Permissions

permissions practice for CSC420 Operating Systems Spring 2024
Lyon College

Marcus Birkenkrahe

April 1, 2024

README

- To code along, download tinyurl.com/permissions-org and save the file as `permissions.org`.
- This section is based on chapter 9 of Shotts, The Linux Command Line (2e), NoStarch Press (2019).
- Code tested with Linux Mint 21.3 Virginia and the Cinnamon 6.0.4 Desktop running on a 2018 Dell Vostro 3470 (six i7-8700 cores).

What is it?

- OS in the UNIX tradition are multi-tasking and multi-user systems.
- If the computer is attached to a network, remote users can log in via `ssh` (secure shell) and operate the computer, including GUIs.
- Multiuser capability of Linux is a deeply embedded OS feature because the first computers were not "personal".
- To make multiuser practical, users (and their data) had to be protected from one another.
- Related topics and commands:

COMMAND	MEANING
`id`	Display user identity
`chmod`	Change a file's mode
`umask`	Set the default file permissions
`su`	Run a shell as another user
`sudo`	Execute a command as another user
`chgrp`	Change a file's group ownership
`passwd`	Change a user's password

Example: a bad experience

- Check the file type of `/etc/shadow` using the `file` command:

```
file /etc/shadow
```

```
/etc/shadow: regular file, no read permission
```

- Try to page the file using the `less` command. Anticipating an error (you have no read permission), redirect `stderr` to `stdout`:

```
less /etc/shadow 2>&1
```

```
/etc/shadow: Permission denied
```

- As regular user, you don't have permission to look at this file. Check if you can at least see who does have permission.

```
ls -l /etc/shadow 2>&1
```

```
-rw-r----- 1 root shadow 1636 Mar 30 10:21 /etc/shadow
```

- You've learnt that the owner (`root`) has read and write access, and members of the `shadow` group have read access. Anyone else has none.
- How would you find out more about `shadow` (without the web)?

```
whatis shadow
man shadow
```

NAME

shadow - shadowed password file

DESCRIPTION

shadow is a file which contains the password information for the system's accounts and optional aging information.

This file must not be readable by regular users if password security is to be maintained.

Each line of this file contains 9 fields, separated by colons (":"), in the following order:

login name
It must be a valid account name, which exist on the system.

encrypted password
This field may be empty, in which case no passwords are required to authenticate as the specified login name. However, some applications which read the /etc/shadow file may decide not to permit any access at all if the password field is empty.

A password field which starts with an exclamation mark means that the password is locked. The remaining characters on the line represent the password field before the password was locked.

Refer to crypt(3) for details on how this string is interpreted.

If the password field contains some string that is not a valid result of crypt(3), for instance ! or *, the user will not be able to use a unix password to log in (but the user may log in the system by other means).

date of last password change
The date of the last password change, expressed as the number of days since Jan 1, 1970.

The value 0 has a special meaning, which is that the user should change her password the next time she will log in the system.

An empty field means that password aging features are disabled.

    minimum password age
The minimum password age is the number of days the user will have to wait before she will be allowed to change her password again.

An empty field and value 0 mean that there are no minimum password age.

    maximum password age
The maximum password age is the number of days after which the user will have to change her password.

After this number of days is elapsed, the password may still be valid. The user should be asked to change her password the next time she will log in.

An empty field means that there are no maximum password age, no password warning period, and no password inactivity period (see below).

If the maximum password age is lower than the minimum password age, the user cannot change her password.

    password warning period
The number of days before a password is going to expire (see the maximum password age above) during which the user should be warned.

An empty field and value 0 mean that there are no password warning period.

    password inactivity period
The number of days after a password has expired (see the maximum password age above) during which the password should still be accepted (and the user should update her password during the next login).

After expiration of the password and this expiration period is elapsed, no login is possible using the current user's password. The user should contact her administrator.

An empty field means that there are no enforcement of an inactivity period.

account expiration date
The date of expiration of the account, expressed as the number of days since Jan 1, 1970.

Note that an account expiration differs from a password expiration. In case of an account expiration, the user shall not be allowed to login. In case of a password expiration, the user is not allowed to login using her password.

An empty field means that the account will never expire.

The value 0 should not be used as it is interpreted as either an account with no expiration, or as an expiration on Jan 1, 1970.

reserved field
This field is reserved for future use.

FILES

/etc/passwd
User account information.

/etc/shadow
Secure user account information.

/etc/shadow-
Backup file for /etc/shadow.

Note that this file is used by the tools of the shadow toolsuite, but not by all user and password management tools.

SEE ALSO

chage(1), login(1), passwd(1), passwd(5), pwck(8), pwconv(8), pwunconv(8), su(1), sulogin(8).

The Unix security model

- In the UNIX security model, a user may *own* files and directories.
- With ownership comes access control (and great responsibility).
- The user can belong to a *group* of one or more users who are given access to files and directories by their owners.
- A user may also grant access rights to everyody (aka the *world*).
- Find out who you are in this model with the command `id`.

```
#id
id | tr ',' '\n'  # ‘tr‘ translates characters

uid=1000(marcus) gid=1000(marcus) groups=1000(marcus)
4(adm)
24(cdrom)
27(sudo)
30(dip)
46(plugdev)
115(lpadmin)
136(sambashare)
```

- When users are created, they are assigned a *user ID* (*uid*), which is mapped to a user name.
- The user is also assigned a *group id* (*gid*) and can be part of other groups.
- In the example **??**, `marcus` is the first user with `uid=1000`. This user is in the group with `gid=1000`, and he is also member of a few other groups:
  1. As member of `adm` (`gid=4`) he can access system logs.
  2. As member of `cdrom` (`gid=24`) he can access CD/DVD ROM drives.

3. As member of `sudo` (`gid=27`) he can become superuser.
4. As member of `dip` (`gid=30`) he can open dial-up modem connections.
5. As member of `plugdev` (`gid=46`) he can manage removable storage.
6. As member of `lpadmin` (`gid=115`) he can manager printers.
7. As member of `sambashare` (`gid=136`) he can share files over network[1].

- The specific output is different for different Linux distros. E.g. Fedora Linux starts numbering uid at 500, Debian/Ubuntu at 1000.
- This information is stored in text files, of course: user accounts in `/etc/passwd`, groups in `/etc/group`.
- Take a look at the last 10 lines of `/etc/passwd` and `/etc/group`:

```
less /etc/passwd | tail

_flatpak:x:121:131:Flatpak system-wide installation helper,,,:/nonexistent:/usr/sb
avahi:x:122:132:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
saned:x:123:133::/var/lib/saned:/usr/sbin/nologin
colord:x:124:134:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nolo
fwupd-refresh:x:125:135:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
hplip:x:126:7:HPLIP system user,,,:/run/hplip:/bin/false
marcus:x:1000:1000:marcus,,,:/home/marcus:/bin/bash
sssd:x:127:137:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin
postfix:x:128:138::/var/spool/postfix:/usr/sbin/nologin
testuser:x:1001:1001:,,,:/home/testuser:/bin/bash

less /etc/group | tail

avahi:x:132:
saned:x:133:
colord:x:134:
fwupd-refresh:x:135:
```

[1]Samba is a free software re-implementation of a networking protocol that enables interoperability (= data exchange) between Unix-like and Windows-like systems - e.g. share files, printers etc.

```
marcus:x:1000:
sambashare:x:136:marcus
sssd:x:137:
postfix:x:138:
postdrop:x:139:
testuser:x:1001:
```

- How many user and group accounts are there? Format the printout so that it looks like this:

```
'/etc/passwd' has .. accounts
'/etc/group'  has .. accounts
```

```
echo "'/etc/passwd' has" $(cat /etc/passwd | wc -l) "accounts"
echo "'/etc/group'  has " $(cat /etc/group | wc -l) "groups"
```

```
'/etc/passwd' has 49 accounts
'/etc/group'  has  80 groups
```

- `/etc/shadow` holds information about the user's password.
- What is the uid of the root user? Use `grep` to get the information about `root` from the file with the `uid` information

  This one returns too many entries:

```
cat /etc/passwd | grep root
```

```
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:117:126:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/n
```

  Need those lines where `root` is the first word:

```
cat /etc/passwd | grep ^root
```

```
root:x:0:0:root:/root:/bin/bash
```

- Can you think about a way to directly get the uid for root?

```
sudo id  # you have to run this in a terminal e.g. M-x shell
```

- You should get this output: `uid=0(root) gid=0(root) groups=0(root)` because `root` is the first account created.

NEXT Reading, Writing, and Executing

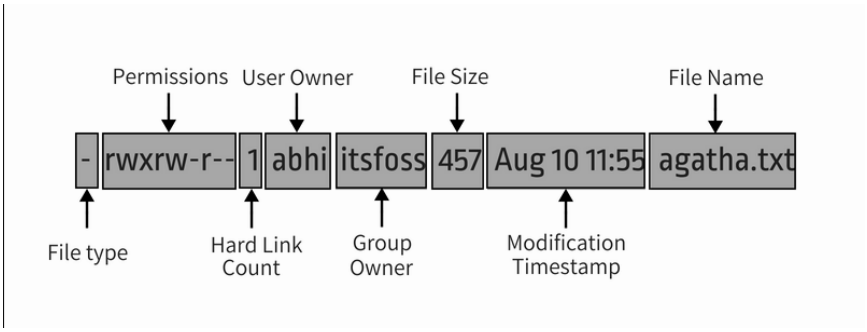
- Access rights to files and directories are defined in terms of **read** access, **write** access, and **execution** access.
- The long listing command `ls -l` shows how this is implemented.
- Create an empty file `foo.txt` using file **redirection**, and then print a long listing of the file.

```
> foo.txt
ls -l foo.txt

-rw-rw-r-- 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

- You've seen this before: now let's analyze the permissions in detail.

File attributes



- The first 10 characters of the listing are *file attributes*. Table gives an overview.

ATTRIBUTE	FILE TYPE
`-`	regular file
`d`	directory
`l`	symbolic link
`c`	character special file
`b`	block special file

- For symbolic links, the remaining attributes are always dummy values. What do you think why that is?

Because a soft/symbolic link is not a file but only a pointer to a file with the real (non-dummy) permissions.

- Create a symbolic link `~/shadow` from `/etc/shadow`:

  1. long-list the symbolic link to see the permissions
  2. execute `less` on the symbolic link

```
rm -rf ~/shadow
ln -s /etc/shadow ~/shadow
ls -l ~/shadow
less ~/shadow
```

```
lrwxrwxrwx 1 marcus marcus 11 Apr  1 22:14 /home/marcus/shadow -> /etc/shadow
/home/marcus/shadow: Permission denied
```

- Which "character special file" did you already encounter? These files handle data as a stream of bytes.

  Answers:

  1. `/dev/null` or the 'bit bucket'
  2. the terminal `tty` used for shell input and output in `/dev`
  3. block special file, e.g. hard drive in `/dev`

```
ls -la /dev/null  # null device
ls -la /dev/tty   # keyboard input output
ls -la /dev/sda   # first disk device
```

```
crw-rw-rw- 1 root root 1, 3 Mar 31 16:36 /dev/null
crw-rw-rw- 1 root tty 5, 0 Apr  1 21:45 /dev/tty
brw-rw---- 1 root disk 8, 0 Mar 31 16:36 /dev/sda
```

- A block special file handles data in blocks, e.g. a hard drive.

File modes

- The remaining nine characters are the *file mode* for the owner, the group, and the world with the permission settings: r=read, w=write, x=execute.

ATTRIBUTE	FILES	DIRECTORIES
r	can be opened	can be listed if x is set (`dr-xr-xr-x`)
w	can be written	files can be created, deleted, renamed if x is set
x	can be run	allows a directory to be entered, e.g. with `cd`

- Table shows the effect that the mode has on files and directories. "Executing" a directory to Unix means "entering" it.
- Scripts(e.g. bash scripts) must also be set readable to be executed.
- Table shows some examples of file attribute settings.

ATTRIBUTE	MEANING
-rwx——	File, readable, writable, executable by owner only. Nobody else can access.
-rw——-	File, readable, writable by owner only. Nobody else can access.
-rw-r–r–	File, readable, writable by owner. Owner's group members & world may read
-rwxr-xr-x	File, readable, writable, executable by owner, can be read and executed by others
-rw-rw—-	File, readable, writable by owner and members of file's owners group only
lrwxrwxrwx	Symbolic link with dummy permissions. Real permissions kept with file pointed to
drwxrwx—	Directory. Owner & members of owner group may enter, create, rename, remove files
drwxr-x—	Directory. Owner may enter, create, rename, delete files here. Group members may enter but cannot write (add or change files).

- Check `/home` where your `$HOME` is. What are the permissions, and what is everybody (the world) allowed to do or see?

```
ls -l /home
```

```
total 8
drwxr-x--- 30 marcus   marcus   4096 Apr  1 22:14 marcus
drwxr-x---  4 testuser testuser 4096 Apr  1 21:35 testuser
```

  Answer: you and your group can enter and read, only you can write to `$HOME`

- Can you (as `$USER`) create a file in `/home`?

```
ls -la /home
id
```

```
total 16
drwxr-xr-x  4 root     root     4096 Mar 30 10:21 .
drwxr-xr-x 20 root     root     4096 Mar 16 13:00 ..
drwxr-x--- 30 marcus   marcus   4096 Apr  1 22:14 marcus
drwxr-x---  4 testuser testuser 4096 Apr  1 21:35 testuser
uid=1000(marcus) gid=1000(marcus) groups=1000(marcus),4(adm),24(cdrom),27(sudo),30
```

> Answer: no! in `/home`, only `root` and `root`'s group have writing rights, and you are not in `root`'s group.

Changing file modes (`chmod`)

- Only file owners and superuser can change the mode of a file or directory using the command `chmod`.
- Mode changes can be specified using octal numbers or symbols. Which you use is a matter of taste and upbringing.

Changing file modes with octal numbers

- Octal people were born with 8 fingers. Different base systems, like octal (base 8), binary (base 2) or hexadecimal (base 16) can be used to abbreviate patterns that adhere to the base.
- Each digit in an octal number represents three ($8 = 2\hat{}3$) binary digits (useful to specify anything that comes in groups of three). Counting in octal is done with the numbers 0 through 7.
- Pixels e.g. are composed of 3 color components: 8 bits of red, green, blue each. A medium blue in binary would be a 24-digit number, but it can be condensed to a 6-digit hexadecimal, 436FCD.
- Table shows the file modes in binary and in octal notation.
- Most languages have conversion functions for different bases, e.g. `oct` or `format` in Python to convert to octal:

```
format(8,'o')  # decimal 8 to octal (10)
format(8,'b')  # decimal 8 to binary (1000)
oct(10)  # octal to decimal
```

OCTAL	BINARY	FILE MODE
0	000	—
1	001	–x
2	010	-w-
3	011	-wx
4	100	r–
5	101	r-x
6	110	rw-
7	111	rwx

- By setting 3 octal digits, we can set the file mode for the owner, group owner, and world.
- Example: run the block **??**. An empty file is created and long-listed.

```
rm -rf foo.txt  # we may already have a file like this
> foo.txt
ls -l foo.txt

-rw-rw-r-- 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

- In the block **??** below, change the permissions (file mode) to 600 with the command `chmod 600 [filename]` and list the file.

  Check with the table that this is what was supposed to happen: read and write permissions for the owner, and no access rights for anyone else.

```
chmod 600 foo.txt # owner: rw- or 110, all others: --- or 000
ls -l foo.txt

-rw------- 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

- Now change the mode of foo.txt to be readable by owner, group, and world, with no other permissions for any of these.

```
chmod 444 foo.txt
ls -l foo.txt

-r--r--r-- 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

- Change the permissions for `foo.txt` back to default (`rw-rw-r--`):

```
chmod 664 foo.txt
ls -l foo.txt

-rw-rw-r-- 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

- What does `chmod 775` do? Why is this a common setting?

```
chmod 775 foo.txt
ls -l foo.txt

-rwxrwxr-x 1 marcus marcus 0 Apr  1 22:14 foo.txt
```

Answer: this is the default setting for your directories (check)

```
ls -ld */  # long-list directories only

drwxrwxr-x 2 marcus marcus 4096 Mar 28 16:53 assignments/
```

Changing file modes with symbols

- Symbolic notation is divided into three parts:
  - Who the change will affect
  - Which operation will be performed
  - What permission will be set
- To specify who is affected, a combination of characters is used, as shown in table .

WHO	MEANING
u	user = file or directory owner
g	group owner
o	others = world
a	all = combination of u,g,o

- If no character is specified, "all" (a) is assumed. Three operations are allowed, see table :

OPERATION	MEANING
+	permission to be added
-	permission to be removed
=	specified permissions to be applied and all others removed

NOTATION	MEANING
u+x	add execute permission for owner
u-x	remove execute permission for owner
+x	add execute permission for owner, group, world
a+x	add execute permission for owner, group, world
o-rw	Remove read, write permissions from anyone except owner, group
go=rw	Set group owner and anyone else to have read, write permissions. Remove existing group owner/world execute permissions
u+x, go=rx	Add execute permissions for owner, set read, execute for group/others

- Table shows some examples. Multiple specifications may be separated by commas.
- Example: in the block **??**, create an empty file `bar.txt` and long-list it:

```
rm -rf bar.txt
> bar.txt
ls -l bar.txt

-rw-rw-r-- 1 marcus marcus 0 Apr  1 22:15 bar.txt
```

- In the block **??** below, set the permissions for the owner, the group and others to read and write only, for `bar.txt`. Use the command `chmod [operation] [filename]`, then list the file.

```
chmod a=rw bar.txt
ls -l bar.txt

-rw-rw-rw- 1 marcus marcus 0 Apr  1 22:15 bar.txt
```

- Change the mode of bar.txt to be readable by owner and group only, with no other permissions for any of these.

```
chmod ug=r,o-rw bar.txt
ls -l bar.txt

-r--r----- 1 marcus marcus 0 Apr  1 22:15 bar.txt
```

Setting permissions in the GUI

You can inspect and set permissions also in GUIs. It usually takes two clicks (except for hidden files, if they're not set to be viewed), and you need administrative rights (which may require an admin login).

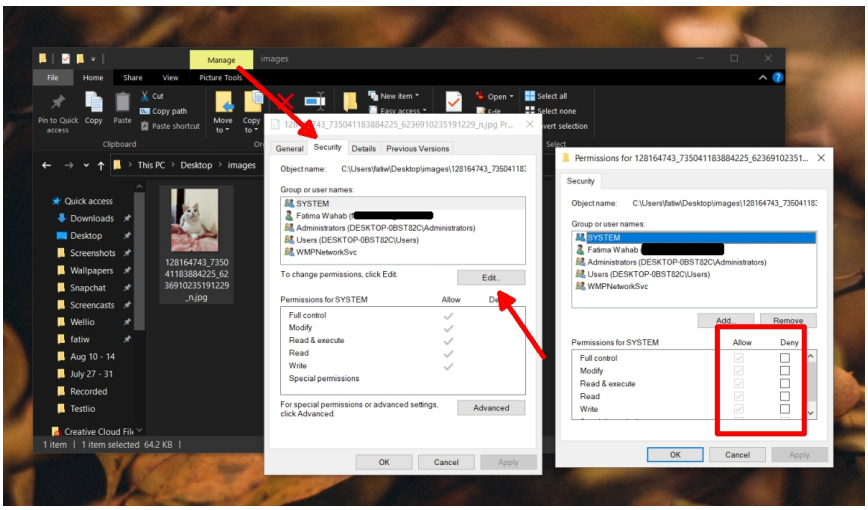


Figure 1: Windows File Explorer

Setting default permissions (`umask`)

- When a file is created, the `umask` command expresses a *mask* of bits to be **removed** from from the mode attributes of a file.
- Running the command without arguments returns the default mask:

```
# default permission mask
umask

0002
```

Figure 2: MacOS Finder

- Review the octal encoding in to see what the '2' means:

Octal	Binary	File
0	000	—
2	010	-w-

- The first bit of the mask is the `setuid` bit (to be covered later).
- Create an empty file `foo.txt` to see default permissions (`0002`):

```
if [ -e "foo.txt" ]; then
    rm -rvf foo.txt
fi
> foo.txt
ls -l foo.txt

removed 'foo.txt'
-rw-rw-r-- 1 marcus marcus 0 Apr  1 22:15 foo.txt
```

- Reset the mask to `0000` ('remove nothing') and create the file again:

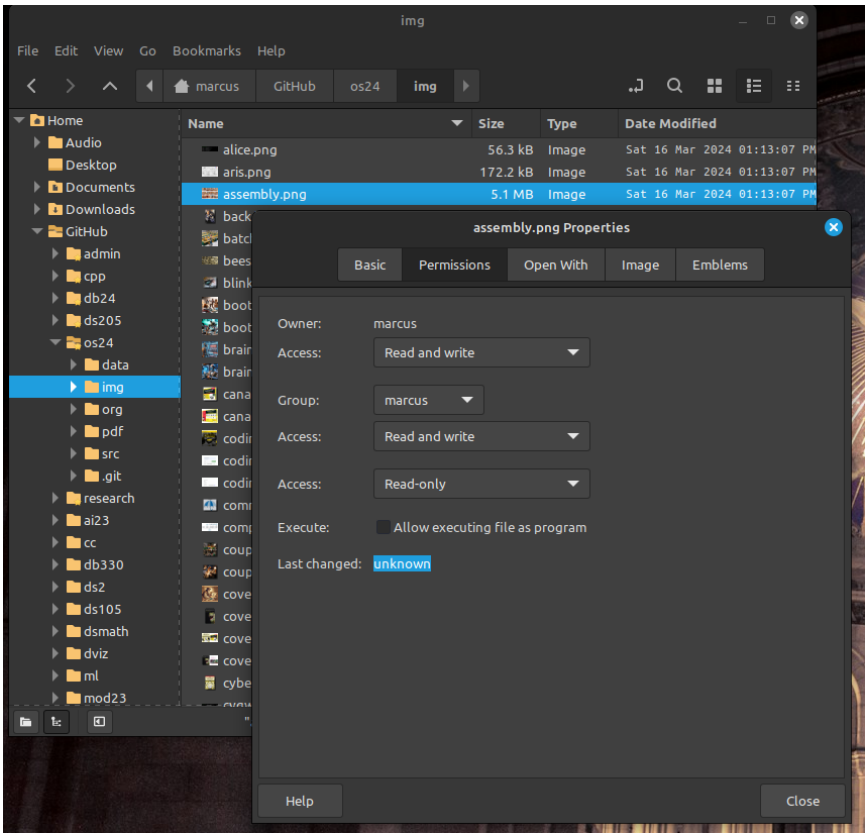


Figure 3: Linux

```
rm foo.txt
umask 0000
> foo.txt
ls -l foo.txt

-rw-rw-rw- 1 marcus marcus 0 Apr  1 22:15 foo.txt
```

- Reset the mask to `0022` and create the file again:

```
rm foo.txt
umask 0022
> foo.txt
ls -l foo.txt

-rw-r--r-- 1 marcus marcus 0 Apr  1 22:15 foo.txt
```

- Expand the mask `0002` to binary and compare it to the attributes: `0002` means 'remove `-w-` from the 'others' permissions:

Original	— rw- rw- rw-
Mask	000 000 000 010
Result	— rw- rw- r–

- Expand the mask `0022` to binary and compare it to the attributes: `0022` means 'remove `-w-` permissions from 'group' and 'others':

Original	— rw- rw- rw-
Mask	000 000 010 010
Result	— rw- r– r–

- Where a 1 appears in the binary value, the corresponding attribute is unset.
- Exercises:
  1. What are the permissions of a new file `foo.txt` when you run `umask 0226`?

```
if [ -e "foo.txt" ]; then
    rm -rvf foo.txt
fi
umask 0226
> foo.txt
ls -l foo.txt

removed 'foo.txt'
-r--r----- 1 marcus marcus 0 Apr  1 22:15 foo.txt
```

2. What is the corresponding binary code for `umask 0226`?

   ```
   0223 = 000 010 010 110
   ```

3. Which permissions are masked (removed) by `umask 0224`?

   ```
   0223 = 000 010 010 110
        = --- -w- -w- rw-  (removed)
        = --- r-- r-- ---  (remaining)
   ```

4. What about `umask 0331` - what does that do?

   ```
   if [ -e "foo.txt" ]; then
       rm -rvf foo.txt
   fi
   umask 0321 # remove --- 011 010 001 or --- -wx -w- --x
   > foo.txt
   ls -l foo.txt

   removed 'foo.txt'
   -r--r--rw- 1 marcus marcus 0 Apr  1 22:15 foo.txt
   ```

5. Which masks would remove all permissions? Show this.

   ```
   if [ -e "foo.txt" ]; then
       rm -rvf foo.txt
   fi
   umask 0666    # for executable files, use 0777
   > foo.txt
   ls -l foo.txt

   removed 'foo.txt'
   ---------- 1 marcus marcus 0 Apr  1 22:15 foo.txt
   ```

- `umask` is useful in practice for enforcing security policies, controlling default permissions, and ensuring that newly created files and directories have the desired level of access restrictions.

Special permissions (`setuid`, `setgit`, sticky bit)

- The `setuid` bit (octal `4000`) changes the user ID from the current user ID running the program to that of the program's owner.
- When an ordinary user runs a program that is `setuid root`, it runs with superuser privileges and can access all files on the computer.
- Listing with `ls -l` shows the special permissions. Here is an example of assigning `setuid` to a program:

```
if [ -e "empty" ]; then
    rm -rvf empty
fi
> empty
chmod -v u+s empty
ls -l empty

mode of 'empty' changed from 0664 (rw-rw-r--) to 4664 (rwSrw-r--)
-rwSrw-r-- 1 marcus marcus 0 Apr  1 22:15 empty
```

- The `setgid` bit (octal `2000`) changes the group ID from the current group ID running the program to that of the file owner.
- If `setgid` is set on a directory, new files will be given the directory's group ownership rather than the file creator's `gid`.
- Now, members of a common group can access all files in that directory, independent of the file owner's group.
- Here is an example of assigning `setgid` to a directory:

```
if [ -d "Empty" ]; then
    rm -rvf Empty
fi
mkdir -v Empty
chmod g+s Empty
ls -ld Empty

mkdir: created directory 'Empty'
drwxrwsr-x 2 marcus marcus 4096 Apr  1 22:15 Empty
```

- The **sticky bit** (octal `1000`) is a Unix artifact that stopped an executable file from being swapped out of cache memory but this is longer required.
- On Linux, if the sticky bit is set on a directory, it prevents users from deleting or renaming files in that directory unless they are the owner of the file, owner of the directory, or the superuser.
- This is used to control access to a shared directory such as `/tmp`.
- Here is an example of a directory with the sticky bit set:

```
if [ -d "Empty" ]; then
    rm -rvf Empty
fi
mkdir -v Empty
chmod +t Empty
ls -ld Empty

removed directory 'Empty'
mkdir: created directory 'Empty'
drwxrwxr-t 2 marcus marcus 4096 Apr  1 22:15 Empty
```

Changing identities (`su`, `sudo`)

- There are three ways to change your user identity:
  1. By logging out and back in as an alternate user.
  2. By using the `su` ('superuser') command.
  3. By using the `sudo` ('superuser do') command.
- With `su`, you can run a shell with an other user and group IDs, or a single command. You can only try this on a fully functional terminal:

```
marcus@marcus@vostro:~ $ su -
Password:
root@marcus-Vostro-3470:~# exit
logout
marcus@marcus@vostro:~ $
```

- The `-` is an abbreviation of the `-l` option of the `su` command, for *login*. If the user is not specified, the superuser is assumed.
- The `-c` flag prepares `su` for accepting a single command. The login shell is entered, the command is executed and left again:

```
marcus@marcus@vostro:~ $ su -c 'ls -l /root'
Password:
total 0
marcus@marcus@vostro:~ $
```

- We cannot usually look at `/root` - check this (`stderr` to `stdout`):

```
ls -l /root 2>&1

ls: cannot open directory '/root': Permission denied
```

- Because the `root` user does not normally (for security resons) have a default password, use of `sudo` is encouraged:
  1. `sudo` can be configured for ordinary users in a controlled way. The `sudoers(5)` man page contains more information.
  2. `sudo` does not require access to the superuser's password. You know this from using `sudo apt update -y` and `sudo apt upgrade -y`.
  3. Authenticating `sudo` use on scripts requires the user's own password. No new shell is started, no new environment is loaded.
- You can see the privileges granted by `sudo` with the `-l` option:

```
sudo -l

Matching Defaults entries for marcus on marcus-Vostro-3470:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbi

User marcus may run the following commands on marcus-Vostro-3470:
    (ALL : ALL) ALL
    (root) NOPASSWD: /usr/bin/mintdrivers-remove-live-media
    (root) NOPASSWD: /usr/bin/mint-refresh-cache
    (root) NOPASSWD: /usr/lib/linuxmint/mintUpdate/synaptic-workaround.py
    (root) NOPASSWD: /usr/lib/linuxmint/mintUpdate/dpkg_lock_check.sh
```

- In the Windoze world, administrative privileges are bestowed on the user without sharing much information. Programs executed by such a user have the potential to damage the system (*malware*).
- In the Unix world, regular users and administrators have traditionally been further apart. Like in database systems, privileges are only granted to users when really needed.
- Operating as `root` all the time makes everything more convenient but reduces the security of a Linux system to that of a Windoze system.
- Ubuntu and its distributions (like Linux Mint) do not give a default password to `root` but use `sudo` to grant superuser privileges.

Change file owner and group (`chown`)

- You need superuser privilegs to change owner and group of a file.
- The syntax of `chown` is:

  ```
  chown [owner] [:[group]] file...
  ```

- Here are some examples for arguments:

Argument	Results
`bob`	Changes ownership to `bob`
`bob:users`	New owner `bob`, new group `users`
`:admins`	New group owner is `admin`
`bob:`	New owner `bob` new group is `bob`'s

- To try this, add a new user called 'tiger'. This requires a fully functional terminal. You can use `M-x term` in Emacs for that.
- This CLI dialog below shows:
  1. creating a new user with `sudo adduser [username]`
  2. checking new user with `cat /etc/passwd | grep [username]`
  3. checking new user with `sudo` in `/etc/shadow`
  4. checking new user's home directory with `ls -l /home`
  5. logging in as new user with `su - [username]`

6. running `pwd` and `whoami` and logging out with `exit`

- Terminal dialog:

```
marcus@marcus@vostro:~/GitHub/os24/org $ sudo adduser testuser
[sudo] password for marcus:
Adding user `testuser' ...
Adding new group `testuser' (1001) ...
Adding new user `testuser' (1001) with group `testuser' ...
Creating home directory `/home/testuser' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for testuser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
marcus@marcus@vostro:~/GitHub/os24/org $ cat /etc/passwd | grep testuser
testuser:x:1001:1001:,,,:/home/testuser:/bin/bash
marcus@marcus@vostro:~/GitHub/os24/org $ cat /etc/shadow | grep testuser
cat: /etc/shadow: Permission denied
marcus@marcus@vostro:~/GitHub/os24/org $ sudo cat /etc/shadow | grep testuser
testuser:$y$j9T$bKWS0c8iZpwvm6dwUnij5.$iCjWP2flDCCs1WZUNn9GuX.1lPChaWtSuox0X8cTc37:19
812:0:99999:7:::
marcus@marcus@vostro:~/GitHub/os24/org $ ls -l /home
total 8
drwxr-x--- 29 marcus   marcus   4096 Mar 29 12:32 marcus
drwxr-x---  3 testuser testuser 4096 Mar 30 10:21 testuser
marcus@marcus@vostro:~/GitHub/os24/org $ su - testuser
Password:
testuser@marcus-Vostro-3470:~$ pwd
/home/testuser
testuser@marcus-Vostro-3470:~$ whoami
testuser
testuser@marcus-Vostro-3470:~$ exit
logout
marcus@marcus@vostro:~/GitHub/os24/org $
```

- Perform all these actions now for a new user `testuser` using the fully functional regular Linux terminal (not Emacs) as password, use "test-password".

```
sudo adduser testuser
cat /etc/passwd | grep testuser
cat /etc/shadow | grep testuser
sudo cat /etc/shadow | grep testuser
ls -l /home
su - testuser
```

- Back home in your original `LyonXX` account, create a file:

```
echo "I am superuser" > superuser.txt
ls -l superuser.txt
cat superuser.txt

-rw-rw-r-- 1 marcus marcus 15 Apr  1 22:15 superuser.txt
I am superuser
```

- You now have two users, `LyonXX` and `testuser`, and your `LyonXX` user has access to superuser privileges. Do this in the terminal:

  1. As superuser, copy your file `superuser.txt` to the home directory of `testuser`, which is in `/home/testuser`:

     ```
     sudo cp superuser.txt ~testuser
     sudo ls -l ~/testuser
     ```

  2. You see that the file is owned by `root` and is of group `root`, too. This means that `testuser` cannot edit it. But you can use `chown` to bestow these privileges:

     ```
     sudo chown testuser: ~testuser/superuser.txt
     sudo ls -l ~/testuser
     ```

  3. The argument `testuser:` has changed both ownership and group.

- On most Linux systems, once you enter your password, a timer gives you 5 minutes without having to re-enter it (see `sudo(8)` timeout).

- Older versions of Linux have a more restricted program, `chgrp`, to only change the group.

Summary

- Unix multiuser capability is fundamental, allowing user data protection from others.

- Essential commands related to user and group management include `id`, `chmod`, `umask`, `su`, `sudo`, `chgrp`, `passwd`.

- The Unix security model encompasses file ownership, group memberships, and access rights, delineating control over resources.

- User IDs (`uid`) and group IDs (`gid`) start at specific numbers varying by distribution, impacting system resource access and management.

- Access rights are categorized into read, write, and execute, with file permissions displayed using `ls -l`.
- `chmod` is used to modify file permissions employing octal notation and symbolic modes for precise access control.
- The `umask` command sets the default permissions of newly created files, and has a critical role in system security.
- Special permissions (`setuid`, `setgid`, sticky bit), provide importance fine-tuning access control and execution rights.
- To change user identities use the `su` and `sudo` commands, with `sudo` particularly emphasized for its security advantages in executing superuser-level commands.
- The chown command alters file and directory ownership using `sudo` superuser privileges. operations.

Noweb chunks

- Remove `foo.txt` if file exists:

```
if [ -e "foo.txt" ]; then
    rm -rvf foo.txt
fi
```

- Remove all files `foo*.txt` if they exist:

```
for file in foo*.txt; do
    if [ -e "$file" ]; then
        rm -rfv $file
    fi
done
```

- Remove `empty` if file exists:

```
if [ -e "empty" ]; then
    rm -rvf empty
fi
```

- Remove `Empty` if directory exists:

```
if [ -d "Empty" ]; then
    rm -rvf Empty
fi
```

how can I check if a directory ‘Empty‘ exists?

[AI]: You can check if a directory named ‘Empty‘ exists in the current directory by using the following command in Emacs:

“‘elisp (file-exists-p "Empty") “‘

This function returns ‘t‘ if the file or directory exists, and ‘nil‘ otherwise.

[ME]: in a bash script

[AI]: To check if a directory named ‘Empty‘ exists in a bash script, you can use the following command:

“‘bash if [-d "Empty"]; then echo "Directory ’Empty’ exists." else echo "Directory ’Empty’ does not exist." fi “‘

This bash script will check if the directory ‘Empty‘ exists in the current directory. If the directory exists, it will print "Directory ’Empty’ exists." Otherwise, it will print "Directory ’Empty’ does not exist."

[ME]: