UNIX File Permissions (1)

/bin

- bin contains essential executables required by the system e.g. ls cat mv etc.

/usr

- this directory is used for user-installed things e.g. libraries
- things that are installed later

/etc

- contains system-wide configuration files

The `/etc` directory is an important part of the file system hierarchy in Unix and Unix-like operating systems (such as Linux). This directory contains configuration files for the system and various applications. Here's a bit more detail:

- **System Configuration**: The `/etc` directory holds global configuration files that affect the system's behavior for all users. These files can include settings for system time, system-wide environment variables, network configuration (like `hosts`, `resolv.conf` for DNS resolution), and more.
- **Application Configuration**: Many system-level applications also store their configuration files in `/etc`. This can include web servers (e.g., Apache, Nginx), database servers (e.g., MySQL, PostgreSQL), and many daemon services. Configuration files in `/etc` are usually text files that can be edited by a system administrator.

On a modern UNIX descended system (LINUX, Mac OS Windows etc)., there is an all powerful user: Root

- root is the *super user*
- root is the system administrator
- ID value of: 0 (UID)

> And so root beget init which beget login...
> And so other users came to pass
> ► Each less powerful than the original root

The Password File

So how do you specify each of the users on a modern UNIX system?

- in the depths of the computers configuration directory (*/etc/*) you find something called the **password file**
- inside the password file you will find a list of all the **users of the system**
  - *essentially a table of all the system users:*

```
$ grep -Ev '^_' /etc/passwd | column -ts :
```

Username	Password	UID	GID	GECOS	Home Directory	Shell
root	*	0	0	Charlie &	/root	/bin/ksh
daemon	*	1	1	The devil himself	/root	/sbin/nologin
operator	*	2	5	System &	/operator	/sbin/nologin
bin	*	3	7	Binaries Commands and Source	/	/sbin/nologin
build	*	21	21	base and xenocara build	/var/empty	/bin/ksh
sshd	*	27	27	sshd privsep	/var/empty	/sbin/nologin
www	*	67	67	HTTP Server	/var/www	/sbin/nologin
nobody	*	32767	32767	Unprivileged user	/nonexistent	/sbin/nologin
joseph	*	1000	1000	Joseph Hallett,,,	/home/joseph	/usr/local/bin/bash

See `man 5 passwd` or your OS's manual pages.

  - each user (UID) can only refer to ONE user
  - a group (GID) can have multiple users

- the password symbol is nearly always **star (*)**

So when you add an extra user it is basically telling the operating system to manipulate the password file to dd another user to the table

The Group File

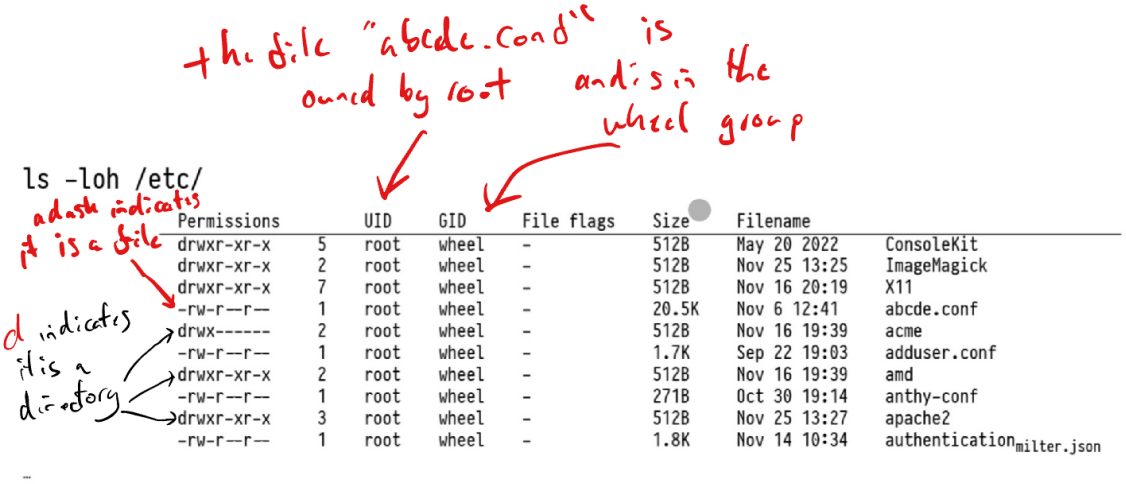
so what is inside the **group file?**

- a big list of users of the system and their id
- similar to users, except a group can have multiple users
- no passwords every listed (they're in: /etc/shadow)

```
$ grep -Ev '^_' /etc/group | column -ts :
```

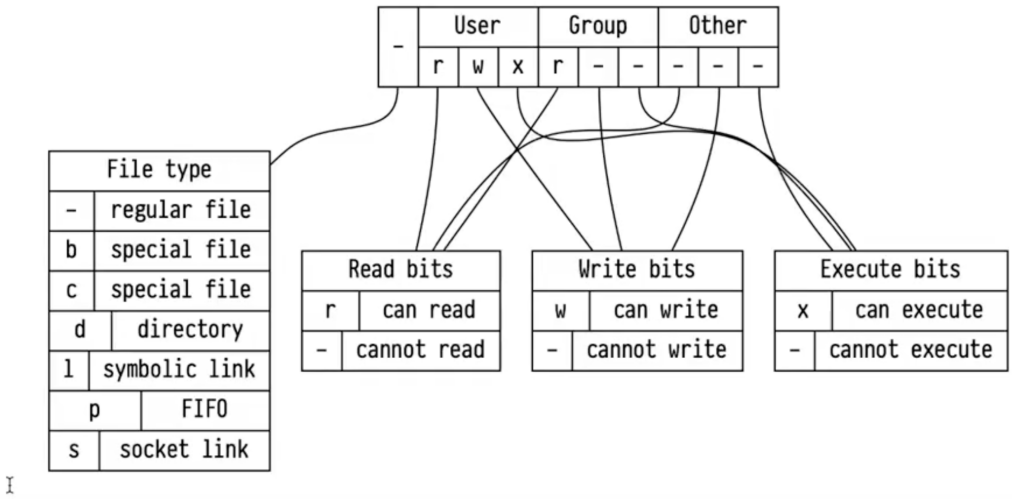
Groupname	Password	GID	Members
wheel	*	0	root,joseph
daemon	*	1	daemon
kmem	*	2	root
sys	*	3	root
tty	*	4	root
operator	*	5	root
bin	*	7	
wsrc	*	9	joseph
users	*	10	
auth	*	11	
games	*	13	
staff	*	20	root,joseph
wobj	*	21	joseph
sshd	*	27	
guest	*	31	root
utmp	*	45	
crontab	*	66	
www	*	67	
network	*	69	
authpf	*	72	
dialer	*	117	
nogroup	*	32766	
nobody	*	32767	
joseph	*	1000	

All files are however owned by a user and a group...



UNIX Discretionary Access Controls

And the owner of each file could set the *permissions* for each file



- first thing tells you what sort of file it is, e.g. “-” or “b”
  - b denotes a **block file** - e.g. a disc drive

- c denotes a **character file** - you can treat it as a series of bytes that can be turned into characters
- d is for **directories**
- l - **symbolic link** - this isnt a specific file but a *reference* to another file somewhere else
- p = **FIFO**
- **s = socket link** a network connection you can read and write files to

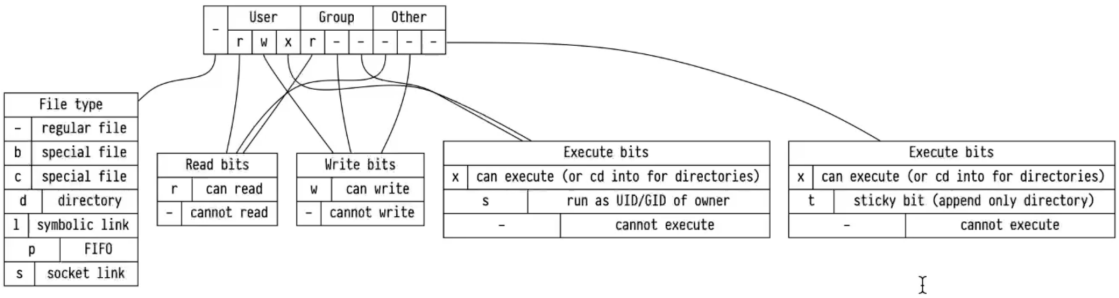
- using these bits and letters you can create this discretionary access control system

in this file for example:

```
-rw-r--r--     1    root    wheel    -            20.5K    Nov 6 12:41     abcde.conf
```

- the user is allowed to write it and read it
- people in the group and everyone else can only read it however

Extra bits



while the three main bits are: **read, write, execute**

set UID bit = when this program runs, **you get the privileges the the user or the group** has for **that file**

- this is used for implementing password changes for example
  - say you want to change your password, but you normally wouldnt have the permissions to view and alter the password file to see other userspasswords or change them
    - the **root** user is temporarily giving you privilege access to the pass word file
      - by giving you the permissions to execute the set UID password program

Weird extra bits...

- the **sticky bit t** mostly for log directories and temporary directories
- You should be able to append log files, but not delete them

set uid / setgid - are used for privilege seperation

for example how do you update your password?

- passwords are usually stored **securely** in the shadow file: /etc/shadow (or equivalent)
  - no readable by most users

Changing Passwords

- the password program changes your password:

```
ls -a $(command -v passwd)

-r-sr-xr-x 1 root bin 21208 Jan 12 03:08 /usr/bin/passwd
```

- you temproarily take on the permissions of the root user to run it

Other helpful setuid programs:

su - temporarily switch to a user if you know their password (by default root)

sudo - switch to user if the *system admin* says you're allowed to with your password

- i.e. system admin trusts this user to do these specific tasks *as* the root user

doas - modern rewrite of sudo with less bugs (and Spiderman references)

setuid programs are dangerous and you need to be careful when using them

- programs that let you write to files as root, if you can exploit this program then you could easily get permissions of the root user == bad

---

System Admin

How do you change who owns a file?

```
ls -l exam

-rw-r--r-- 1 joseph joseph 0 Jan 12 11:49 exam
```

- e.g. the file exam
- how do you change the permissions the above command shows us:

```
chown joseph:staff exam


OR ALTERNATIVELY
```

```
chown :staff exam
```

- this then sets members of the staff group as owners of the exam file

```
ls -l exam

-rw-r--r-- 1 joseph staff 0 Jan 12 11:49 exam
(See man 1 chown)
```

- the file is still readable by everyone however, which you dont want:

```
chmod go-wx exam
```

- this takes away from group and other (go) the writable and executable (wx) permissions
    - alternatively +go would give those permissions to group and other

Octal for Permissions

Footnote

Some people like to use octal (base 8) to express permissions, where r=4, w=2, x=1...
Instead of saying `go-wx` to remove `w` and `x` bits from the group and other permissions they'll say:
`chmod 744 exam`
I suggest you give these people a wide berth.

- (but you should know how to do it)

```
ls -l exam //the file is exam

-rw-r--r-- 1 tom tom
```

```
chown tom:staff exam // this lets the user tom and the group sta
```

How do you change a files permissions?

```
chmod go-wx exam // this takes away the writeable and executable

//chmod go+wx would add these permissions (- takes, + adds)

ls -l exam
```

Recap

- systems have users
- UNIX DAC (discretionary access control) lets you set file permissions
- setuid and setgit programs set the user and group ids
- **chmod** to change permissions
- **chown** to change file ownership

One more thing:

- traditionally the root user can do anything
    - in most modern operating systems however this has been sploit up a bit more
- e..g LINUX uses **capabilities** to set what any user can do
- **namespaces** allow for multiple root users with different capabilities

- e.g. not one ALL powerful root user but numerous ones with power in different sectors

We wont need to know this stuff unless we use docker...