Linux Command Line

Linux distributions

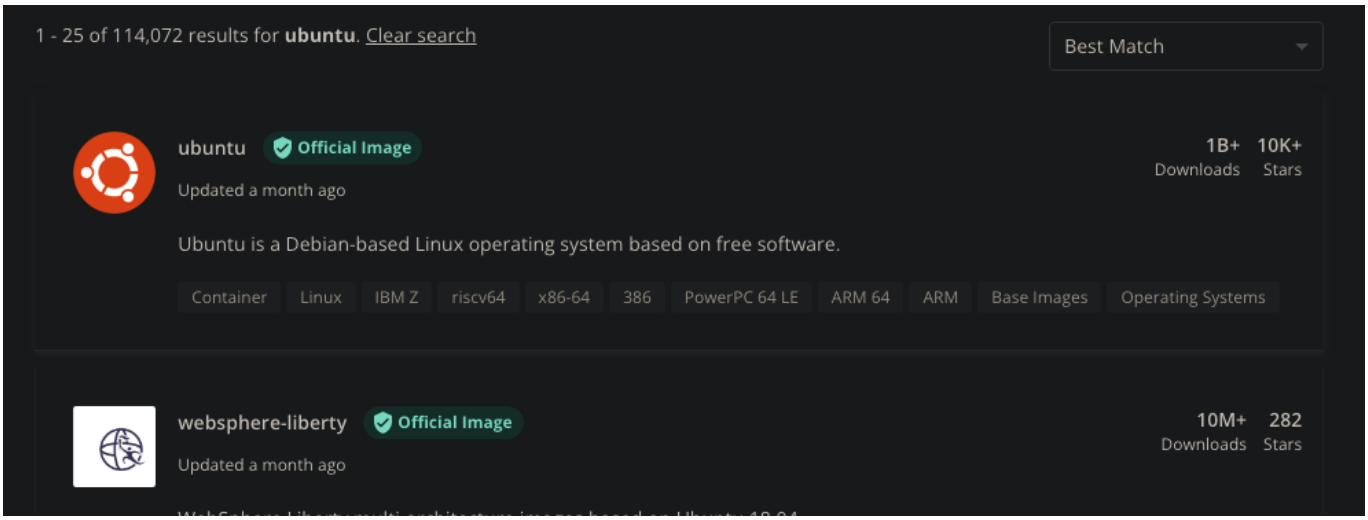
Linux is an open-source OS. Therefore, many development communities have built their our custom versions of it. We call them *Linux distributions* or *Linux distros* for short. Some of the most popular are:

- Ubuntu (the one we'll use here)
- Debian
- Alpine
- Fedora
- CentOS

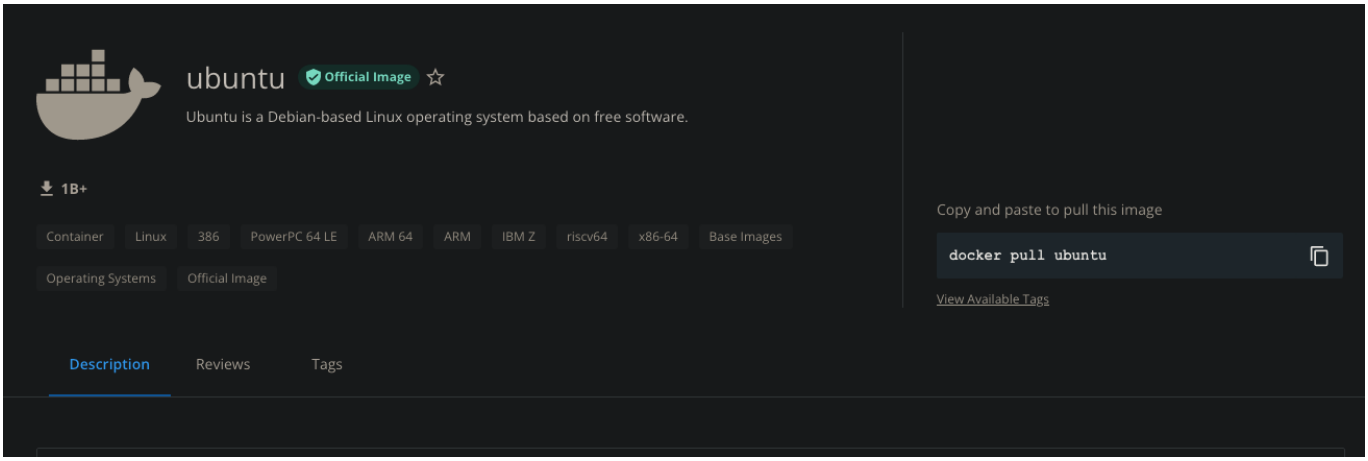
The commands are mostly the same for all of them.

Running Linux

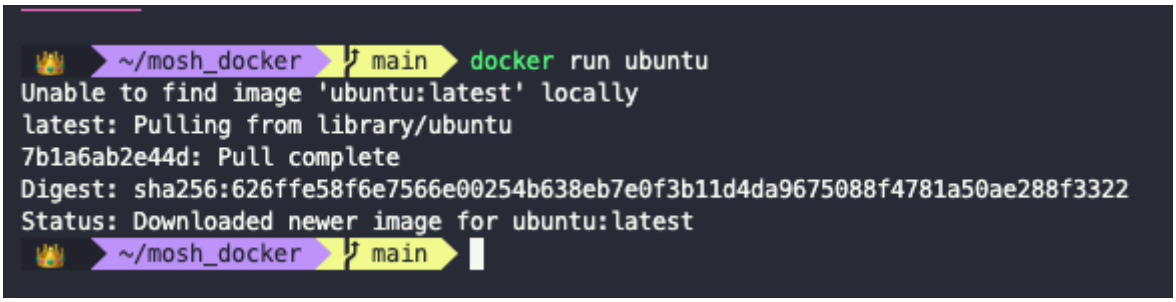
Head on to DockerHub and search for **ubuntu**.



Click on the image card and DockerHub will redirect you to its page. Here you can find the `docker pull` command to download this image into your computer. In this case that is simply `docker pull ubuntu`.

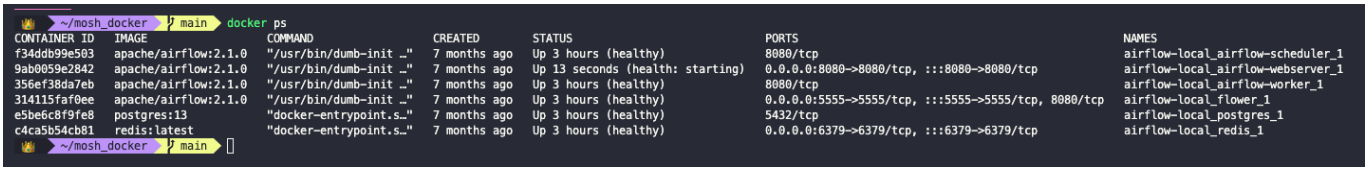


If instead you run `docker run ubuntu`, docker will first check if the image is locally available. If it is, it will run it. If it is not, it will pull it and then run it.



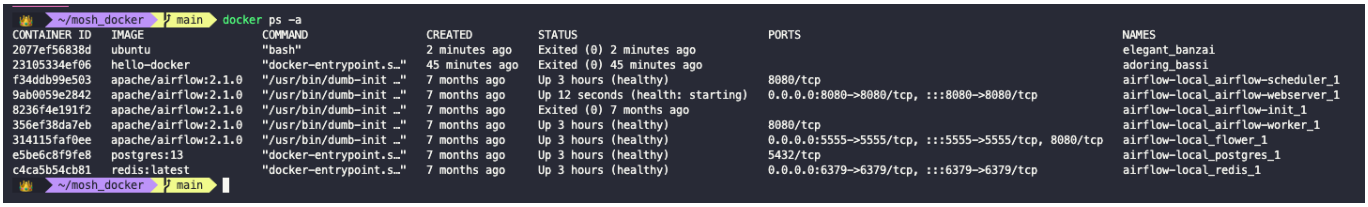
To check which containers are running we run:

```
docker ps
```



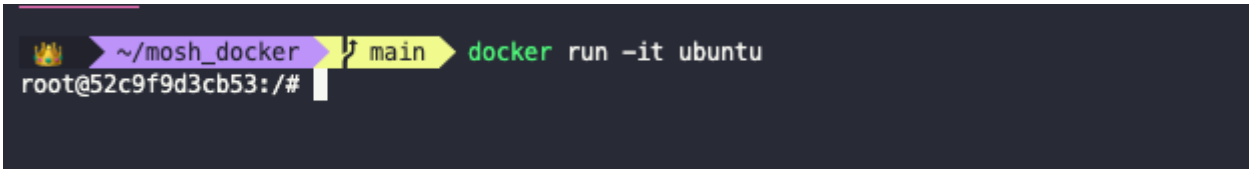
The `ubuntu` image is not being used in any of our containers and is therefore not shown. If we want to display all containers (running or otherwise), we need to add the `-a` flag (which stands for *all*):

```
docker ps -a
```



If we need to start a container in the interactive mode we need to add the `-it` flag, as well as the name of the image to run:

```
docker run -it <image_name>
```



What we have done is to open a shell in the container. This shell is not waiting for our commands. The first part of the prompt says that we are currently logged in as the `root` user. Following that comes the name of the machine. So user `root` is at (`@`) machine ID `52c9f9d3cb53`. After the colon (`:`), the forward slash (`/`) represents where we are in the file system. In this case, we are at the root directory. Lastly, the pound sign (`#`) means that we have the highest levels of privileges. If we had logged in as a regular user we use see a currency sign (`$`).

Some commands that we can use are:

- `echo` to print something to the terminal
- `whoami` will display the user name
- `echo $0` will display the location of the shell program
- `ls` will display all directories
- `history` will display a list of all the commands that we've run
- with `!#` where `#` is a number from the history, we can re-run that command

Managing packages

In Ubuntu we manage packages with `apt` (*Advances Package Tool*). If we run `apt` in the container, we can get a list of all the sub-commands that we can use

```
root@52c9f9d3cb53:/# apt
apt 2.0.6 (amd64)
Usage: apt [options] command

apt is a commandline package manager and provides commands for
searching and managing as well as querying information about packages.
It provides the same functionality as the specialized APT tools,
like apt-get and apt-cache, but enables options more suitable for
interactive use by default.

Most used commands:
  list - list packages based on package names
  search - search in package descriptions
  show - show package details
  install - install packages
  reinstall - reinstall packages
  remove - remove packages
  autoremove - Remove automatically all unused packages
  update - update list of available packages
  upgrade - upgrade the system by installing/upgrading packages
  full-upgrade - upgrade the system by removing/installing/upgrading packages
  edit-sources - edit the source information file
  satisfy - satisfy dependency strings

See apt(8) for more information about the available commands.
Configuration options and syntax is detailed in apt.conf(5).
Information about how to configure sources can be found in sources.list(5).
Package and version choices can be expressed via apt_preferences(5).
Security details are available in apt-secure(8).
                                        This APT has Super Cow Powers.
root@52c9f9d3cb53:/#
```

For example, we can use `apt list` to see the list of all available packages and their status.

```
root@52c9f9d3cb53:/# apt list
Listing... Done
adduser/now 3.118ubuntu2 all [installed,local]
apt/now 2.0.6 amd64 [installed,local]
base-files/now 11ubuntu5.4 amd64 [installed,local]
base-passwd/now 3.5.47 amd64 [installed,local]
bash/now 5.0-6ubuntu1.1 amd64 [installed,local]
bsdutils/now 1:2.34-0.1ubuntu9.1 amd64 [installed,local]
bzip2/now 1.0.8-2 amd64 [installed,local]
coreutils/now 8.30-3ubuntu2 amd64 [installed,local]
dash/now 0.5.10.2-6 amd64 [installed,local]
debconf/now 1.5.73 all [installed,local]
debianutils/now 4.9.1 amd64 [installed,local]
diffutils/now 1:3.7-3 amd64 [installed,local]
dpkg/now 1.19.7ubuntu3 amd64 [installed,local]
e2fsprogs/now 1.45.5-2ubuntu1 amd64 [installed,local]
fdisk/now 2.34-0.1ubuntu9.1 amd64 [installed,local]
findutils/now 4.7.0-1ubuntu1 amd64 [installed,local]
gcc-10-base/now 10.3.0-1ubuntu1~20.04 amd64 [installed,local]
gpgv/now 2.2.19-3ubuntu2.1 amd64 [installed,local]
grep/now 3.4-1 amd64 [installed,local]
gzip/now 1.10-0ubuntu4 amd64 [installed,local]
hostname/now 3.23 amd64 [installed,local]
init-system-helpers/now 1.57 all [installed,local]
libacl1/now 2.2.53-6 amd64 [installed,local]
libapt-pkg6.0/now 2.0.6 amd64 [installed,local]
libattr1/now 1:2.4.48-5 amd64 [installed,local]
```

If we don't see the package that we need in that list, we can run `apt update` and the list will be updated from a predefined list of sources.

```
root@52c9f9d3cb53:/# apt update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [33.6 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1108 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [797 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1758 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packages [50.8 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [21.7 kB]
Get:15 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [733 kB]
Get:16 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [1335 kB]
Get:17 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [30.1 kB]
Get:18 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [828 kB]
Fetched 20.1 MB in 2s (8110 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
root@52c9f9d3cb53:/#
```

To install a package we use.

```
apt install <package_name>
```

```
root@52c9f9d3cb53:/# apt install nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  hunspell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 269 kB of archives.
After this operation, 868 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 nano amd64 4.8-1ubuntu1 [269 kB]
Fetched 269 kB in 2s (164 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package nano.
(Reading database ... 4127 files and directories currently installed.)
Preparing to unpack .../nano_4.8-1ubuntu1_amd64.deb ...
Unpacking nano (4.8-1ubuntu1) ...
Setting up nano (4.8-1ubuntu1) ...
update-alternatives: using /bin/nano to provide /usr/bin/editor (editor) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz because associated file /usr/share/man/man1/nano.1.gz (of link group editor) doesn't exist
update-alternatives: using /bin/nano to provide /usr/bin/pico (pico) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/pico.1.gz because associated file /usr/share/man/man1/nano.1.gz (of link group pico) doesn't exist
root@52c9f9d3cb53:/#
```

Here we are installing `nano` which is a lightweight text editor for Ubuntu. To remove it we run:

```
apt remove nano
```

```
root@52c9f9d3cb53:/# nano
root@52c9f9d3cb53:/# apt remove nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  nano
0 upgraded, 0 newly installed, 1 to remove and 1 not upgraded.
After this operation, 868 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 4194 files and directories currently installed.)
Removing nano (4.8-1ubuntu1) ...
root@52c9f9d3cb53:/#
```

Linux file system

In Linux (just as with other OSs), files are organized in a tree. In Linux, everything is a file (including processes and directories). The root directory in Linux is just `/`. After that, standard directories include

- `bin` which includes all the binaries for that distro
- `boot` which includes all the files related to booting the system
- `dev` which is short for *devices* (not development) and includes all the files needed for accessing devices
- `etc` which is short for *editable text configurations* (not etcetera) and includes all the configuration files
- `home` which where all home directories (users) are stored
- `root` which is the home directory of the root user
- `lib` which is where software library dependencies are stored
- `var` which is short for *variable* and is where we store files that are update frequently (like logs)
- `proc` which is where we store files that represent running processes

```
/
|_ bin
|_ boot
|_ dev
|_ etc
|_ home
|_ root
|_ lib
|_ var
|_ proc
```

Navigating the file system

To navigate the files we need to use commands.

- `pwd` will *print working directory*
- `ls` will *list* the files in that directory. Some flags that we can add are:
  - `-1` for listing them one per row
  - `-l` for long listing (includes more details)
  - `<path>` will show the files in that path (without having to navigate to it)
- `cd` will *change directory*. We can use either relative or absolute paths:

- relative paths are relative to the current working directory
- absolute paths always start from the root `/`
- `cd ..` will take us one level up in the tree
- `cd ../..` will take us two levels up in the tree
- `cd ~` will take us to the home directory of the current user

Manipulating files and directories

- `mkdir` will create a new directory as a child of the current directory
- we use the `mv <current_location> <new_location>` to *move* a file or rename it
- `touch <file_name>` will create a new file
- `touch <file_name> ... <file_name>` will create all the files that we list
- `rm <file_name> ... <file_name>` will delete all the files that we list. We can also use patterns.
- `rm -r <directory_name>` will delete the directory and all its files

Viewing and editing files

To view and edit files we can use `nano`. We open the file in nano by running.

```
nano <file_name>
```

We ca ¿n now make changes to the file in the text editor. To exit a file we press `control + x`.

To see the content of a file we run:

```
cat <file_name>
```

`cat` is short for *concatenate* and we can use it to combine files. When viewing files, `cat` is only useful for short files. For longer files we use:

```
more <path/to/file>
```

When using `more` we can scroll down one page at a time by pressing the space bar, or one line at a time with the enter key We can exit by pressing `q`. But we can only scroll down. If we want to be able to scroll both up and down we run:

```
less <path/to/file>
```

When using `less` we can scroll both up and down by using the arrows. The space bar, enter and `q` still work with `less`.

To look just at the head of a file we can use the `head` command followed by `-n #` where `#` is the number of lines.

```
head -n 5 <file_name>
```

Similarly, we can use `tail` to see the last `n` lines.

```
tail -n 5 <file_name>
```

Redirection

Tow basic concepts in Linux are

- Standard input => keyboard
- Standard output => screen

But we can change the sources of either input or output. This is referred to as *redirection*. For example, `cat file1.txt` will print the content of `file1.txt` to the screen. But we can redirect the output by using the redirection operator (`>`) and send it to `file2.txt`.

Similarly, we can use the `<` sign to redirect the standard input.

Searching for text

To search for text in a file we use the `grep` command, which stands for *global regular expression*. This search is case sensitive. We use it by supplying a string pattern and a file name.

```
grep Hello file1.txt
```

We can remove case sensitivity with the `-i` flag.

```
grep -i hello file1.txt
```

We can search in multiple files by supplying multiple file names or by supplying a file name pattern.

To search in directories, we can supply a directory name, or use `.` to refer to the current directory with the `-r` flag.

Finding files and directories

To find files and directories we use the `find` command. If we run it without any arguments, it will list all files and directories in the current directory recursively. To search somewhere else, we need to supply a path.

If we only want to list directories we need to add the `-type` option with the value of `d`:

```
find -type d
```

Likewise, we can use the `f` value for only getting files:

```
find -type f
```

We can add patterns. For example, to search for all files who's names start with `f` we use:

```
find -type f -name "f*"
```

To make the search case insensitive we use:

```
find -type f -iname "f*"
```

Chaining commands

We can chain multiple commands with semi-colons (`;`).

```
mkdir test; cd test; echo done
```

If we want execution to stop at the first error, we use the *and* operator (`&&`).

```
mkdir test && cd test && echo done
```

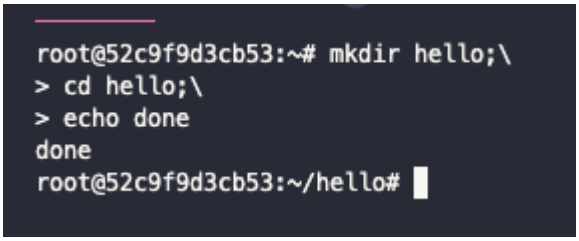
Likewise, we can use an *or* operator (`||`)

```
mkdir test || echo "directory exists"
```

We can also use piping (`|`) to chain command.

```
ls /bin | less
```

We can write multi-line commands by using a backslash (`\`) and pressing enter.

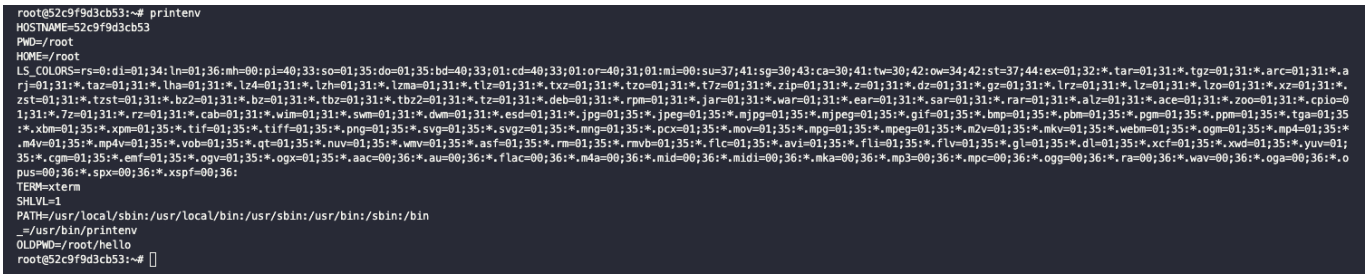


Environment variables

We use environment variables to set or store settings for our application. To view the environment variables we run:

```
printenv
```

The return value is a list of `key=value` pairs.



An important one to understand is `PATH`. Its value is the semi-colon separated list of paths that the OS will search for when requested to execute a program. If we only want to receive the value of one env variable (for example, `PATH`) we can use either of the following:

```
printenv PATH
```

```
echo $PATH
```

To set the value of an environment variable we use:

```
export VAR_NAME=value
```

This variable will be set for the current session. If we want the variables to be available when re-starting, we need to include them in the `.bashrc` file. This is the user's personal startup file. We can achieve this with chaining:

```
echo VAR_NAME=value >> .bashrc
```

The `>>` means append (as opposed to re-write the entire file). This changes will only be available starting from the next terminal session. If we need the `.bashrc` file to be re-loaded, we need to use:

```
source ~/.bashrc
```

Managing processes

A process is an instance of a running program. To see all the running processes we use:

```
ps
```

To put a process in the background we use include an ampersand (`&`) after it. For example, the following will wait for 100 seconds in the background and still let us use the terminal.

```
sleep 100 &
```

To kill a process we use the `kill` command and supply the process ID number.

Managing users

To manage users we use the following commands:

- `useradd` to add a new user
- `usermod` to modify a user
- `userdel` to delete a user

To create a user called `john` with a home directory we run:

```
useradd -m john
```

To see the user we run:

```
cat /etc/passwd
```

```
root@52c9f9d3cb53:~# useradd -m john
root@52c9f9d3cb53:~# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
john:x:1000:1000::/home/john:/bin/sh
root@52c9f9d3cb53:~#
```

We can see in the last line that the user `john` was added. This output prints all the users and their account information, separated by colons (`:`).

- first comes the user name
- the following `x` means that their password is store somewhere else
- the first `1000` is the user ID
- then comes the group ID (in this case also `1000`)
- next we can see the home directory of this user (in this case `/home/john`)
- lastly we see the shell program used when this user logs in (in this case `bin/sh`)

We can modify the shell program as follows:

```
usermod -s /bin/bash john
```

The user passwords can be found in the `shadow` file:

```
cat /etc/shadow
```

This file is only accessible to the root user, and passwords are stored in encrypted format.

To login is as a different user we open a new terminal window. First we need to run `docker ps` to get the container ID. Then we need to execute a bash session inside the container.

```
docker exec -it -u USER_NAME CONTAINER_ID bash
```

```
~/mosh_docker main ± docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED        STATUS        PORTS     NAMES
52c9f9d3cb53   ubuntu    "bash"    7 hours ago    Up 7 hours              condescending_lamport
~/mosh_docker main ± docker exec -it -u john 52c9f9d3cb53 bash
john@52c9f9d3cb53:/$
```

To delete the user we run

```
userdel USER_NAME
```

We can also create users with the newer `adduser` command. This command will use `useradd` under the hood, but is more verbose.

```
adduser USER_NAME
```

```
root@52c9f9d3cb53:~# adduser bob
Adding user `bob' ...
Adding new group `bob' (1001) ...
Adding new user `bob' (1001) with group `bob' ...
Creating home directory `/home/bob' ...
Copying files from `/etc/skel' ...
New password:
```

Here we can see that `bob` was created and added to a user group of the same name. This is the default behavior. Now it is asking us to set the password. After that, it will ask us to enter additional information. This command is interactive, so we don't want to use it when building automated workflows.

Managing groups

To manage groups we use the following commands:

- `groupadd` to add a new group
- `groupmod` to modify a group
- `groupdel` to delete a group

We use groups to manage access permissions.

We can checkout the groups with:

```
cat /etc/group
```

To add a user to a group we use `-g` for the primary group or `-G` for secondary groups:

```
usermod -G GROUP_NAME USER_NAME
```

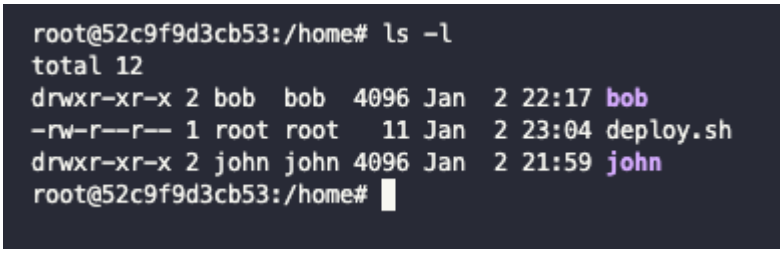
To see the group that a user belongs to we use:

```
groups USER_NAME
```

File permissions

To see the permissions of a file we run:

```
ls -l
```



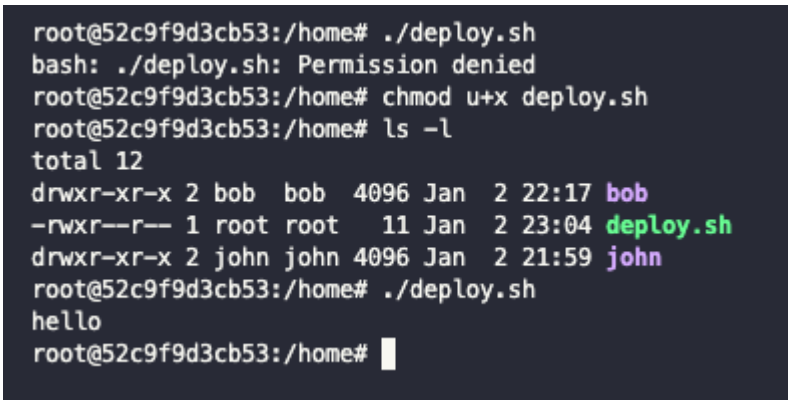
In this output, entries that start with `d` are directories, and entries that start with `-` are files. Then comes 9 letter divided into 3 groups of 3 letters. `r` stands for read, `w` for write, and `x` for execute permissions. A `-` means that we do not have that permission for that file. In the case of a directory, the execute command permission means that we can `cd` into it.

The first group represent the permissions for the user who owns this file. The second represent the permission for the group that owns the file. The third group of letter represent permissions for everyone else.

If we try to execute the file we get a permissions error

Here's where we need to use the `chmod` command (which stands for *change mode*). To it we need to pass `u` for users, `g` for groups, or `o` for others. The we pass `+x` which means to *add the execute permission*. If we wanted to remove the execute permission instead, we would use `u-x`.

```
chmod u+x FILE_NAME
```



We can use combinations of the `chmod` command. For example, `chmod og+x+w-r FILE_NAME` means that, to other users and to the group that owns the file, we want to add the execute permission, add the write permissions, and remove the read permissions. We can also pass multiple files by listing them or by supplying patterns.