissions and information about the owner and group the file belongs to attached to it. These permissions are first separated into 3 different scopes or classes.

1. The first class concern the **permissions for the user** of the file,
2. the second class concern the **permissions for the group** the file belongs to,
3. and the third class concern the **permissions for all others users**

Each class is again separated into 3 different permissions: **read(r), write(w)** and **execute(x).**

This gives each file or directory 3 x 3 possible permissions: rwx(owner), rwx(group), rwx(others)

Setuid, setgid and the sticky bit

Unix-like systems typically employ three additional modes. These special modes are for a file or directory overall, not by a class.

The **set user ID, setuid, or SUID mode**: when a file with setuid is executed, the resulting process will assume the effective user ID given to the owner class. This enables users to be treated temporarily as root (or another user).

The **set group ID, setgid, or SGID mode**: when a file with setgid is executed, the resulting process will assume the group ID given to the group class. When setgid is applied to a directory, new files and directories created under that directory will inherit their group from that directory.

The **sticky bit mode**: on a directory, the sticky bit prevents users from renaming, moving or deleting contained files owned by users other than themselves, even if they have write permission to the directory. Only the directory owner and superuser are exempt from this.

Symbolic notation

There are several ways by which Unix permissions are represented. The most common form is symbolic notation as shown by 'ls -l':

```
tp stefan # ls -l /home/
drwxr-xr-x  5 root   root    4096 28. Nov 2011  .
drwxr-xr-x 24 root   root    4096  9. Jun 09:33 ..
drwxrwx--- 70 stefan stefan  4096 25. Jun 22:36 stefan
```

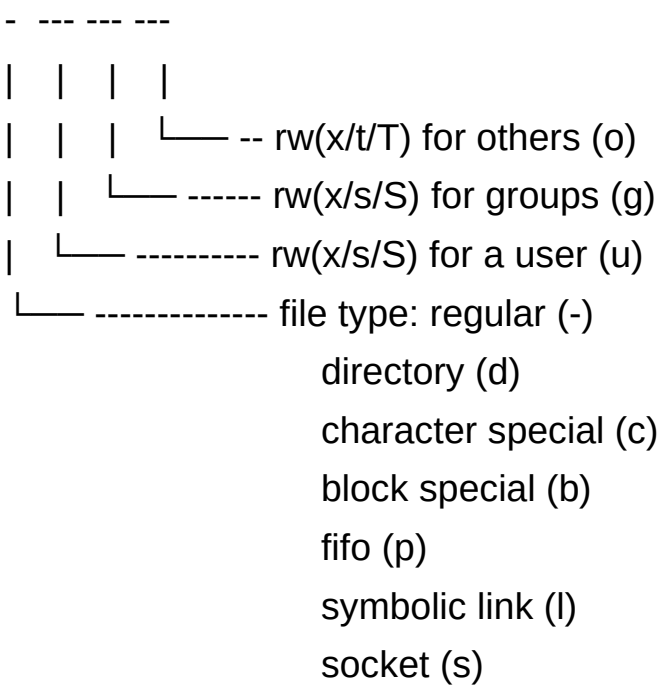
The very first character shows either “-” for files, “d” for directories or “l” for links (etc...). The next 9 characters represent the 3 different permissions (read, write, execute) for the 3 classes (user, group, others)

r if reading is permitted, - if it is not

w if writing is permitted, - if it is not.

x if execution is permitted, - if it is not. The third character additionally represents the setuid (s/S in the first triplet), setgid(s/S in the second triplet) or sticky bit (t/Tin the third triplet) modes.

Symbolic notation (hooray for ascii art!)



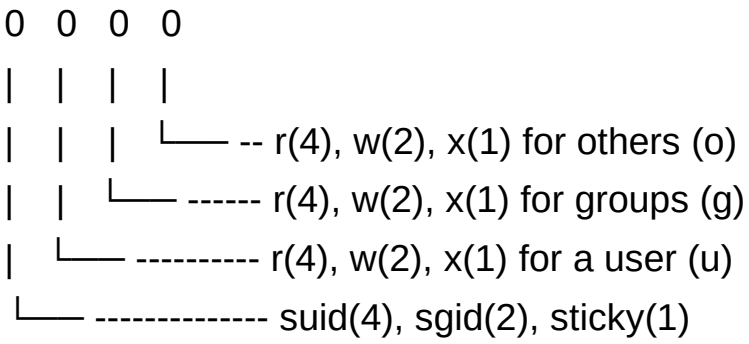
Numeric / octal notation

Another method for representing Unix permissions is an octal (base-8) notation as shown by 'stat -c %a'. This notation consists of at least three digits. Each of the three rightmost digits represents a different component of the permissions: owner, group, and others. Each of these digits is the sum of its component bits in the binary numeral system.

- The read bit adds 4 to its total (in binary 100),
- The write bit adds 2 to its total (in binary 010), and
- The execute bit adds 1 to its total (in binary 001).

These values never produce ambiguous combinations; each sum represents a specific set of permissions. More technically, this is an octal representation of a bit field – each bit references a separate permission, and grouping 3 bits at a time in octal corresponds to grouping these permissions by user, group, and others.

Numeric / octal notation (hooray for ascii art)



Symbolic and numeric notation – some examples...

Symbolic	Octal	Human language
----------	0000	no permissions
---x--x--x	0111	execute
--w--w--w-	0222	write
--wx-wx-wx	0333	write & execute
-r--r--r--	0444	read
-r-xr-xr-x	0555	read & execute
-rw-rw-rw-	0666	read & write
-rwxrwxrwx	0777	read, write, & execute
-rwsrwxrwx	4777	read, write, execute & setuid set
-rwSrwxrwx	4677	read, write & setuid set (but NOT executable)
-rwxrwsrwx	2777	read, write, execute & setguid set
-rwxrwSrwx	2767	read, write & setgid set (but NOT executable)
-rwxrwxrwt	1777	read, write, execute & sticky bit set
-rwxrwxrwT	1776	read, write & sticky bit set (but NOT executable)

Changing ownership

To change the owner of a file / directory, we use the chown utility:

- chown USERNAME /path/to/file_or_directory # changes only owner
- chown USERNAME:GROUPNAME /path/to/file_or_directory # changes owner and group
- chown USERNAME\: /path/to/file_or_dir # changes owner and group to same

To change the group of a file / directory, we use the chgrp utility:

- chrgp GROUPNAME /path/to/file_or_directory

Change file permissions / access modes

To change the permissions or access modes of a file, we use the chmod utility.

- Symbolic mode:
  - u: user/owner, g: group, o: others, a: all
  - +: give permission, -: take permission, =: set exactly to this mode
  - Examples:
    - chmod u+w /path/to/file # gives user (owner) write permission
    - chmod g+r /path/to/file # gives the group read permission
    - chmod o-w /path/to/file # takes write permission away from others
    - chmod a+x /path/to/file # gives all the execute permission
    - chmod a+r,a+x /path/to/file # gives all the read and execute permissions

- Octal mode:
  - Examples:
    - chmod 664 /path/to/file # gives rw to owner and group; read to others
    - chmod 755 /path/to/file # gives rwx to owner; rx to group and others