



DevOps external course

Linux users and files

Lecture 5.5

Module 5 **Linux Essentials**

Serge Prykhodchenko



USERS IN LINUX

Agenda

- 1. General information about users in Linux. Main configuration files.
- 2. User management, commands adduser, useradd, userdel, usermod.
- 3. Changing access rights to files and directories (chmod, chgrp).
- 4. Changing the owner of a file and directories (chown)
-

Objective: To get knowledge with access rights in Linux OS, as well as with privileges. Learn how to change the owner of files permissions and directories. Learn the basic mechanisms of user management

SECURITY SYSTEM IMPLEMENTATION MECHANISMS



Traditional unix access control

The earliest and simplest versions of UNIX never had a single "hub" of access control. However, there were general rules that influenced on the design of systems.

- Objects (such as files and processes) have owners. Owners possess extensive (but not necessarily unlimited) control over their objects.
- You are the owners of the new objects you create.
- The special **root** user, known as the superuser, can act as the owner of any object in the system.
- Only the superuser can perform special administrative operations.

Traditional unix access control

File System Access Control

Every file in the traditional UNIX-like system belongs to its owner and a group, sometimes referred to as a “group owner”. The file owner has a special privilege that is not available to other users of the system: he is allowed to change the access rights to the file. In particular, the owner can set the permissions so that no one else can access the file. A file is always owned by one person, while an owner group can log in multiple users. Traditionally, information about groups was stored in the **/etc/group** file, but now it is more often stored on a network server NIS (Network Information Service) or LDAP (Lightweight Directory Access Protocol)).

Process ownership

The owner can send signals to the process and also lower its priority. Several identifiers are associated with processes: real, current and saved user ID; the real, current and saved group ID, and on Linux the “filesystem user ID”, which is only used to determine file permissions. Generally speaking, real numbers are used to account for the use of system resources, and current numbers are used to indicate access rights. In most cases, the real and current identifiers are the same.

Superuser account

The UNIX root account is owned by the all-powerful admin user, known as superuser, although his real name is “root”. The defining characteristic of a superuser account is the value UID equal to zero. Nothing prohibits changing the name of this account or creating another record with a zero ID, but such actions are not good. Their consequence will be the emergence of new gaps in the security system, as well as the confusion and anger of other administrators who will have to deal with the specifics of configuring such a system.

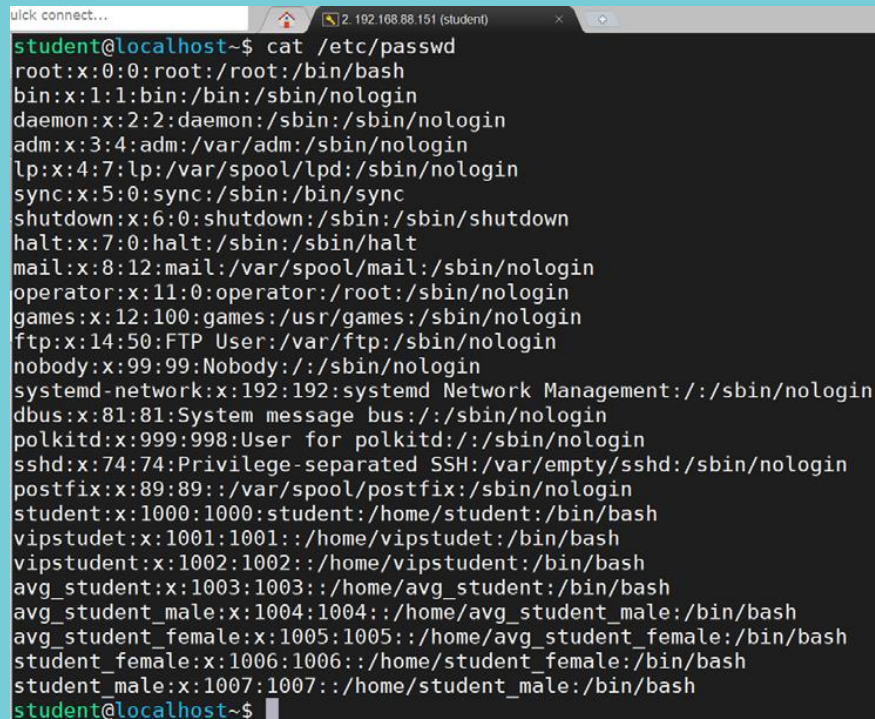
A traditional UNIX system allows the superuser (i.e. any process whose current user ID is zero) execute on the file, or process any valid operation.

Here are examples of operations that are only available to the superuser:

- changing the root directory of a process using the chroot command;
- creation of device files;
- setting the system clock;
- increasing the limits for the use of resources and raising the priorities of processes;
- setting the network name of the computer;
- configuring network interfaces;
- opening privileged network ports (whose numbers are less than 1024);
- system shutdown.

File structure of `/etc/passwd`.

User management starts when there are available, i.e. first you need to add users to the new system. There are several ways to do this task. Each of the methods modifies the contents of the password file `/etc/passwd`. The `/etc/passwd` file format is the same for all Linux distros.

A terminal window with a dark background and light green text. The prompt is 'student@localhost~\$'. The command 'cat /etc/passwd' has been executed, displaying the contents of the password file. Each line represents a system or user account in the format 'username:x:uid:gid:gecos:home:shell'. The accounts listed are root, bin, daemon, adm, lp, sync, shutdown, halt, mail, operator, games, ftp, nobody, systemd-network, dbus, polkitd, sshd, postfix, student, vipstudet, vipstudent, avg_student, avg_student_male, avg_student_female, student_female, and student_male. The terminal window has a title bar with a browser icon and the text 'quick connect...'.

```
student@localhost~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/ssh:/sbin/nologin
postfix:x:89:89:/:/var/spool/postfix:/sbin/nologin
student:x:1000:1000:student:/home/student:/bin/bash
vipstudet:x:1001:1001:/:/home/vipstudet:/bin/bash
vipstudent:x:1002:1002:/:/home/vipstudent:/bin/bash
avg_student:x:1003:1003:/:/home/avg_student:/bin/bash
avg_student_male:x:1004:1004:/:/home/avg_student_male:/bin/bash
avg_student_female:x:1005:1005:/:/home/avg_student_female:/bin/bash
student_female:x:1006:1006:/:/home/student_female:/bin/bash
student_male:x:1007:1007:/:/home/student_male:/bin/bash
student@localhost~$
```


File structure of /etc/passwd.

This file contains lines of the following form, separated by colons:

username: pswd: uid: gid: uid comments: directory: shell

Where:

username - username

pswd - password

uid - unique identifier of the user within the system

gid - unique identifier of the group within the system to which the user belongs

uid comments - comment, extended user description, for example, full name

directory - user's home directory

shell - program name - the user's command interpreter

The username must not contain a colon character. In addition, it is not recommended to use a period (.)

In a name, or start it with + or -.

The password field can be empty, indicating that no password is required to register a user. It can also contain not more than 13 characters of the encrypted version of the password.

uid is a simple numeric designation for an individual user. This is usually a positive number not more than 65535 (sometimes 32-bit). Some identifiers are reserved for special use. These include 0 (root), 1-999 (daemons, pseudo-users, system and reserved users), 1000+ (regular users).

File structure of `/etc/passwd`.

Since `/etc/passwd` is usually readable, hidden password schemes or a shadow password mechanism are usually used for security purposes. Of course, this will redirect encrypted passwords to a restricted file that may contain additional information (`/etc/shadow`).

The bad password category includes dictionary words, login name, missing password, or information contained in the user's comment field.

```
cat: /etc/shadow: Permission denied
student@localhost~$ sudo !!
sudo cat /etc/shadow
[sudo] password for student:
root:$6$zrKhRuWl0yNiZtbL$lhB0AWrWSLwgNrFFXpJ3lvWcw7B0H1kPdx299WYB5TL0STQK0NVsunN1JC/ZZpduL
ESyBt2yI1TWQ5EEpyQG0:!:0:99999:7:::
bin:!:17834:0:99999:7:::
daemon:!:17834:0:99999:7:::
adm:!:17834:0:99999:7:::
lp:!:17834:0:99999:7:::
sync:!:17834:0:99999:7:::
shutdown:!:17834:0:99999:7:::
halt:!:17834:0:99999:7:::
mail:!:17834:0:99999:7:::
operator:!:17834:0:99999:7:::
games:!:17834:0:99999:7:::
ftp:!:17834:0:99999:7:::
nobody:!:17834:0:99999:7:::
systemd-network:!!:18228:!:!:!:
dbus:!!:18228:!:!:!:
polkitd:!!:18228:!:!:!:
sshd:!!:18228:!:!:!:
postfix:!!:18228:!:!:!:
student:$6$6AjQC0K03VPtTjSE$rEkXmabZ8f1uQg31QLaSv7hM0eNGc7Av461itQHEkk82Sag2.maP6PvpvDq3x
leaXFAtGcHDUD/FLYnDy0z./:!:0:99999:7:::
vipstudent:!!:18522:0:99999:7:::
vipstudent:!!:18522:0:99999:7:::
avg_student:!!:18522:0:99999:7:::
avg_student_male:!!:18522:0:99999:7:::
avg_student_female:!!:18522:0:99999:7:::
student_female:!!:18522:0:99999:7:::
student_male:!!:18522:0:99999:7:::
student@localhost~$
```

The file structure `/etc/group`

The `/etc/group` file applies to the general security scheme for Unix-like systems: user, group, and file access. The format for this file is as follows:

group_name:password:group_id:list

The **group_name** field contains the text name for the group. The **password** field contains the encrypted password of this group. If this field is empty, then no password is required. The **group_id** field contains a unique identifier for this group. The **list** field contains a comma separated list of users belonging to this group. Users do not need to be included in the list of those groups that are specified as their primary in the `/etc/passwd` file.

```
student@localhost~$ cat /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:
mem:x:8:
kmem:x:9:
wheel:x:10:student
cdrom:x:11:
mail:x:12:postfix
man:x:15:
dialout:x:18:
floppy:x:19:
games:x:20:
tape:x:33:
video:x:39:
ftp:x:50:
lock:x:54:
audio:x:63:
nobody:x:99:
users:x:100:
utmp:x:22:
utempter:x:35:
input:x:999:
systemd-journal:x:190:
systemd-network:x:192:
dbus:x:81:
polkitd:x:998:
ssh_keys:x:997:
sshd:x:74:
postdrop:x:90:
postfix:x:89:
student:x:1000:student
vipstudet:x:1001:
vipstudent:x:1002:
avg_student:x:1003:
avg_student_male:x:1004:
avg_student_female:x:1005:
student_female:x:1006:
student_male:x:1007:
student@localhost~$
```

The file structure /etc/group

Pseudo-users. Each of the UNIX variants contains pseudo-user description lines in the password file. These descriptions are never edited. Users of these names are not registered in the system and are only needed to confirm ownership of the processes. The most used are:

daemon - Used by system service processes

bin - Gives ownership of executables command

adm - Owns registration files

nobody - Used by many services

sshd – used by the secure shell server.

Commands for working with user accounts

The creation, modification and deletion of lines in password and group files depends on the version of your operating system. Of course, password files are edited directly from the command line, or graphical utilities are used. These utilities make it easy and comfortable to edit user accounts.

The main commands for working with Linux accounts are `useradd`, `userdel`, and `usermod`, as well as the password file editor `vipw`. The command interface is as follows:

1) ***useradd [-c uid comment] [-d dir] [-e expire] [-f inactive] [-g gid] [-m [-k skel_dir]] [-s shell]***

[-u uid [-o]] username

2) ***userdel [-r] username***

3) ***usermod [-c uid comment] [-d dir [-m]] [-e expire] [-f inactive] [-g gid] [-G gid [, gid]] [-l new username] [-s shell] [-u uid [-o]] username***

Commands for working with user accounts

The main parameters have the following meanings:

username is the login name of the user. This is the only required parameter in all commands.

uid comment is an additional comment about the user with the specified name.

dir - indicates to the user's home directory.

expire - indicates the exact date until registration record are available .

inactive - indicates a continuous number of days without registration in the system before this record is blocked.

gid - defines the id or name of the group to which the user belongs.

new_username - Replaces the old login name.

shell - defines the shell for the command interpreter for the given user.

skel_dir - contains files which must be copied to the new user's home directory.

uid is the unique user identifier associated with this name.

-m - indicates need to create a new home directory (useradd) or move the current one to a new location (usermod).

-o - Allows repeating the same user ID.

-g - Select the main group for the login name.

-G - selects additional groups.

-r - Tells the user's home directory to be moved. If the home directory for the registration entry is out of date, existing files will be migrated to the new directory.

USERADD example

Centos7 [Running] - Oracle VM VirtualBox

```
student@localhost~$ sudo useradd test2
student@localhost~$ sudo passwd test2
Changing password for user test2.
New password:
BAD PASSWORD: The password is a palindrome
Retype new password:
passwd: all authentication tokens updated successfully.
student@localhost~$
```

Press
Alt + F3

Centos7 [Running] - Oracle VM VirtualBox

```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.el7.x86_64 on an x86_64

localhost login: test2
Password:
[test2@localhost ~]$ pwd
/home/test2
[test2@localhost ~]$
```

Password change command **passwd**.

Command ***passwd*** is used to change user passwords. The format of this:

passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]

The meaning of the command options :

-k - Used to update only those passwords that have expired.

-l - blocks the specified user (available only to the administrator). Blocking is performed by adding the ! Prefix to the password.

stdin - accept new password from standard input.

-u - unblock user and remove prefix ! (available only to the administrator).

-d - cancel the password for the user (available only to the administrator). Allows the user to log in without a password and change it themselves.

-n - sets the minimum period in days before changing the password (available only to the administrator).

-x - sets the maximum period in days before changing the password (available only to the administrator).

-w - sets the period in days when the user starts receiving messages about the need to change the password (available only to the administrator).

-i - sets the period in days until the old password is no longer active and the registration record is blocked (available only to the administrator).

-S - displays short information about the state of the password (about its validity period). Available only to the administrator

Access rights hierarchy

`$ls -la`

The first character indicates the file type:

- - regular file;
- d - directory;
- b - block device;
- c - character device;
- l - symbolic link;
- p - pipe (pipe, fifo);
- s - socket.

```
-rw-r--r--  1 root root    338 Nov 18  2014 updatedb.conf
drwxr-xr-x  3 root root   4096 Jul 26 13:02 update-manager
drwxr-xr-x  2 root root   4096 Jul 26 13:02 update-notd.d
drwxr-xr-x  2 root root   4096 Jul 26 13:02 update-notifier
drwxr-xr-x  2 root root   4096 Jul 26 12:55 vim
drwxr-xr-x  4 root root   4096 Jul 26 13:01 vmware-tools
lrwxrwxrwx  1 root root     23 Jul 26 12:55 vtrgb -> /etc/alternatives/vtrgb
-rw-r--r--  1 root root   4942 May  8  2018 wgetrc
drwxr-xr-x  5 root root   4096 Jul 26 12:59 X11
drwxr-xr-x  4 root root   4096 Jul 26 13:00 xdg
drwxr-xr-x  2 root root   4096 Jul 26 13:00 xml
-rw-r--r--  1 root root    477 Jul 19  2015 zsh_command_not_found
```

Access rights hierarchy

Nine characters representation as "**rw xrwx**rw", where some "r", "w" and "x" can be replaced with "-". Symbols reflect the three types of access accepted in Linux - read, write and use - however they are present in triplicate in the shortcut!

Furthermore, any Linux user (process) in relation to any file can act in three roles: as the owner (user), as a member of the group that owns the file (group), and as an outsider (other), has no ownership relations of this file. The attribute string is 3 three **rw x** that describe the file permissions of the owner of this file (the first triplet, "u"), the group that owns the file (the second triplet, "g") and outsiders (the third triplet, "o"). If a letter is missing in any triplet, and instead of it there is a "-", then the user in the corresponding role will be denied the corresponding type of access. s - bit for changing user or group identifier; t - sticky bit flag.

When the relationship between the file and the user who started the process, the role is determined as follows:

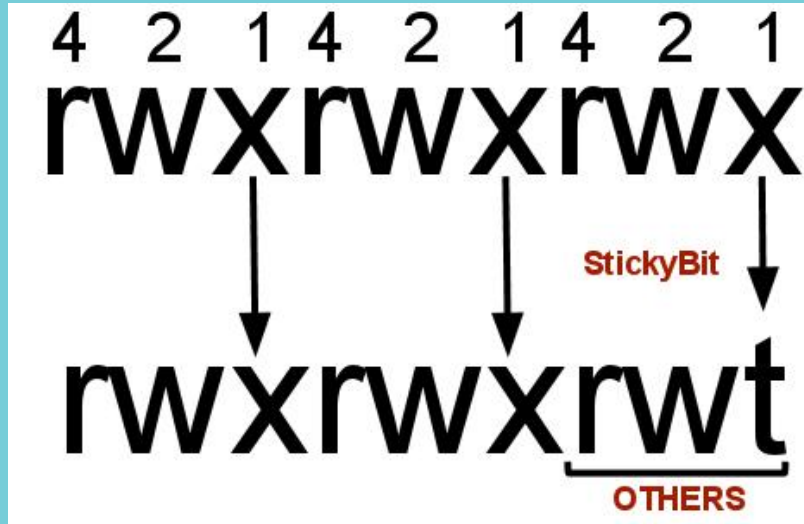
If the UID of the file is the same as the UID of the process, the user is the owner of the file

If the GID of the file matches the GID of any group the user belongs to, he is a member of the group to which the file belongs.

If neither the UID nor the GID of a file overlaps with the UID of the process and the list of groups that the user running it belongs to, that user is an outsider

STICKY BIT

Sticky Bit is mainly used on folders in order to avoid deletion of a folder and its content by other users though they having write permissions on the folder contents. If Sticky bit is enabled on a folder, the folder contents are deleted by only owner who created them and the root user. No one else can delete other users data in this folder (Where sticky bit is set). This is a security measure to avoid deletion of critical folders and their content (sub-folders and files), though other users have full permissions.



STICKY BIT

Example: Create a project (a folder) where people will try to dump files for sharing, but they should not delete the files created by other users.

How can I setup Sticky Bit for a Folder?

Sticky Bit can be set in two ways

1.Symbolic way (t,represents sticky bit)

2.Numerical/octal way (1, Sticky Bit bit as value 1)

Use **chmod** to set Sticky Bit on Folder: **/opt/dump/**

Symbolic way:

chmod o+t /opt/dump/ or chmod +t /opt/dump/

Let me explain above command, We are setting Sticky Bit(+t) to folder /opt/dump by using **chmod** command.

Numerical way:

chmod 1757 /opt/dump/

Here in 1757, 1 indicates Sticky Bit set, 7 for full permissions for owner, 5 for read and execute permissions for group, and full permissions for others.

STICKY BIT

Checking if a folder is set with Sticky Bit or not?

Use **ls -l** to check if the x in others permissions field is replaced by **t** or **T**

For example: /opt/dump/ listing before and after Sticky Bit set

Before Sticky Bit set:

```
ls -ltotal 8-rwxr-xrwx 1 xyz xyzgroup 148 Dec 22 03:46 /opt/dump/
```

After Sticky Bit set:

```
ls -l total 8 -rwxr-xrwt 1 xyz xyzgroup 148 Dec 22 03:46 /opt/dump
```

Permissions

Permissions can be changed using three commands: **chown** (change owner), **chgrp** (change group), and **chmod** with extended parameter format: before the access part (before the "+" or "-" sign), can list the roles "u", "g", "o" and "a" (all, which corresponds to "ugo") for which access is being changed. Besides, when specifying access, you can use "=" instead of "+" and "-", then the specified access methods are allowed for the specified roles, and the unspecified ones are banned.

Instead of a pair of chown commands, the master is the file; chgrp group file you can use one: chown master: group file, which changes both the UID and GID of the file (directory, links, etc.).

chmod never changes permissions on symbolic links

This is not a problem because the rights of symbolic link are never used. However, for each symbolic link given on the command line, chmod changes the permissions of the associated file. However, chmod ignores symbolic links encountered during recursive directory processing.

Octal representation of attributes.

All twelve attributes can be represented as bits of a binary number, equal to 1 if the attribute is set, and 0 if not. The order of the bits is as follows:

`sU | sG | t | rU | wU | xU | rG | wG | | xG | rO | wO | xO`,

where `sU` is SetUID, `sG` is SetGID, `t` is a `t` attribute (`ls -dl` - then the directory is displayed as a file), then three triples of access attributes.

This format can be used in the `chmod` command instead of the construct `"roles = access_types"`, and the number must be written in octal system. For example, the `/ tmp` directory attributes will be equal `1777`, the `/ bin / su` attributes will be `- 4711`, the `/ bin / ls` attributes will be `- 755`, and so on.

The same bitwise representation of attributes regulates the default access rights when creating files and directories. This is done using the `umask` command. The only `umask` parameter is an octal number that specifies the attributes that should not be set on a new file (`666`) or directory (`777`).

For example, `umask 0` will cause files to be created with `"rw-rw-rw-"` attributes and directories `"rwxrwxrwx"`. The `umask 022` command removes write permissions from the default attributes for everyone except the owner (it turns out `"rw-r - r--"` and `"rwxr-xr-x"`, respectively), and with `umask 077` new files and directories become completely are not available (`"rw - - - - -"` and `"rwx - - - - -"`) to everyone except their owners.

<https://www.redhat.com/sysadmin/suid-sgid-sticky-bit>

- Linux traditional rights system
- Linux Security Enhanced rights system
- Turning Commands into a Script
- Q&A

LINUX TRADITIONAL RIGHTS SYSTEM

File types more detailed

The first character indicates the file type:

- - regular file;
- d - directory;
- b - block device;
- c - character device;
- l - symbolic link;
- p - pipe (pipe, fifo);
- s - socket.

```
Last login: Thu Dec 17 01:07:09 2020
student@localhost~$ ls -l /bin/ls /dev/sda /dev/tty /sbin/halt
-rwxr-xr-x. 1 root root 117680 Oct 30 2018 /bin/ls
brw-rw---- 1 root disk 8, 0 Dec 16 17:04 /dev/sda
crw-rw-rw- 1 root tty 5, 0 Dec 17 07:30 /dev/tty
lrwxrwxrwx. 1 root root 16 Nov 28 2019 /sbin/halt -> ../bin/systemctl
student@localhost~$ █

student@localhost~/myPipe$ mkfifo myPipe
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:45 myPipe
student@localhost~/myPipe$ █

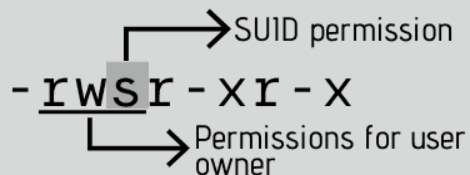
student@localhost~/myPipe$ echo "winter is coming..." > myPipe
student@localhost~/myPipe$ █

student@localhost~/myPipe$ cat myPipe
winter is coming...
student@localhost~/myPipe$ █

student@localhost~/myPipe$ nc -lU mySocket1.sock
eqeqweqweq
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:51 myPipe
srwxrwxr-x 1 student student 0 Dec 17 08:58 mySocket1.sock
srwxrwxr-x 1 student student 0 Dec 17 08:57 mySocket1.sock
student@localhost~/myPipe$ █
```

SUID

When the SUID bit is set on an executable file, this means that the file will be executed with the same permissions as the owner of the executable file.



The diagram shows the permissions `-rwsr-xr-x`. An arrow points from the `s` in `rws` to the text "SUID permission". Another arrow points from the `r` in `sr-x` to the text "Permissions for user owner".

If you look at the binary executable file of the **passwd** command, it has the SUID bit set.

```
student@localhost~/myPipe$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 27832 Jun 10 2014 /usr/bin/passwd
student@localhost~/myPipe$
```

This means that any user running the **passwd** command will be running it with the same permission as root

SUID

Note that a normal user can't change passwords for other users, only for himself/herself. But why? If you can run the **passwd** command as a regular user with the same permissions as root and modify the files like **/etc/passwd**, why can you not change the password of other users?

If you check the code for the **passwd** command, you'll find that it checks the UID of the whose password is being modified with the UID of the user that ran the command. If it doesn't match and if the command wasn't run by root, it throws an error.

The **setuid/SUID** concept is tricky and should be used with utmost cautious otherwise you'll leave security gaps in your system. It's an essential security concept and many commands (like **ping** command) and programs (like **sudo**) utilize it

How to set SUID bit

Symbolic way easier while setting SUID bit. You can use **chmod** command in this way:

chmod u+s file_name

Numeric way. You just need to add a fourth digit to the normal permissions. **The octal number used to set SUID is always 4.**

chmod 4766 file_name

```
student@localhost~/myPipe$ touch test2.txt
student@localhost~/myPipe$ chmod 755 test2.txt
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:51 myPipe
srwxrwxr-x 1 student student 0 Dec 17 08:58 mySocket1.sock
srwxrwxr-x 1 student student 0 Dec 17 08:57 mySocket.sock
-rwxr-xr-x 1 student student 0 Dec 17 09:53 test2.txt
student@localhost~/myPipe$ chmod u+s test2.txt
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:51 myPipe
srwxrwxr-x 1 student student 0 Dec 17 08:58 mySocket1.sock
srwxrwxr-x 1 student student 0 Dec 17 08:57 mySocket.sock
-rwsr-xr-x 1 student student 0 Dec 17 09:53 test2.txt
student@localhost~/myPipe$
```

How to remove SUID bit

Symbolic mode: in **chmod** command **chmod u-s test2.txt**

Numeric mode: with 0 instead of 4 with the permissions you want to set:

chmod 0766 test2.txt

Difference between small s and capital S as SUID bit

If you set the SUID bit for a file without set “executable” attribute, it will show a capital **S**, not small **s**

```
student@localhost~/myPipe$ touch test.txt
student@localhost~/myPipe$ ls
myPipe  mySocket1.sock  mySocket.sock  test.txt
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:51 myPipe
srwxrwxr-x 1 student student 0 Dec 17 08:58 mySocket1.sock
srwxrwxr-x 1 student student 0 Dec 17 08:57 mySocket.sock
-rw-rw-r-- 1 student student 0 Dec 17 09:46 test.txt
student@localhost~/myPipe$ chmod u+s test.txt
student@localhost~/myPipe$ ls -l
total 0
prw-rw-r-- 1 student student 0 Dec 17 08:51 myPipe
srwxrwxr-x 1 student student 0 Dec 17 08:58 mySocket1.sock
srwxrwxr-x 1 student student 0 Dec 17 08:57 mySocket.sock
-rwSrwxr-- 1 student student 0 Dec 17 09:46 test.txt
student@localhost~/myPipe$
```

How to find all files with SUID set?

If you want to search files with this permission, use **find** command in the terminal with option **-perm**.

(from sudo or root)

find / -perm /4000

```
student@localhost~/myPipe$ sudo find / -perm /4000
[sudo] password for student:
find: '/proc/21364/task/21364/fd/6': No such file or directory
find: '/proc/21364/task/21364/fdinfo/6': No such file or directory
find: '/proc/21364/fd/5': No such file or directory
find: '/proc/21364/fdinfo/5': No such file or directory
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/mount
/usr/bin/chage
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/su
/usr/bin/umount
/usr/bin/sudo
/usr/bin/crontab
/usr/bin/pkexec
/usr/bin/passwd
/usr/sbin/pam_timestamp_check
/usr/sbin/unix_chkpwd
/usr/sbin/usernetctl
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/libexec/dbus-1/dbus-daemon-launch-helper
/home/student/myPipe/test2.txt
student@localhost~/myPipe$
```

SGID

SGID is similar to SUID. With the SGID bit set, any user executing the file will have same permissions as the group owner of the file.

It's benefit is in handling the directory. When SGID permission is applied to a directory, all sub directories and files created inside this directory will get the same group ownership as main directory (not the group ownership of the user that created the files and directories).

How to set SGID?

Set the SGID bit in symbolic mode like this:

chmod g+s directory_name

The numeric way. You just need to add a fourth digit to the normal permissions.

The octal number used to SGID is always 2.

chmod 2775 folder2

How to remove SGID bit?

Use the -s instead of +s like this:

chmod g-s folder

Removing SGID is the same as removing SUID.

Use the additional 0 before the permissions you want to set:

chmod 0755 folder

SUID, SGID, Sticky Bit

```
-rwxrwxrwx 1 serge serge 0 Jul 28 19:24 testperm.txt
serge@ubuntuTXT:~/dir1$ chmod u+s,g+s testperm.txt
serge@ubuntuTXT:~/dir1$ ls -l
total 0
-rwsrwsrwx 1 serge serge 0 Jul 28 19:24 testperm.txt
serge@ubuntuTXT:~/dir1$ chmod o+s testperm.txt
serge@ubuntuTXT:~/dir1$ ls -l
total 0
-rwsrwsrwx 1 serge serge 0 Jul 28 19:24 testperm.txt
serge@ubuntuTXT:~/dir1$ chmod +t testperm.txt
serge@ubuntuTXT:~/dir1$ ls -l
total 0
-rwsrwsrwt 1 serge serge 0 Jul 28 19:24 testperm.txt
serge@ubuntuTXT:~/dir1$ chmod 0644 testperm.txt
serge@ubuntuTXT:~/dir1$ ls -l
total 0
-rw-r--r-- 1 serge serge 0 Jul 28 19:24 testperm.txt
```

Permissions in examples

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

SetUID	SetGID	Sticky Bit	Read User	Write User	eXec User	Read Group	Write Group	eXec Group	Read Other	Write Other	Exec Other
0	0	0	1	1	1	1	0	1	1	0	1
0			7			5			5		
			r	w	x	r	-	x	r	-	x

Permissions in examples

SetUID	SetGID	Sticky Bit	Read User	Write User	eXec User	Read Group	Write Group	eXec Group	Read Other	Write Other	Exec Other
0	0	0	1	1	1	1	0	1	1	0	1
0			7			5			5		
			r	w	x	r	-	x	r	-	x

`chmod 777 fil.txt = chmod u+rwx,g+rwx,o+rwx fil.txt`

`-rwxrwxrwx fil.txt`

`chmod 644 fil.txt = u-x,g-wx,o-wx fil.txt`

`-rw-r--r- fil.txt`

Chattr Command Attributes; append and immutable

The **chattr** command supports several attribute, most of them are less significant and rarely used in file system management. For initial stage only two attributes are important; append and immutable.

Before we understand append and immutable attributes in details, let's have a quick look on two important commands that are used to manage and list the attributes.

chattr command

This command is used to set or unset the attributes. This command uses following syntax.

#chattr [operator] [attribute flag] [filename]

In above command,

chattr: - This is the main command.

Operator: - There are three operators; + (add), - (remove) and = (keep). 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.

attribute flag: - This is an attribute which we want to update.

filename: - This is the name of file which attribute we want to change

Chattr Command Attributes; append and immutable

```
student@localhost~/myPipe$ touch immortal
student@localhost~/myPipe$ lsattr
lsattr: Operation not supported While reading flags on ./myPipe
lsattr: Operation not supported While reading flags on ./mySocket.sock
lsattr: Operation not supported While reading flags on ./mySocket1.sock
----- ./test2.txt
----- ./immortal
student@localhost~/myPipe$ chattr +a immortal
chattr: Operation not permitted while setting flags on immortal
student@localhost~/myPipe$ sudo !!
sudo chattr +a immortal
[sudo] password for student:
\student@localhost~/myPipe$ lsattr
lsattr: Operation not supported While reading flags on ./myPipe
lsattr: Operation not supported While reading flags on ./mySocket.sock
lsattr: Operation not supported While reading flags on ./mySocket1.sock
----- ./test2.txt
-----a----- ./immortal
student@localhost~/myPipe$ rm immortal
rm: cannot remove 'immortal': Operation not permitted
student@localhost~/myPipe$
```

```
student@localhost~/myPipe$ sudo chattr -a immortal
student@localhost~/myPipe$ sudo chattr +i immortal
student@localhost~/myPipe$ lsattr
lsattr: Operation not supported While reading flags on ./myPipe
lsattr: Operation not supported While reading flags on ./mySocket.sock
lsattr: Operation not supported While reading flags on ./mySocket1.sock
----- ./test2.txt
-----i----- ./immortal
student@localhost~/myPipe$
```

<https://www.computernetworkingnotes.com/rhce-study-guide/how-to-set-immutable-sticky-bit-with-chattr-command.html>

Add User to wheel group in CentOS 7

In CentOS 7 members of the wheel group can run linux commands with sudo privileges. To add a user to the wheel group in CentOS 7 we can use either **usermod** or **gpasswd** command.

Add User to wheel group using usermod command

In **usermod** command **-G** option use to specify the group that user wants be added.(if **-a** options is not used user will be removed from other groups he is already a member of).

usermod -a -G wheel username

Example:

usermod -a -G wheel test2

The above example will add the user test2 to the CentOS 7 **wheel** group.

<https://elearning.wsldp.com/pcmagazine/add-user-to-wheel-group-centos-7/>

LINUX SECURITY ENHANCED RIGHTS SYSTEM

Using the "setuid" and "setgid" Bits

The traditional UNIX access control system is supplemented by a change system permissions, which is implemented by the kernel in cooperation with the file system, which allows specially prepared files to be executed using higher-level privileges (usually superuser privileges). This mechanism allows developers and administrators to create conditions for unprivileged users in which they can perform privileged operations. The fact is that there are two special bits set in the rights mask access to the file: "setuid" (Set User ID - bit change user ID) and "Setgid" (Set Group ID - bit for changing the group identifier). If you run an executable file that has one of these bits set, then the current IDs of the process being created are the IDs of the file owner, not the IDs of the user who launched the program. The change of authority is valid only for the duration of the program. For example, users should be able to change their passwords. But since passwords are stored in a secure */etc/shadow* file, users need to use the `passwd` command with `setuid` privileges to "strengthen" your privileges. The importance of the word "acceptable" should be emphasized. Some operations (for example, launching a file for which the execution bit is not set) are prohibited even by the superuser. The `passwd` command checks who is running it and, depending on the result, configures its behavior accordingly, so regular users can only change their own passwords, and the superuser can change any. (This, by the way, another example of a "special case" in UNIX access control system - the rules are built into the `passwd` command code.)

Obvious disadvantages of this model

- From a security point of view, the superuser represents the only account an entry that could cause a potential system failure. If the superuser compromises himself, the integrity of the entire system can be compromised. A cracker in this case can cause colossal harm to the system.
- The only way to share special superuser privileges is in writing setuid programs. Unfortunately, as the endless stream of security updates on the Internet shows, it is difficult to write truly secure software. Also, you shouldn't write programs whose purpose can be summed up as “I wish these three users could perform backup tasks on the file server.”
- The security model is not robust enough for network use. No computer that an unprivileged user has physical access to can guarantee that it accurately represents the running processes. Who can say that such-and-such user reformatted the disk and did not install his own copy of Windows or Linux with his own UIDs?
- Many security-intensive environments have conventions that simply cannot be enforced with traditional UNIX security systems. For example, US government standards require computer systems to prohibit privileged users (such as those in the highest security clearance category) from publishing high-profile documents at a lower security level.
- Since many of the rules related to access control are embedded in the code of individual commands and daemons, it is impossible to override system behavior without modifying the source code and recompiling the programs. But this is not really practiced.
- There is minimal support for tracking user activity. You can easily find out which groups a user belongs to, but you are not able to determine exactly what actions a member is allowed to do in those groups.

Role Based Access Control

Role-based access control (RBAC) is a theoretical model formalized in 1992 by David Ferraiolo and Rick Kuhn. This model is based on the idea of adding a layer of indirection to the access control mechanism. Rather than assigning permissions directly to users, they are assigned to intermediate constructs called “roles,” and roles are in turn assigned to users. In order to make an access control decision, a special library lists the roles of the current user and finds out if at least one of these roles has the appropriate authority.

There are some similarities between roles and UNIX groups, and there is no one opinion as to whether these constructs differ in substance. In practice, roles are more useful than groups because the systems in which they are implemented allow them to be used outside the context of the filesystem. Roles can also have hierarchical relationships with each other, which greatly simplifies administration.

For example, you could define the role of “senior administrator” who has all the powers of “administrator”, as well as additional powers X, Y and Z

SELinux: Linux Systems with Enhanced Security

The SELinux operating system (as a project) was developed by the US National Security Agency (NSA), but from the end of 2000 it was transferred to the open source developers. SELinux is included in the Linux kernel (since version 2.6) and therefore is currently available in most distributions (but often in a not fully functional state).

SELinux is an implementation of a mandatory access control (MAC) system in which all privileges are assigned by administrators.

In a MAC environment, users cannot delegate their rights and cannot set access control parameters on objects that they (users) own. And therefore, such an operating system is more suitable for nodes with special requirements

Pluggable Authentication Modules

Pluggable Authentication Modules (**PAM**) constitute an authentication technology, not an access control technology. In others in words, the PAM technology is designed to look for an answer not to the question: “Does the user have the right X perform operation Y? ”. Answer to the question: “ How do you know that this is really user X? ” PAM technology is an important link in the access control chain in most systems.

Kerberos cryptographic authentication network protocol

Like PAM, Kerberos is designed to solve problems of authentication, not access control. (It is named after the three-headed dog that protected the entrance to the realm of Hades - Cerberus, or Kerberos.) But if PAM can be called a structural shell for authentication, then Kerberos is a specific method of authentication. The Kerberos implementation uses a trusted (third party) server to perform network-wide authentication tasks. Instead of selfauthentication on your computer, you provide your credentials (tickets) to the Kerberos service, and it gives you cryptographic credentials that you can present to other services as proof of your identity.

Access Control Lists

File system access control is an essential part of UNIX and Linux systems, and therefore its improvement is the primary goal of the developers of these systems. Special attention has been paid to support for access control lists (ACLs) as a generalization of the traditional model of **user/group/“everyone else”** privileges set for multiple users and groups at once.

ACLs are part of the filesystem implementation, so the filesystem you are using must provide explicit support for them. Almost all UNIX and Linux file systems today support ACLs **in one form or another**.

Real World Access Control

Despite the excellent operating capabilities described above, systems, in most nodes for system administration tasks are still the superuser account is used. Many complaints about the traditional system have real ground, but there are serious problems inherent in the alternatives. Therefore, additional software tools such as sudo are of particular importance, which to some extent allow bridge the gap between ease of use and safety.

Opportunities are often used to make decisions in special circumstances. POSIX or role-based access controls (for example, when you need to allow everyone in a particular department to reinstall a printer or daemon), while your administrative team continues to rely on sudo and the superuser account for day-to-day tasks.

In some cases, it is necessary for nodes to use such powerful and “shockproof” systems as SELinux. Since superuser access is a prerequisite for system administration and a central point for system security, it is very important to properly use the rights and responsibilities of the root account.

Su command: replace user id

We recommend that you access the root account using the su command. With no arguments, it prompts for the superuser password and then starts the command interpreter as root. The interpreter will run in privileged mode until it exits.

The **su** command does not record the actions performed in the interpreter environment, but adds an entry to the log file indicating who logged on as superuser and when.

The **su** command can also substitute other user names for root

```
# cp /usr/local/lib/skel/. [a-zA-Z]* ~tyler
# chown tyler:staff ~tyler/. [a-zA-Z]*
# chmod 600 ~tyler/. [a-zA-Z]*

# chown tyler:staff ~tyler/.*
```

The sudo utility: a limited version of the su command

If the root account is available to the Administrators group, then you cannot uniquely identify WHO and WHAT is doing. Therefore, the sudo utility is used to obtain superuser rights. The **sudo** utility takes a command line as an argument, which must be executed as root (or another authorized user). The utility accesses the **/etc/sudoers** file, which contains a list of users who have permission to execute it, and a list of commands that they can enter on a particular computer. If the requested command is allowed, the sudo utility prompts the user for his own password and runs the command as the superuser.

About pseudo-users

Only the root user has a special status for the Linux kernel. There are, however, a few more pseudo-user accounts that are used for system purposes. These bogus accounts can be identified by UID values, which are typically less than 100. Typically, accounts with UIDs less than 10 belong to the system, and UIDs 10 to 100 are reserved for pseudo-users associated with special programs.

The passwords of these pseudo-users in the **/etc/shadow** file are usually replaced with an asterisk, so that you cannot log in with a service name. To defend against remote login attacks (when SSH key files are used instead of passwords), specify **/bin/false** or **/bin/nologin** as shells (instead of **/bin/bash** or **/bin/sh**).

Files and processes that are part of the operating system but should not owned by root, sometimes passed into the ownership of bin users or daemon. It is believed that this will help avoid the risk associated with acting from superuser name. However, due to the inconclusiveness of this argumentation in modern systems simply use the root account.

On some systems, the sys user owns special files (images kernel memory) that are stored in the **/dev** directory. Few programs have access to these files, and they all change the current user ID to sys. Sometimes a kmem or sys group is created instead of the sys user.

TURNING COMMANDS INTO A SCRIPT

The sha-bang

The sha-bang (**#!**) at the head of a script tells your system that this file is a set of commands to be fed to the command interpreter indicated. The **#!** is actually a two-byte magic number, a special marker that designates a file type, or in this case an executable shell script (type `man magic` for more details on this fascinating topic).

Immediately following the sha-bang is a path name. This is the path to the program that interprets the commands in the script, whether it be a shell, a programming language, or a utility. This command interpreter then executes the commands in the script, starting at the top (the line following the sha-bang line), and ignoring comments.

```
#!/bin/sh
```

```
#!/bin/bash
```

```
#!/usr/bin/perl
```

```
#!/usr/bin/tcl
```

```
#!/bin/sed -f
```

```
#!/bin/awk -f
```

Each of the above script header lines calls a different command interpreter, be it `/bin/sh`, the default shell (bash in a Linux system) or otherwise.

Special Characters

- Comments. Lines beginning with a # (with the exception of #!) are comments and will not be executed.

Command separator [semicolon]. Permits putting two or more commands on the same line.

"dot" character match. When matching characters, as part of a regular expression, a "dot" matches a single character.

" partial quoting [double quote]. "STRING" preserves (from interpretation) most of the special characters within STRING.

' full quoting [single quote]. 'STRING' preserves all special characters within STRING. This is a stronger form of quoting than "STRING".

escape [backslash]. A quoting mechanism for single characters. \X escapes the character X. This has the effect of "quoting" X, equivalent to 'X'. The \ may be used to quote " and ', so they are expressed literally.

<https://tldp.org/LDP/abs/html/special-chars.html>

Variables

Let us carefully distinguish between the name of a variable and its value. If variable1 is the name of a variable, then \$variable1 is a reference to its value, the data item it contains.

```
bash$ variable1=23
```

```
bash$ echo variable1
```

```
variable1
```

```
bash$ echo $variable1
```

```
23
```

```
echo "$uninitialized"
```

```
# (blank line)
```

```
let "uninitialized += 5"
```

```
# Add 5 to it.
```

```
echo "$uninitialized"
```

```
# 5
```

<https://tldp.org/LDP/abs/html/varsubn.html>

Exit

The `exit` command terminates a script, just as in a C program. It can also return a value, which is available to the script's parent process.

Every command returns an exit status (sometimes referred to as a return status or exit code). A successful command returns a 0, while an unsuccessful one returns a non-zero value that usually can be interpreted as an error code. Well-behaved UNIX commands, programs, and utilities return a 0 exit code upon successful completion, though there are some exceptions.

Likewise, functions within a script and the script itself return an exit status. The last command executed in the function or script determines the exit status. Within a script, an `exit nnn` command may be used to deliver an `nnn` exit status to the shell (`nnn` must be an integer in the 0 - 255 range).

```
#!/bin/bash
```

```
echo hello
```

```
echo $? # Exit status 0 returned because command executed successfully.
```

```
lskdf # Unrecognized command.
```

```
echo $? # Non-zero exit status returned -- command failed to execute.
```

```
echo
```

```
exit 113 # Will return 113 to shell.
```

```
# To verify this, type "echo $?" after script terminates.
```

<https://tldp.org/LDP/abs/html/exit-status.html>

Tests

An if/then construct tests whether the exit status of a list of commands is 0 (since 0 means "success" by UNIX convention), and if so, executes one or more commands.

There exists a dedicated command called [(left bracket special character). It is a synonym for test, and a builtin for efficiency reasons. This command considers its arguments as comparison expressions or file tests and returns an exit status corresponding to the result of the comparison (0 for true, 1 for false).

With version 2.02, Bash introduced the [[...]] extended test command, which performs comparisons in a manner more familiar to programmers from other languages. Note that [[is a keyword, not a command.

Bash sees [[\$a -lt \$b]] as a single element, which returns an exit status.

The ((...)) and let ... constructs return an exit status, according to whether the arithmetic expressions they evaluate expand to a non-zero value. These arithmetic-expansion constructs may therefore be used to perform arithmetic comparisons.

<https://tldp.org/LDP/abs/html/testconstructs.html>

Operators

= All-purpose assignment operator, which works for both arithmetic and string assignments.

+ plus

- minus

** multiplication*

/ division

*** exponentiation*

<https://tldp.org/LDP/abs/html/ops.html>

QUESTIONS & ANSWERS

A world map with a light beige background and dark beige landmasses. The text "THANK YOU!" is centered over the Atlantic Ocean in a black, serif, all-caps font.

THANK YOU!