

## **Creating Users and Groups/ Ownership and Permissions**

*Mocktar ISSA*  
*Full-Stack Software Engineer*



# Introduction

During the installation process, most installers create a normal user and either give this user the permission to execute administrative commands with `sudo` or require the root user account password be configured as part of the installation process. Most Linux systems are configured to allow for one unprivileged (non-root) user to log in, as well as have the ability to effectively execute commands as the root user, either directly or indirectly.

If the computer is to be used by only one person, then having only one regular user account might be sufficient. However, if a computer needs to be shared by multiple people, then it is desirable to have a separate account for each person who uses it. There are several advantages to individuals having their own separate accounts:

Accounts can be used to grant selective access to files or services. For example, the user of each account has a separate home directory that is generally not accessible to the other users.

The `sudo` command can be configured to grant the ability to execute select administrative commands. If users are required to use the `sudo` command to perform administrative commands, then the system logs when users perform these commands.

Each account can have group memberships and rights associated with it allowing for greater management flexibility.

On some distributions, creating a new user account also automatically creates a group account for the user, called a User Private Group (UPG). On these systems, the group and username would be the same, and the only member of this new group would be the new user.

Before you begin creating users, you should plan how to use groups. Users can be created with memberships in groups that already exist, or existing users can be modified to have memberships in existing groups.

If you already have planned which users and groups you want, it is more efficient to create your groups first and create your users with their group memberships. Otherwise, if you create your users first, and then your groups, you'll need to take an extra step to modify your users to make them members of your groups.

# Groups

The most common reason to create a group is to provide a way for users to share files. For example, if several people who work together on the same project and need to be able to collaborate on documents stored in files for the project. In this scenario, the administrator can make these people members of a common group, change the directory ownership to the new group and set permissions on the directory that allows members of the group to access the files.

After creating or modifying a group, you can verify the changes by viewing the group configuration information in the `/etc/group` file with the `grep` command. If working with network-based authentication services, then the `getent` command can show you both local and network-based groups.

For local usage, these commands show the same result, in this case for the root group:

```
adavtyan@artur-lpt:~$ sudo grep root /etc/group
root:x:0:
adavtyan@artur-lpt:~$ getent group root
root:x:0:
```

The `groupadd` command can be executed by the root user to create a new group. The command requires only the name of the group to be created. The `-g` option can be used to specify a group id for the new group:

```
adavtyan@artur-lpt:~$
adavtyan@artur-lpt:~$ sudo groupadd -g 1100 researcher
adavtyan@artur-lpt:~$ sudo grep researcher /etc/group
researcher:x:1100:
```

If the `-g` option is not provided, the `groupadd` command will automatically provide a GID for the new group. To accomplish this, the `groupadd` command looks at the `/etc/group` file and uses a number that is one value higher than the current highest GID number.

## Modifying a Group

The **groupmod** command can be used to either change the name of a group with the **-n** option or change the GID for the group with the **-g** option. Changing the name of the group may confuse users who were familiar with the old name and haven't been informed of the new name. However, changing the group name won't cause any problems with accessing files, since the files are owned by GIDs, not group names.

For example:

```
adavtyan@artur-lpt:~$ ls -ls index.html
-rw-rw-r-- 1 adavtyan research 0  Uuy  6 18:13 index.html
adavtyan@artur-lpt:~$ sudo groupmod -n researcher research
adavtyan@artur-lpt:~$ ls -ls index.html
-rw-rw-r-- 1 adavtyan researcher 0  Uuy  6 18:13 index.html
```

After the previous **groupmod** command, the `index.html` file has a different group owner name. On the other hand, if you change the GID for a group, then all files that were associated with that group will no longer be associated with that group. In fact, all files that were associated with that group will no longer be associated with any group name. Instead, these files will be owned by a GID only, as shown below:

```
adavtyan@artur-lpt:~$ sudo groupmod -g 1101 researcher
adavtyan@artur-lpt:~$ ls -ls index.html
-rw-rw-r-- 1 adavtyan 1100 0  Uuy  6 18:13 index.html
```

If you decide to delete a group with the **groupdel** command, be aware that any files that are owned by that group will become orphaned. Only supplemental groups can be deleted, so if any group that is the primary group for any user, it cannot be deleted. The administrator can modify which group is a user's primary group, so a group that was being used as a primary group can be made into a supplemental group and then can be deleted.

## ***Users***

*User account information is stored in the `/etc/passwd` file and user authentication information (password data) is stored in the `/etc/shadow` file. Creating a new user can be accomplished by manually adding a new line to each of these files, but that is generally not the recommended technique.*

*By using an appropriate command to add a new user, these files can be modified automatically and safely. If you were to modify these files manually, you would risk making a mistake that could prevent all users from being able to log in normally.*

*Before you begin creating users for your system, you should verify or establish practical values that are used by default with the `useradd` command. These settings can be found in the configuration files that are used by the `useradd` command.*

*Ensuring that the values in these configuration files are reasonable before adding users can help save you the time and trouble of having to correct user account settings after adding the users.*

# User Configuration File

The **-D** option to the **useradd** command allows you to view or change some of the default values used by the **useradd** command. The values shown by **useradd -D** can also be viewed or updated by manipulating the **/etc/default/useradd** file:

```
adavtyan@artur-lpt:~$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
adavtyan@artur-lpt:~$
```

*The following describes each of these values:*

## **Group: GROUP=100**

In distributions not using UPG, this is the default primary group for a new user, if one is not specified with the **useradd** command. This is usually the users group with a GID of 100.

This setting affects the primary group ID field of the **/etc/passwd** file highlighted below:

```
adavtyan@artur-lpt:~$ cat /etc/passwd | grep adavtyan
adavtyan:x:1000:1000:Artur Davtyan,,,:/home/ adavtyan:/bin/bash
adavtyan@artur-lpt:~$
```

*Note: The **-g** option to the **useradd** command allows you to use a different primary group than the default when creating a new user account.*



# User Configuration File

## Home: HOME=/home

The /home directory is the default base directory under which the user's new home directory is created. This means that a user with an account name of adavtyan would have a home directory of /home/adavtyan.

This setting affects the home directory field of the `/etc/passwd` file highlighted below:

```
adavtyan@artur-lpt:~$ cat /etc/passwd | grep adavtyan
adavtyan:x:1000:1000:Artur Davtyan,,,:/home/adavtyan:/bin/bash
adavtyan@artur-lpt:~$
```

The `-b` option to the `useradd` command allows you to use a different base directory group than the default when creating a new user account.

## Inactive: INACTIVE=-1

This value represents the number of days after the password expires that the account is disabled. A value of -1 means this feature is not enabled by default and no "inactive" value is provided for new accounts by default.

This setting affects the inactive field of the `/etc/shadow` file highlighted below:

```
adavtyan:pw:15020:5:30:7:60:15050
```

The `-f` option to the `useradd` command allows you to use a different INACTIVE value than the default when creating a new user account.

# User Configuration File

## *Expire: EXPIRE=*

By default, there is no value set for the expiration date. Usually, an expiration date is set on an individual account, not all accounts by default. For example, if you had a contractor that was hired to work until the end of the day on November 1, 2019, then you could ensure that they would be unable to log in after that date by using the EXPIRE field.

This setting affects the expire field of the `/etc/shadow` file highlighted below:

```
adavtyan:pw:15020:5:30:7:60:15050
```

The `-e` option to the `useradd` command allows you to use a different EXPIRE value than the default when creating a new user account.

## *Shell: Shell=/bin/bash*

The SHELL setting indicates the default shell for a user when they log in to the system.

This setting affects the shell field of the `/etc/passwd` file highlighted below:

```
adavtyan@artur-lpt:~$ cat /etc/passwd | grep adavtyan
adavtyan:x:1000:1000:Artur Davtyan,,,: /home/adavtyan:/bin/bash
adavtyan@artur-lpt:~$
```

The `-s` option to the `useradd` command allows you to use a different login shell than the default when creating a new user account.



# User Configuration File

## *Skeleton Directory: **SKEL=/etc/skel***

The SKEL value determines which skeleton directory has its contents copied into the new user's home directory. The contents of this directory are copied into the new user's home directory, and the new user is given ownership of the new files. This setting provides administrators with an easy way to populate a new user account with key configuration files.

The **-k** option to the **useradd** command allows you to use a different SKEL directory than the default when creating a new user account.

## *Create Mail Spool: **CREATE\_MAIL\_SPOOL=yes***

A mail spool is a file where incoming email is placed. Currently, the value for creating a mail spool is yes, which means that users by default are configured with the ability to receive and store local mail. If you are not planning on using local mail, then this value could be changed to no.

To modify one of the useradd default values, the **/etc/default/useradd** file could be edited with a text editor. Another (safer) technique is to use the **useradd -D** command.

For example, if you wanted to allow users to have expired passwords that they could still log in with for up to thirty days, then you could execute:

```
adavtyan@artur-lpt:~$  
adavtyan@artur-lpt:~$ sudo useradd -D -f 30
```

# Account Considerations

Creating a user account for use with a Linux system may require that you gather several pieces of information. While all that may be required is the account name, you may also want to plan the UID, the primary group, the supplementary groups, the home directory, the skeleton directory, and the shell to be used. When planning these values, consider the following:

**Username:** The only required argument for the `useradd` command is the name you want the account to have. The username should follow the same guidelines as for group names. Following these guidelines can help you to select a username that is portable:

*The first character of the name should be either an underscore “\_” character or a lower-case alphabetic a-z character. Up to 32 characters are allowed on most Linux distributions, but using more than 16 can be problematic as some distributions may not accept more than 16. After the first character, the remaining characters can be alphanumeric, a dash “-” character or an underscore “\_” character. The last character should not be a hyphen “-” character.*

If the user needs to access multiple systems, it is usually recommended to have the account name be the same on those systems. The account name must be unique for each user.

```
adavtyan@artur-lpt:~$  
adavtyan@artur-lpt:~$ sudo useradd anna
```

**User Identifier (UID):** Once you create a user with a specific UID, the system generally increments the UID by one for the next user that you create. If attached to a network with other systems, you may want to ensure that this UID is the same on all systems to help provide consistent access.

Adding the `-u` option to the `useradd` command allows you to specify the UID number. UIDs typically can range anywhere from zero to over four billion, but for greatest compatibility with older systems, the maximum recommended UID value is 60,000.

```
adavtyan@artur-lpt:~$  
adavtyan@artur-lpt:~$ sudo useradd -u 1100 anna
```

# Account Considerations

The root user has a UID of 0, which allows that account to have special privileges. Any account with a UID of 0 would effectively be able to act as the administrator.

System accounts are generally used to run background services called daemons. By not having services run as the root user, the amount of damage that can be done with a compromised service account is limited. System accounts used by services generally use UIDs that are in the reserved range. One system account that is an exception to this rule is the user `nfsnobody`, which has a UID of 65534.

The reserved range used for service accounts has expanded over time. Initially, it was for UIDs between 1 and 99. Then, it expanded to be between 1 and 499. The current trend among distributions is that system accounts are any account that has a UID between 1 and 999, but the range 1-499 is also still commonly used.

When setting up a new system, it is a good practice to start UIDs no lower than 1000 ensuring there are sufficient UIDs available for many system services and giving you the ability to create many GIDs in the reserved range.

**Primary Group:** In distributions which feature UPG, this group is created automatically with a GID and group name that matches the UID and username of the newly created user account. In distributions not using UPG, the primary group ordinarily defaults to the users group with a GID of 100.

To specify a primary group with the `useradd` command, use the `-g` option with either the name or GID of the group. For example, to specify users as the primary group:

```
adavtyan@artur-lpt:~$  
adavtyan@artur-lpt:~$ sudo useradd -g users
```

# Account Considerations

**Supplementary Group:** To make the user a member of one or more supplementary groups, the **-G** option can be used to specify a comma-separated list of group names or numbers. For example to specify sales and research as supplementary groups:

```
adavtyan@artur-lpt:~$  
adavtyan@artur-lpt:~$ sudo useradd -G developer,research
```

**Home Directory:** By default, most distributions create the user's home directory with the same name as the user account underneath whichever base directory is specified in the HOME setting of the /etc/default/useradd file, which typically specifies the /home directory. For example, if creating a user account named jane, the user's new home directory would be /home/anna.

```
adavtyan@artur-lpt:~$ grep '/home/anna' /etc/passwd  
anna:x:1001:1001::/home/anna:/bin/sh
```

There are several options for the **useradd** command that can affect creating the user's home directory

- If CREATE\_HOME is set to no or this setting is not present, then the directory will not be created automatically. Otherwise, the **-M** option is used to specify to the useradd command that it should not create the home directory, even if CREATE\_HOME is set to yes.
- If the CREATE\_HOME setting in the /etc/login.defs file is set to yes, the home directory is created automatically. Otherwise, the **-m** option can be used to make the home directory.
- The **-b** option allows you to specify a different base directory under which the user's home directory is created.
- The **-d** option allows you to specify either an existing directory or a new home directory to create for the user. This should be a full path for the user's home directory.
- The **-k** option to specifies a different skeleton directory. When using the **-k** option, the **-m** option is required.

```
adavtyan@artur-lpt:~$ sudo useradd -m anna  
adavtyan@artur-lpt:~$ ls -ld /home/anna  
drw-r-xr-x 2 anna anna 4096  Uuy  6 18:13 /home/anna
```

# Account Considerations

## Skeleton Directory

By default, the contents of the `/etc/skel` directory are copied into the new user's home directory. The resulting files are also owned by the new user. By using the `-k` option with the `useradd` command, the contents of a different directory can be used to populate a new user's home directory. When specifying the skeleton directory with the `-k` option, the `-m` option must be used or else the `useradd` command will fail with an error.

The following example uses `/home/sysadmin` as the skeleton directory:

```
adavtyan@artur-lpt:~$ sudo useradd -mk /home/sysadmin jane
adavtyan@artur-lpt:~$ sudo ls /home/jane
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

## Shell

While the default shell is specified in the `/etc/default/useradd` file, it can also be overridden with the `useradd` command using the `-s` option at the time of account creation:

```
adavtyan@artur-lpt:~$ sudo useradd -s /bin/bash jane
```

It is common to specify the `/sbin/nologin` shell for accounts to be used as system accounts.

## Comment

The comment field, originally called the General Electric Comprehensive Operating System (GECOS) field, is typically used to hold the user's full name. Many graphical login programs display this field's value instead of the account name. The `-c` option of the `useradd` command allows for the value of this field to be specified.

```
adavtyan@artur-lpt:~# sudo useradd -c 'Anna Stepanyan' /bin/bash anna
```

# Creating a User

Once you've verified which default values to use and you've gathered the information about the user, then you are ready to create a user account. An example of a **useradd** command using a few options looks like the following:

```
root@artur-lpt:~# useradd -u 1009 -g users -G sales,research -mc 'Jane' jane
```

This example of the **useradd** command creates a user with a UID of 1009, a primary group of users, supplementary memberships in the sales and research groups, a comment of "Jane Doe", and an account name of jane.

After executing the previous command, the information about the jane user account is automatically added to the `/etc/passwd` and `/etc/shadow` files, while the information about supplemental group access is automatically added to the `/etc/group` file:

```
root@artur-lpt:~# grep jane /etc/passwd
jane:x:1009:100:Jane Doe:/home/jane:/bin/sh
root@artur-lpt:~# grep jane /etc/shadow
jane::!17883:0:99999:7:30::
root@artur-lpt:~# grep jane /etc/group
research:x:1005:jane
sales:x:999:jane
```

In addition, if the `CREATE_MAIL_SPOOL` is set to yes then the mail spool file `/var/spool/mail/jane` is created:

```
root@artur-lpt:~# ls /var/spool/mail
jane root rpc sysadmin
```

# Passwords

Choosing a good password is not an easy task, but it is critical that it is done properly or the security of an account (perhaps the entire system) could be compromised. Picking a good password is only a start; you need to be very careful with your password so that it is not revealed to other people. You should never tell anyone your password and never let anyone see you type your password. If you do choose to write down your password, then you should store it securely in a place like a safe or safe deposit box.

There are numerous factors to consider when you are trying to choose a password for an account:

- **Length:** The `/etc/login.defs` file allows the administrator to specify the minimum length of the password. While some believe that the longer the password, the better, this isn't really correct. The problem with passwords that are too long is that they are not easily remembered and, as a result, they are often written down in a place where they can easily be found and compromised.
- **Composition:** A good password should be composed of a combination of alphabetic, numeric and symbolic characters.
- **Lifetime:** The maximum amount of time that a password can be used should be limited for several reasons:
  - If an account is compromised and the time that the password is valid is limited, the intruder will ultimately lose access when the password becomes invalid.
  - If an account is not being used, then it can automatically be disabled when the password is no longer valid.
  - If attackers are attempting a "brute-force" attack by trying every possible password, then the password can be changed before the attack can succeed

Opinions vary about how often users should be forced to change their passwords. For highly-sensitive accounts, it is recommended to change passwords more frequently, such as every 30 days. On the other hand, for non-critical accounts without any access to sensitive information, there is less need for frequent change. For accounts with minimal risk, having a duration of 90 days would be considered more reasonable.



# Setting a User Password

There are several ways for a user password to be changed. The user can execute the `passwd` command, the administrator can execute the `passwd` command providing the username as an argument, or graphical tools are also available.

The administrator can use the `passwd` command to either set the initial password or change the password for the account. For example, if the administrator had created the account `jane`, then executing `passwd jane` provides the administrator a prompt to set the password for the `jane` account. If completed successfully, then the `/etc/shadow` file will be updated with the user's new password.

While regular users must follow many password rules, the root user only needs to follow one rule: the password cannot be left blank. When the root user violates all other password rules that normally apply to regular users, it results in a warning being printed to the screen and the rule not being enforced:

```
root@artur-lpt:~# passwd jane
Enter new UNIX password:
BAD PASSWORD: it is WAY to short
BAD PASSWORD: is too simple
Retype new UNIX password:
```

Assuming that the administrator has set a password for a user account, the user can then log in with that account name and password. After the user opens a terminal, they can execute the `passwd` command with no arguments to change their own password. They are prompted for their current password and then prompted to enter the new password twice.

Using the privileges of the root user, the encrypted passwords and other password-related information can be viewed by viewing the `/etc/shadow` file. Recall that regular users cannot see the contents of this file.

# Modifying User

The **usermod** command offers many options for modifying an existing user account. Many of these options are also available with the **useradd** command at the time the account is created. The following chart provides a summary of the **usermod** options:

Several of these options are worthy of discussion because of how they impact user management. It can be very problematic to change the user's UID with the **-u** option, as any files owned by the user will be orphaned. On the other hand, specifying a new login name for the user with **-l** does not cause the files to be orphaned.

Deleting a user with the **userdel** command can either orphan or remove the user's files on the system. Instead of deleting the account, another choice is to lock the account with the **-L** option for the **usermod** command. Locking an account prevents the account from being used, but ownership of the files remains.

There are some important things to know about managing the supplementary groups. If you use the **-G** option without the **-a** option, then you must list all the groups to which the user would belong. Using the **-G** option alone can lead to accidentally removing a user from all the former supplemental groups that the user belonged to.

If you use the **-a** option with **-G** then you only have to list the new groups to which the user would belong. For example, if the user **jane** currently belongs to the sales and research groups, then to add her account to the development group, execute the following command:

```
adavtyan@artur-lpt:~$ sudo su -  
root@artur-lpt:~# usermod -a -G development jane  
root@artur-lpt:~# id jane  
uid=1001(jane) gid=1001(jane) groups=1001(jane),1100(development)
```

# Deleting User

The **userdel** command is used to delete users. When you delete a user account, you also need to decide whether to delete the user's home directory. The user's files may be important to the organization, and there may even be legal requirements to keep the data for a certain amount of time, so be careful not to make this decision lightly. Also, unless you've made backup copies of the data, once you've executed the command to delete the user and their files, there is no reversing the action.

To delete the user jane without deleting the user's home directory /home/jane, execute:

```
root@artur-lpt:~#  
root@artur-lpt:~# userdel jane  
root@artur-lpt:~#
```

Beware that deleting a user without deleting their home directory means that the user's home directory files will be orphaned and these files will be owned solely by their former UID and GID.

To delete the user, home directory, and mail spool as well, use the **-r** option:

```
root@artur-lpt:~#  
root@artur-lpt:~# userdel -r jane  
root@artur-lpt:~#
```

# File Ownership

By default, users own the files that they create. While this ownership can be changed, this function requires administrative privileges. Although most commands usually show the user owner as a name, the operating system is associating the user ownership with the UID for that username.

Every file also has a group owner. By default, the primary group of the user who creates the file is the group owner of any new files. Users are allowed to change the group owner of files they own to any group that they belong to. Similar to user ownership, the association of a file with a group is not done internally by the operating system by name, but by the GID of the group.

Since ownership is determined by the UID and GID associated with a file, changing the UID of a user (or deleting the user) has the effect of making a file that was originally owned by that user have no real user owner. When there is no UID in the `/etc/passwd` file that matches the UID of the owner of the file, then the UID (the number) is displayed as the user owner of the file instead of the username (which no longer exists). The same occurs for groups.

The `id` command can be useful for verifying which user account you are using and which groups you have available to use. By viewing the output of this command, you can see the user's identity information expressed both as a number and as a name.

The output of the `id` command displays the UID and user account name of the current user followed by the GID and group name of the primary group and the GIDs and group names of all group memberships:

```
adavtyan@artur-lpt:~$ id
uid=1000(adavtyan) gid=1000(adavtyan) groups=1000(adavtyan),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),999(docker)
```

The above example shows the user has a UID of 1000 for the user account `sysadmin`. It also shows that the primary group for this user has a GID of 1000 for the group `sysadmin`.

# Changing Groups

If you know that the file you are about to create should belong to a group different from your current primary group, then you can use the `newgrp` command to change your current primary group.

```
newgrp group_name
```

The `id` command lists your identity information, including your group memberships. If you are only interested in knowing what groups you belong to, then you can execute the `groups` command:

```
adavtyan@artur-lpt:~$ groups
adavtyan adm cdrom sudo dip plugdev lpadmin docker
```

The output of the `groups` command may not be as detailed as the output of the `id` command, but if all you need to know is what groups you can switch to by using the `newgrp` command, then the `groups` command provides the information that you need. The `id` command output does show your current primary group, so it is useful for verifying that the `newgrp` command succeeded.

```
adavtyan@artur-lpt:~$ newgrp sudo
adavtyan@artur-lpt:~$ id
uid=1000(adavtyan) gid=27(sudo) groups=,4(adm),24(cdrom),30(dip),46(plugdev),116(lpadmin),999(docker),1000(adavtyan)
adavtyan@artur-lpt:~$ touch /tmp/filetest
adavtyan@artur-lpt:~$ ls -l /tmp/filetest
-rw-rw-r-- 1 adavtyan sudo 0 Uuy 26 17:46 /tmp/filetest
adavtyan@artur-lpt:~$
```

The `newgrp` command opens a new shell; as long as the user stays in that shell, the primary group won't change. To switch the primary group back to the original, the user can leave the new shell by running the `exit` command.

# Changing Group and User Ownership

To change the group owner of an existing file the `chgrp` command can be used.

```
chgrp group_name file
```

As the root user, the `chgrp` command can be used to change the group owner of any file to any group. As a user without administrative privileges, the `chgrp` command can only be used to change the group owner of a file to a group that the user is already a member of:

To change the group ownership of all of the files of a directory structure, use the recursive `-R` option to the `chgrp` command. For example, the command in the following example would change the group ownership of the `test_dir` directory and all files and subdirectories of the `test_dir` directory.

```
chgrp -R group_name directory
```

The `chown` command allows the root user to change the user ownership of files and directories. A regular user cannot use this command to change the user owner of a file, even to give the ownership of one of their own files to another user. However, the `chown` command also permits changing group ownership, which can be accomplished by either root or the owner of the file. There are three different ways the `chown` command can be executed.

- The first method is used to change just the user owner of the file.
- The second method is to change both the user and the group; this also requires root privileges. To accomplish this, you separate the user and group by either a colon or a period character.
- If a user doesn't have root privileges, they can use the third method to change the group owner of a file just like the `chgrp` command. To use `chown` only to change the group ownership of the file, use a colon or a period as a prefix to the group name:

```
chown user /path/to/file
```

```
chown user:group /path/to/file
```

```
chown .group /path/to/file
```

# Permissions

The output of the `ls -l` command displays ten characters at the beginning of each line. These characters indicate the type of file and the permissions of the file. For example, consider the output of the following command: *`ls -l /etc/passwd`*

```
-rw-r--r-- 1 root root 2951  6 18:13 /etc/passwd
```

## File Type

The first character of each line indicates the type of file:

The following table describes the possible values for the file type:

Character	Type of the File
-	A regular file, which may be empty, or contain text or binary data.
d	A directory file, which contains the names of other files and links to them.
l	A symbolic link is a file name that refers (points) to another file.
b	A block file is one that relates to a block hardware device where data is read in blocks of data.
c	A character file is one that relates to a character hardware device where data is read one byte at a time.
p	A pipe file works similar to the pipe symbol, allowing for the output of one process to communicate to another process through the pipe file, where the output of the one process is used as input for the other process.
s	A socket file allows two processes to communicate, where both processes are allowed to either send or receive data.



# Permission Groups

The next nine characters demonstrate the permissions of the file.

```
-rw-r--r-- 1 root root 2951 Uuy 6 18:13 /etc/passwd
```

The permissions set on these files determine the level of access that a user has on the file. When a user runs a program and the program accesses a file, then the permissions are checked to determine whether the user has the correct access rights to the file.

The permissions are grouped into three different roles, representing the different users that may try to access the file. If you aren't the owner and you're not a member of the **file/directory** group, then your permissions would be others.

**User Owner:** Characters 2-4 indicate the permissions for the user that owns the file. If you are the owner of the file, then only the user owner permissions are used to determine access to that file.

```
-rw-r--r-- 1 root root 2951 Uuy 6 18:13 /etc/passwd
```

**Group Owner:** Characters 5-7 indicate the permissions for the group that owns the file. If you are not the owner but are a member of the group that owns the file, then only group owner permissions are used to determine access to that file.

```
-rw-r--r-- 1 root root 2951 Uuy 6 18:13 /etc/passwd
```

**Other Permissions:** Characters 8-10 indicate the permissions for others or what is sometimes referred to as the world's permissions. This group includes all users who are not the file owner or a member of the file's group.

```
-rw-r--r-- 1 root root 2951 Uuy 6 18:13 /etc/passwd
```

# Permission Type

Each group is attributed three basic types of permissions: read, write, and execute.

User Owner			Group Owner			Other		
Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
r	w	x	r	w	x	r	w	x

## Read

The first character of each group represents the read permission. There is an r character if the group has the read permission, or a - character if the group does not.

- On a file, this allows processes to read the contents of the file, meaning the contents can be viewed and copied.
- On a directory, file names in the directory can be listed, but other details are not available.

## Write

The second character of each group represents the write permission. There is a w character if the group has the write permission, or a - character if the group does not.

- A file can be written to by the process, so changes to a file can be saved. Note that w permission really requires r permission on the file to work correctly.
- On a directory, files can be added to or removed from the directory. Note that w permission requires x permission on the directory to work correctly.

## Execute

The third character of each group represents the execute permission. There is an x character if the group has the execute permission, or a - character if the group does not.

- A file can be executed or run as a process.
- On a directory, the user can use the cd command to "get into" the directory and use the directory in a pathname to access files and, potentially, subdirectories under this directory.

# Symbolic Method

The **chmod** (change mode) command is used to change permissions on files and directories. There are two techniques that can be used with this command: symbolic and numeric. Both techniques use the following basic syntax:

```
chmod new_permission file_name
```

**Symbolic Method:** If you want to modify some of the current permissions, the symbolic method is usually easier to use. With this method, you specify which permissions you want to change on the file, and the other permissions remain as they are. When specifying the *new\_permission* argument of the **chmod** command using the symbolic method three types of information are required.

*Start by using one or more of the following characters to indicate which permission group(s) to apply the changes to:*

u	user owner
g	group owner
o	others
a	all (user owner, group owner, and others)

*Then choose one of the following operators to indicate how to modify the permissions:*

+	add
-	remove
=	equals

Lastly, use the following characters to specify the permissions type(s) to change:

r	read
w	write
e	execute

# Symbolic Method

For example, to give the group owner write permission on a file named abc.txt, you could use the following command:

```
adavtyan@artur-lpt:~/tmp/Documents$ chmod g+w abc.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls -l abc.txt
-rw-rw-r-- 1 adavtyan adavtyan 0 Uu 26 19:41 abc.txt
```

Only the group owner's permission was changed. All other permissions remained as they were prior to the execution of the chmod command. You can combine values to make multiple changes to the file's permissions.

For example, consider the following command which adds the execute permission to the user owner and group owner and removes the read permission for others:

```
adavtyan@artur-lpt:~/tmp/Documents$ chmod ug+x,o-r abc.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls -l abc.txt
-rwxrwx--- 1 adavtyan adavtyan 0 Uu 26 19:41 abc.txt
```

Lastly, you could use the = character, which adds specified permissions and causes unmentioned ones to be removed. For example, to give the user owner only read and execute permissions, removing the write permission:

```
adavtyan@artur-lpt:~/tmp/Documents$ chmod u=rx abc.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls -l abc.txt
-r-xrwx--- 1 adavtyan adavtyan 0 Uu 26 19:41 abc.txt
```

# Numeric Method

The numeric method (also called the octal method) is useful when changing many permissions on a file. It is based on the octal numbering system in which each permission type is assigned a numeric value:

4	read
2	write
1	execute

By using a combination of numbers from 0 to 7, any possible combination of read, write and execute permissions can be specified for a single permission group set. For example:

7	rwx		3	-wx
6	rw-		2	-w-
5	r-x		1	--x
4	r--		0	---

The `new_permission` argument is specified as three numbers, one number for each permission group. When the *numeric method* is used to change permissions, all nine permissions must be specified. Because of this, the symbolic method is generally easier for changing a few permissions while the numeric method is better for changes that are more drastic.

For example, to set the permissions of a file named *abc.txt* to be *rwxr-xr--* you could use the following command:

```
adavtyan@artur-lpt:~/tmp/Documents$ chmod 754 abc.txt
adavtyan@artur-lpt:~/tmp/Documents$ ls -l abc.txt
-rwxr-xr-- 1 adavtyan adavtyan 0 26 19:41 abc.txt
```

# Default Permissions

The **umask** command is a feature that is used to determine default permissions that are set when a file or directory is created. Default permissions are determined when the umask value is subtracted from the maximum allowable default permissions. The maximum default permissions are different for files and directories:

file	rw-rw-rw-
directories	rw-rw-rwx

The permissions that are initially set on a file when it is created cannot exceed rw-rw-rw-. To have the execute permission set on a file, you first need to create the file and then change the permissions.

The **umask** command can be used to display the current umask value:

```
adavtyan@artur-lpt:~/tmp/Documents$ umask
0002
```

A breakdown of the output:

- The first 0 indicates that the umask is given as an octal number.
- The second 0 indicates which permissions to subtract from the default user owner's permissions.
- The third 0 indicates which permissions to subtract from the default group owner's permissions.
- The last number 2 indicates which permissions to subtract from the default other's permissions.

```
adavtyan@artur-lpt:~/tmp/Documents$ sudo su
root@artur-lpt:/home/adavtyan/tmp/Documents$ umask
0022
```

*Note that different users may have different umasks. Typically the root user has a more restrictive umask than normal user accounts:*

## Default Permissions

To understand how umask works, assume that the umask of a file is set to 027 and consider the following:

File Default	666	
Umask	-027	
Result	640	rw-r-----

Because the default permissions for directories are different than for files, a umask of 027 would result in different initial permissions on new directories:

Directory Default	777	
Umask	-027	
Result	750	rw-xr-x---

*Example:*

```
adavtyan@artur-lpt:~$ umask 027
adavtyan@artur-lpt:~$ touch sample
adavtyan@artur-lpt:~$ ls -l sample
-rw-r----- 1 adavtyan adavtyan 0  Uuy  6 18:22 sample
adavtyan@artur-lpt:~$ mkdir test-dir
adavtyan@artur-lpt:~$ ls -ld test-dir/
drwxr-x--- 2 adavtyan adavtyan 4096  Uuy  6 18:23 test-dir/
```

The new umask is only applied to file and directories created during that session. When a new shell is started, the default umask will again be in effect. Permanently changing a user's umask requires modifying the `.bashrc` file located in that user's home directory.



# Linux Services

A service is a program that runs in the background outside the interactive control of system users as they lack an interface. This in order to provide even more security, because some of these services are crucial for the operation of the operating system.

We can list all services in Linux

```
adavtyan@artur-lpt:~$ sudo systemctl list-unit-files --type service --all
```

When the command is run, we will see all the services that are on the system. However, we will also see that some have a defined status. Let's learn what all these mean.

- **Enabled** services are currently running. They usually have no problems.
- **Disabled** services are not active but can be activated at any time without a problem.
- **Masked** services won't run unless we take that property away from them.
- **Static** services will only be used in case another service or unit needs it.
- Finally, there are services **generated** through a SysV or LSB initscript with systemd generator.

In case we want to know only the services that are active, we have to use a command together with **grep**, like so:

```
adavtyan@artur-lpt:~$ sudo systemctl | grep running
```

# Managing Linux Services

Now it is time to learn how to manage a specific service. Note that each service represents software that works differently. In this tutorial, we will only show how to start, check the status of and stop services – the basic controls

*To start a service on Linux, we need to run the following command:*

```
adavtyan@artur-lpt:~$ sudo systemctl start [service_name]
```

*If the service is correctly configured, it will start. Now, if we want to stop it, we will use the following command:*

```
adavtyan@artur-lpt:~$ sudo systemctl stop [service_name]
```

*Meanwhile, to check the status of a service we can use:*

```
adavtyan@artur-lpt:~$ sudo systemctl status [service_name]
```

*It is also possible to have a service run while the operating system is being loaded:*

```
adavtyan@artur-lpt:~$ sudo systemctl enable [service_name]
```

*Or remove it from the initial load:*

```
adavtyan@artur-lpt:~$ sudo systemctl disable [service_name]
```

# Install Linux NGINX Services

*Install Nginx in the ubuntu*

```
adavtyan@artur-lpt:~$ sudo apt install nginx
```

*To check the status of a service we can use:*

```
adavtyan@artur-lpt:~$ sudo systemctl status nginx
```

*Set that service run while the operating system is being loaded:*

```
adavtyan@artur-lpt:~$ sudo systemctl status nginx
```

*Where story index.html file in the Nginx*

```
adavtyan@artur-lpt:~$ ls -la /var/www/html
```

*Where we can find nginx configs in the server*

```
adavtyan@artur-lpt:~$ ls -la /etc/nginx
```

**Thank you for your attention !**

**Q&A**

