User accounts and Groups

Error: Permission denied!

Users and groups

- Linux/Unix were created as multi-user OS
  - Multiple users can login and work on the same machine at the same time
  - MAINFRAMES!
- User/Group permissions are a method of protecting users from each other
- Permissions are applied to files and stored in the filesystem
- Groups are a way to organize users
  - Users belong to a primary group
  - Users can also be member of many “secondary” groups at the same time
  - Use groups to bulk manage permissions and access to system resources

User accounts

- The “`adduser`”/“`deluser`” command is the recommended way to create and remove user accounts **in Ubuntu**
- Read: https://ubuntu.com/server/docs/security-users
  - The “adduser” will create new user, and add it to a primary group with the same name as the username
  - The “deluser” will NOT delete the home directory of the removed user by default (use “--help” option to see what options are available)
- The “`passwd`” and “`chage`” commands:
  - change password for a user
  - Lock (-l) and unlock (-u) the “password” to deny/allow password login
  - Set password expiry

Groups

- To add or remove groups: addgroup/delgroup commands
  - To add user group: `addgroup  <user_group>`
    - Can also use: `adduser --group <user_group>`
  - To remove a group: `delgroup <group>`
- The adduser/deluser commands are used to add/remove a user to/from a group:
  - To add a user to a group: `adduser <username> <group>`
  - To remove a user from a group: `deluser <username> <group>`
- Use “groups” command to find out what groups do the user belong to
  - `sudo groups <username>`

System users and groups

- You can create normal users/groups and “system” users/groups
  - For example:
    - `addgroup --system <system_group>`
    - `adduser --system <system_user>`
- There is no inherent difference between the two types in the kernel, however:
  - System groups are special purpose groups used for system operation like backup, maintenance or for granting access to hardware
  - Processes may have a “user” and “group” to create and access files
  - System users/groups (lower UID) and regular users/groups (higher UID) are treated differently only by “convention”
    - For example, it is used to not display system users in a graphical login manager

Important files/directories

- `/etc/passwd`: user account names, UID, GID, info, home directory, and default shell
- `/etc/shadow`: All encrypted passwords are stored in this file
- `/etc/group`: All groups definitions are stored in this file
- adduser configuration:
  - Home folder template for new users: `/etc/skel`
  - The adduser command settings: `/etc/adduser.conf`
    - Example setting: DIRMODE=0755 # can change to 0750 for example
- Password policy and settings:
  - `/etc/pam.conf`
  - `/etc/pam.d/` ← if this directory exists system will ignore `/etc/pam.conf`
    - Password policy: `/etc/pam.d/common-password`

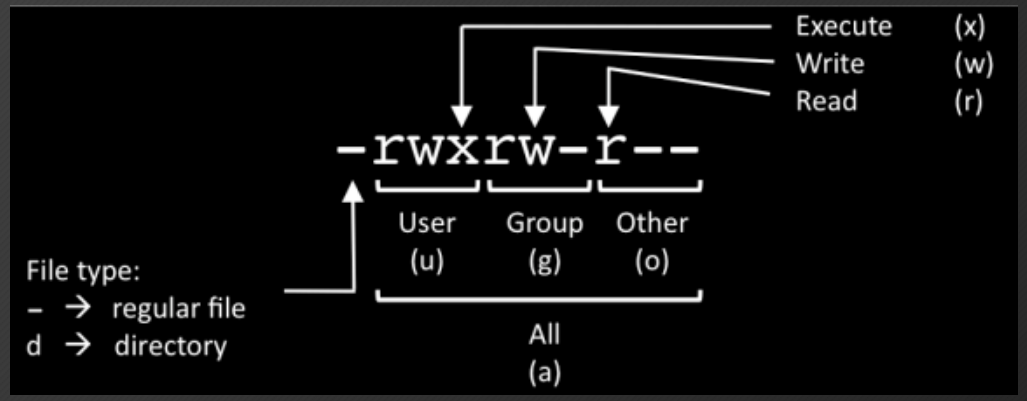
Demo

- Use “cat /etc/passwd” and see if you can identify the fields!
- In what order are the users listed?
- Explore /etc/shadow file:
  - What does ! And * mean?
- See here for more info:
  - man 5 passwd (what is the difference with man passwd?)
  - man shadow
  - Also see:
    - https://www.cyberciti.biz/faq/understanding-etcshadow-file/
    - https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/

File Permissions and Notations

Permissions

- Generic permissions for files and directories:
  - Read (r), write (w), execute (x)
- Commands:
  - `chown`
  - `chmod`
  - `chgrp`
- The user issuing the commands need to have write permission to modify the file permissions, before and after the modification!
  - You may need root access



Permissions, numeric vs. symbolic

- The chmod command is used to change permissions
  - `chmod [options] mode[,mode] file1 [file2 ...]`
- Examples of chmod command:
  - `$: chmod 777 filename`
  - `$: chmod u+x filename`
  - `$: chmod o-w filename`
  - `$: chmod g+rwx filename`

- Mmm… these are weird:
  - `#: chmod 2755 directoryname`
  - `#: chmod u+s filename`

Octal	Decimal	Permission	Representation
000	0	No permission	---
001	1	Execute	--x
010	2	Write	-w-
011	3	Wire + execute	-wx
100	4	Read	r--
101	5	Read + execute	r-x
110	6	Read + write	rw-
111	7	Read + write + execute	rwx

Additional permissions

- What if you have an application that you want to always run as a specific user/group regardless of any user executing it?
- What if you have a shared directory that you want all files and directories to be owned by the group, and not specific users' group who created them?

- ANSWER: set-user-id and set-group-id bits!

Setuid and setguid bits

Permission	Numeric	Relative	If on files	If on directory
SUID	4	u+s	Users execute file with permissions of the file owner	No meaning
SGID	2	g+s	Users execute file with permissions of the group owner	File created in directory gets the same group owner
Sticky bit	1	+t	No meaning	Users are prevented from deleting other user's files

- `#: chmod 2750 directoryname`
- `#: chmod u+s filename`
- For detailed explanation and typical usage scenarios see: https://linuxconfig.org/how-to-use-special-permissions-the-setuid-setgid-and-sticky-bits

Security and Monitoring

- Who is/was logged in and when?
  - Two choices: dig into the log files directly, OR use these commands:
    - `id`
    - `who`
    - `w`
    - `last`

Linus/Unix User Account Best Practices

- Always use strong passwords
  - Enforce password lifecycles: PAM
- More secure ssh access: SSH passphrase + public key instead of username + password
- Fail2ban.org
- Extended File Attributes in Linux
  - Set ACL (Access Control List) for files!
  - Supported in Linux, but not all Unix distros
  - Use xattr command
  - See: https://en.wikipedia.org/wiki/Extended_file_attributes
  - And: https://www.techrepublic.com/blog/linux-and-open-source/learn-to-use-extended-file-attributes-in-linux-to-boost-security/