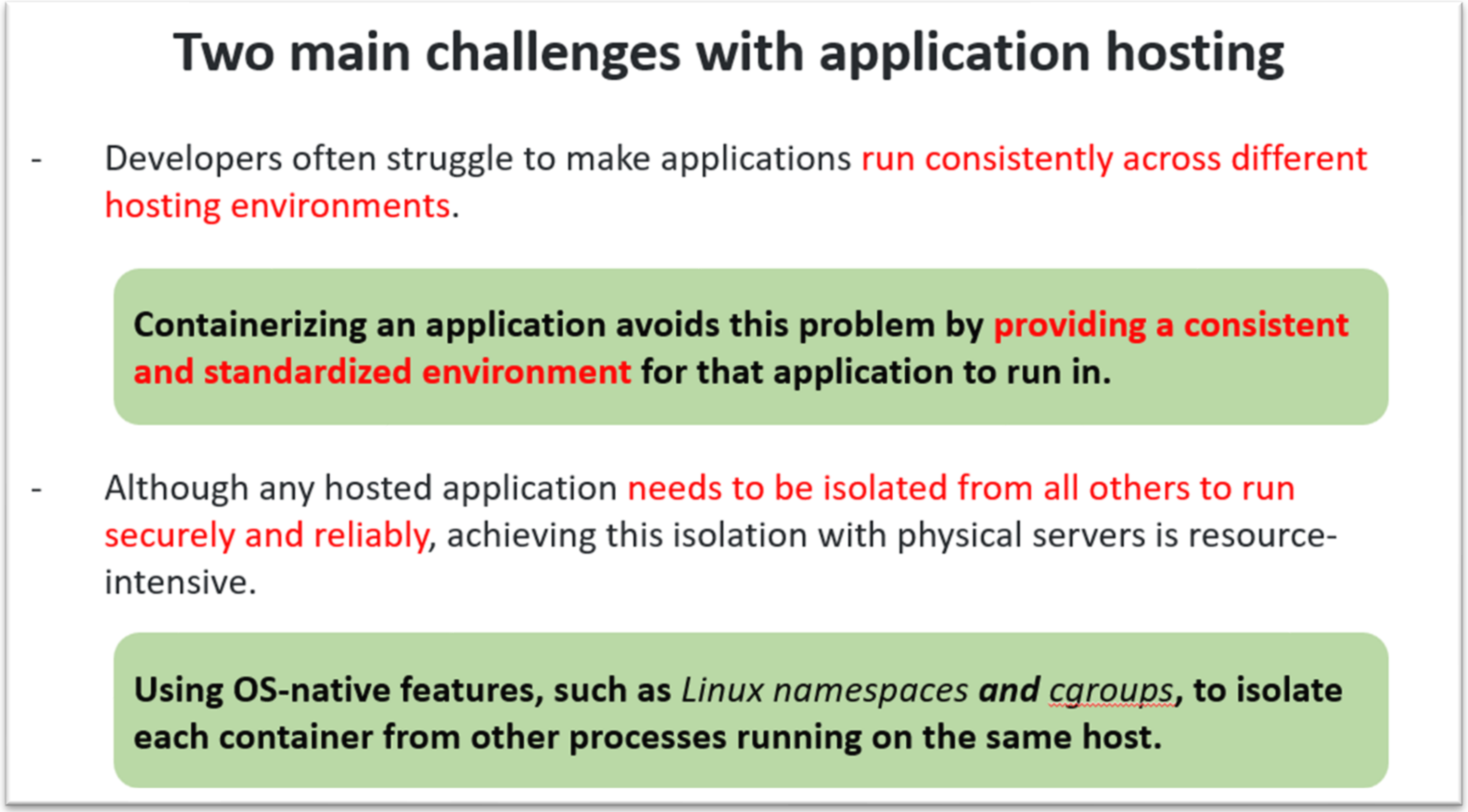
Containerization 2

By Majid Babaei



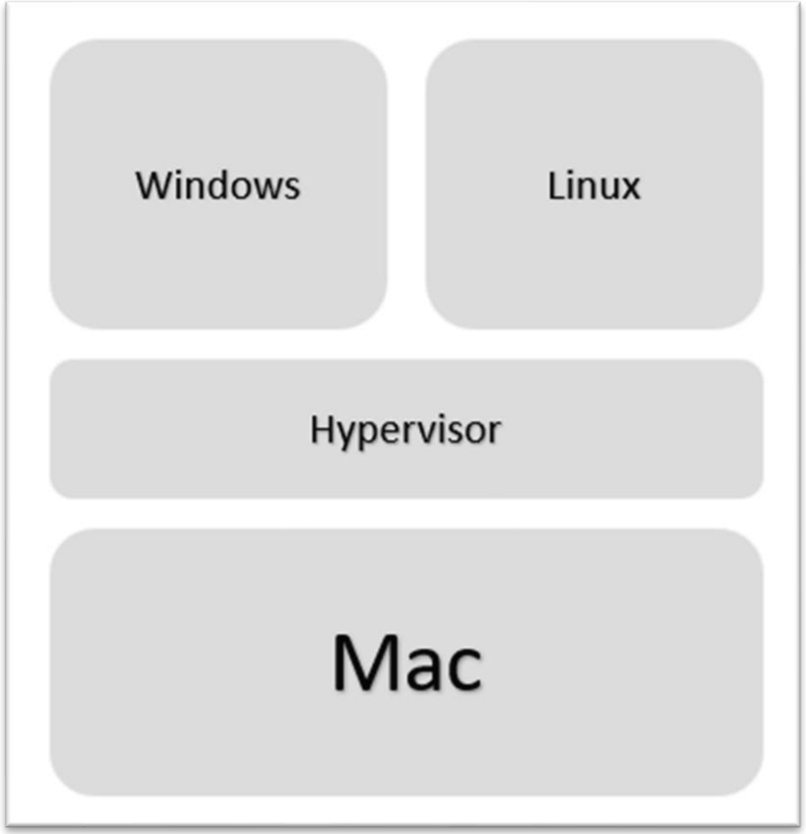
Two main challenges with application hosting

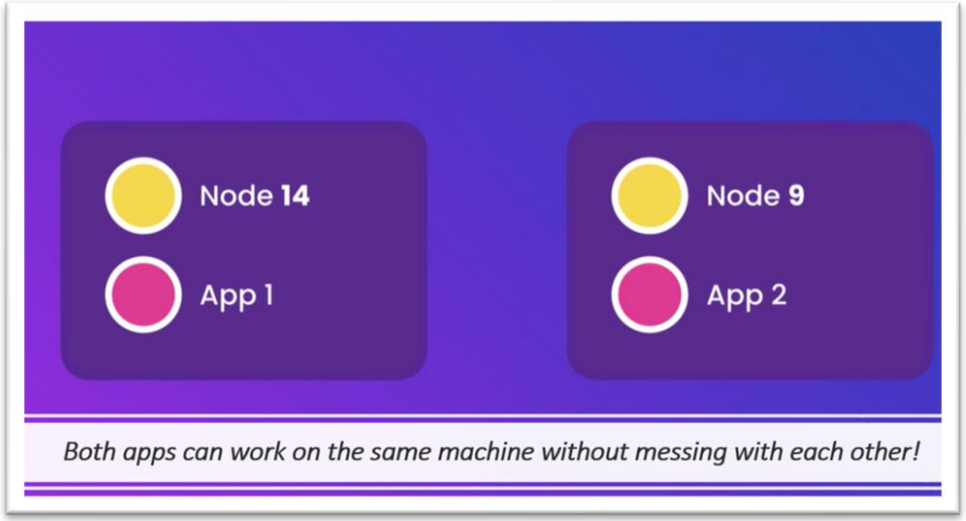
- Developers often struggle to make applications run consistently across different hosting environments.

Containerizing an application avoids this problem by providing a consistent and standardized environment for that application to run in.

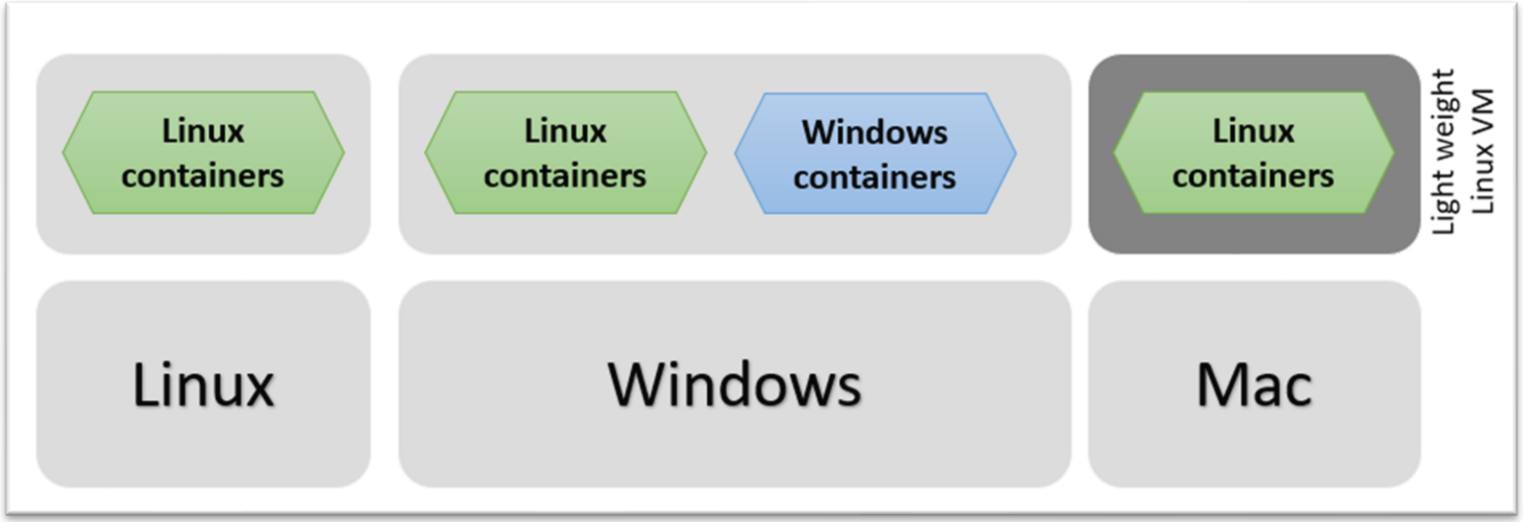
- Although any hosted application needs to be isolated from all others to run securely and reliably, achieving this isolation with physical servers is resource-intensive.

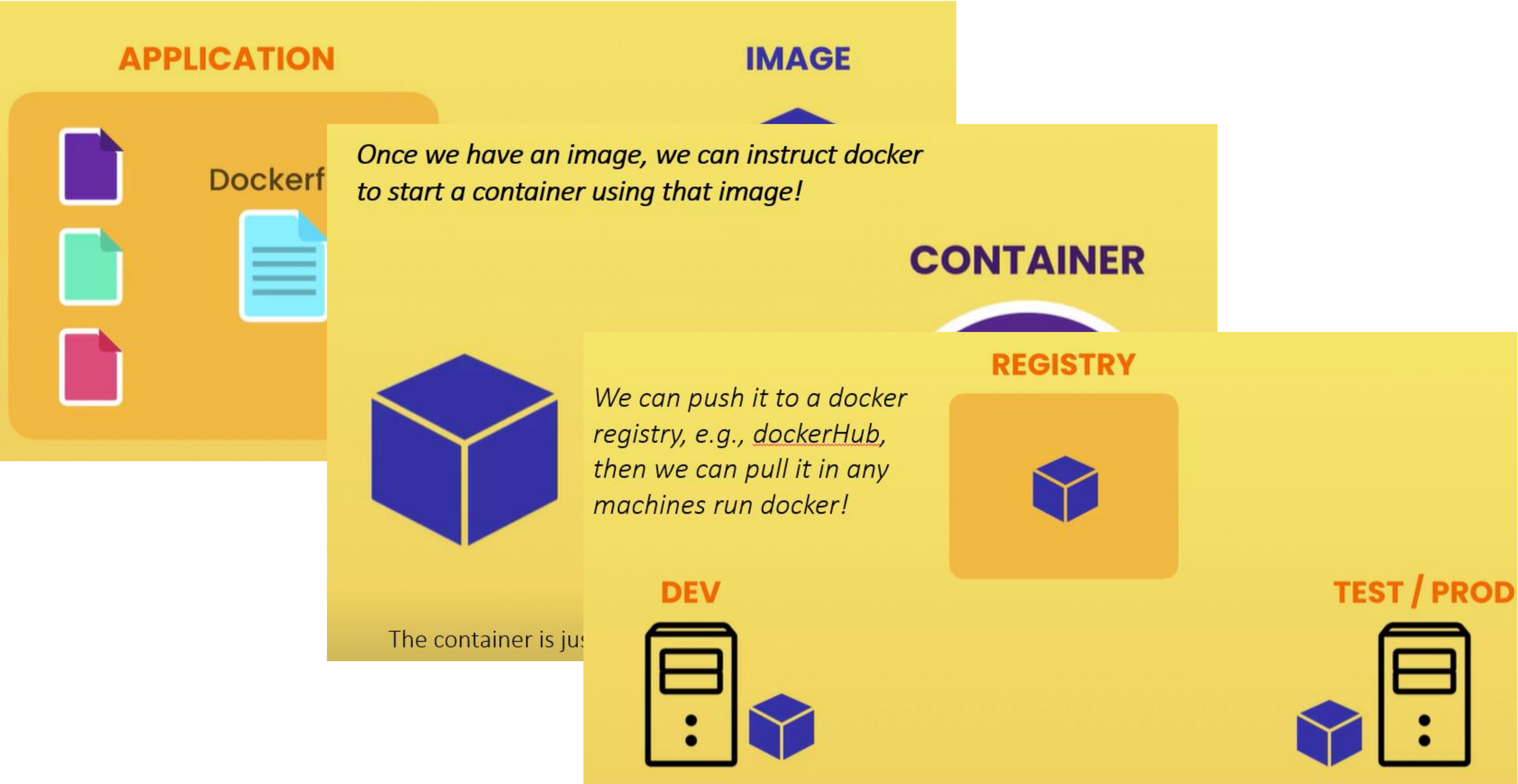
Using OS-native features, such as *Linux namespaces and cgroups*, to isolate each container from other processes running on the same host.





Both apps can work on the same machine without messing with each other!



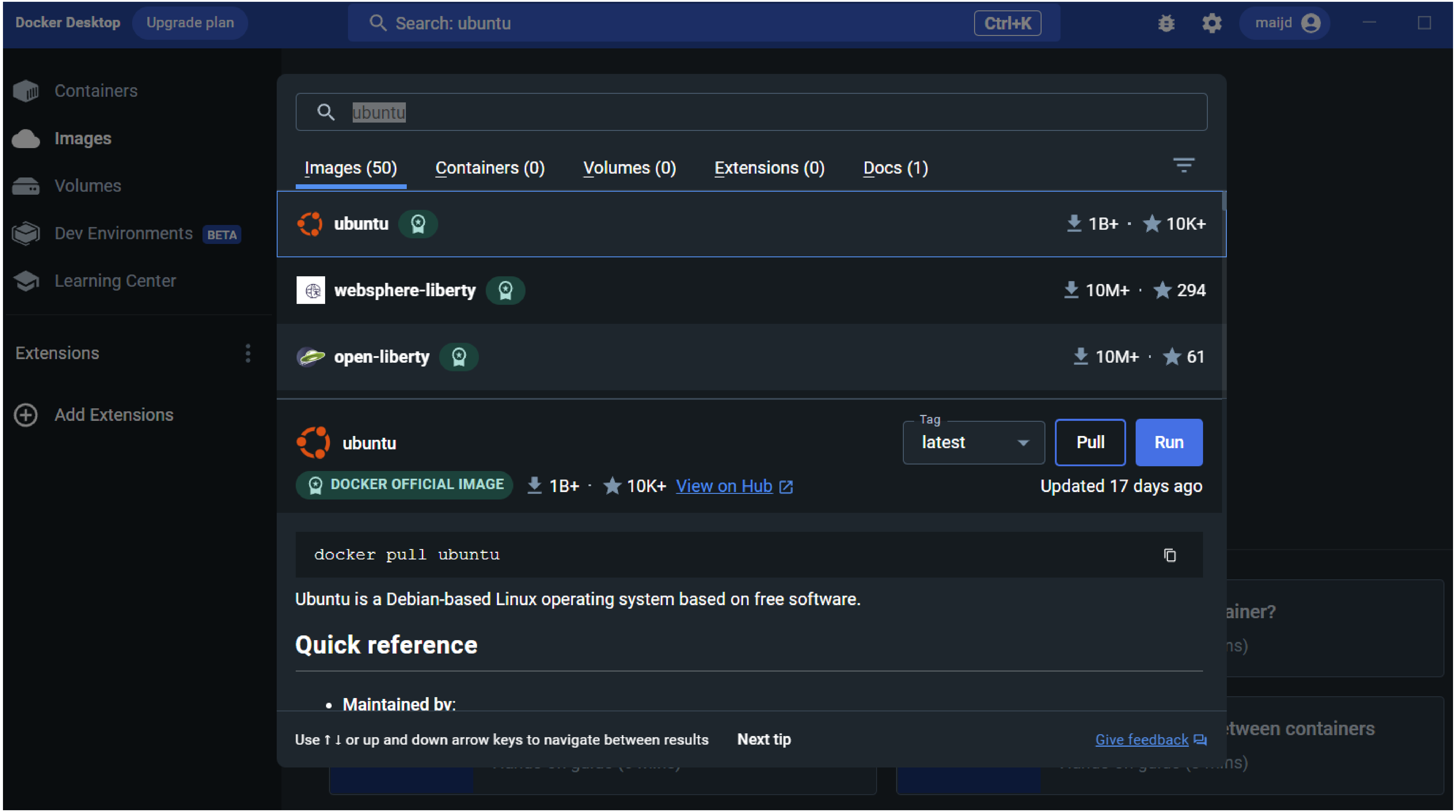


Once we have an image, we can instruct docker to start a container using that image!

The container is jus

We can push it to a docker registry, e.g., dockerHub, then we can pull it in any machines run docker!

Docker is built on basic Linux concepts



Search: ubuntu

Image	Downloads	Stars
ubuntu	1B+	10K+
websphere-liberty	10M+	294
open-liberty	10M+	61

ubuntu

DOCKER OFFICIAL IMAGE · 1B+ · 10K+

Tag: latest

Updated 17 days ago

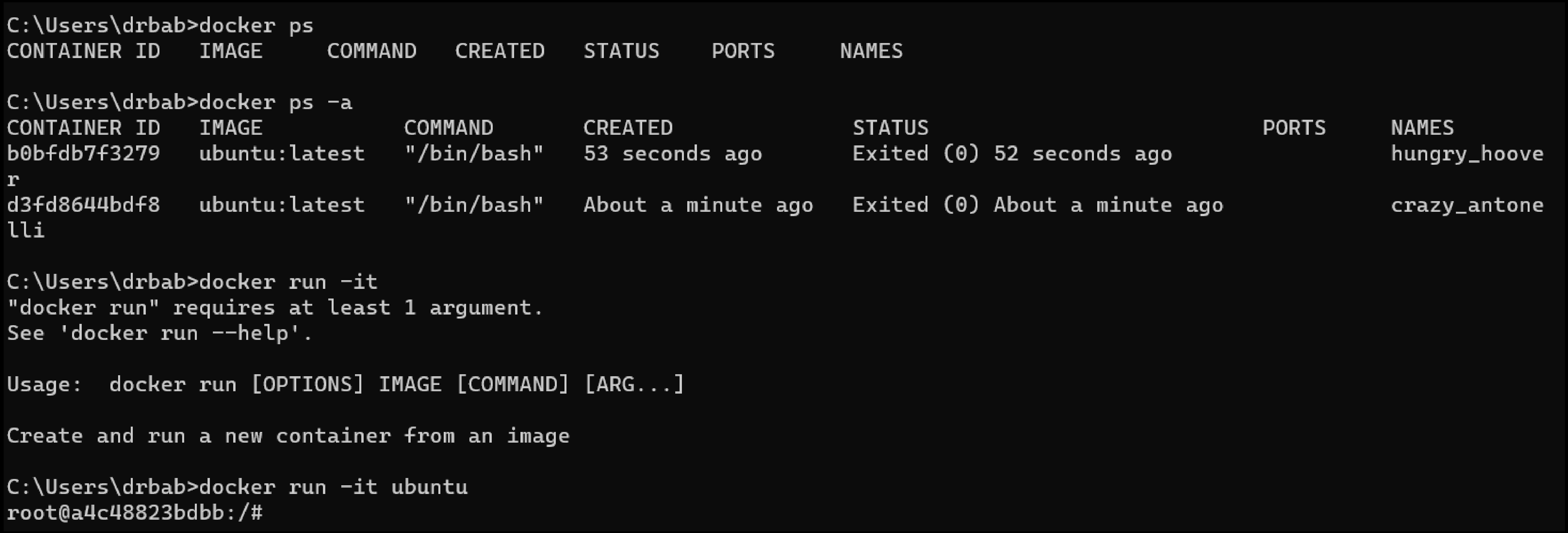
```
docker pull ubuntu
```

Ubuntu is a Debian-based Linux operating system based on free software.

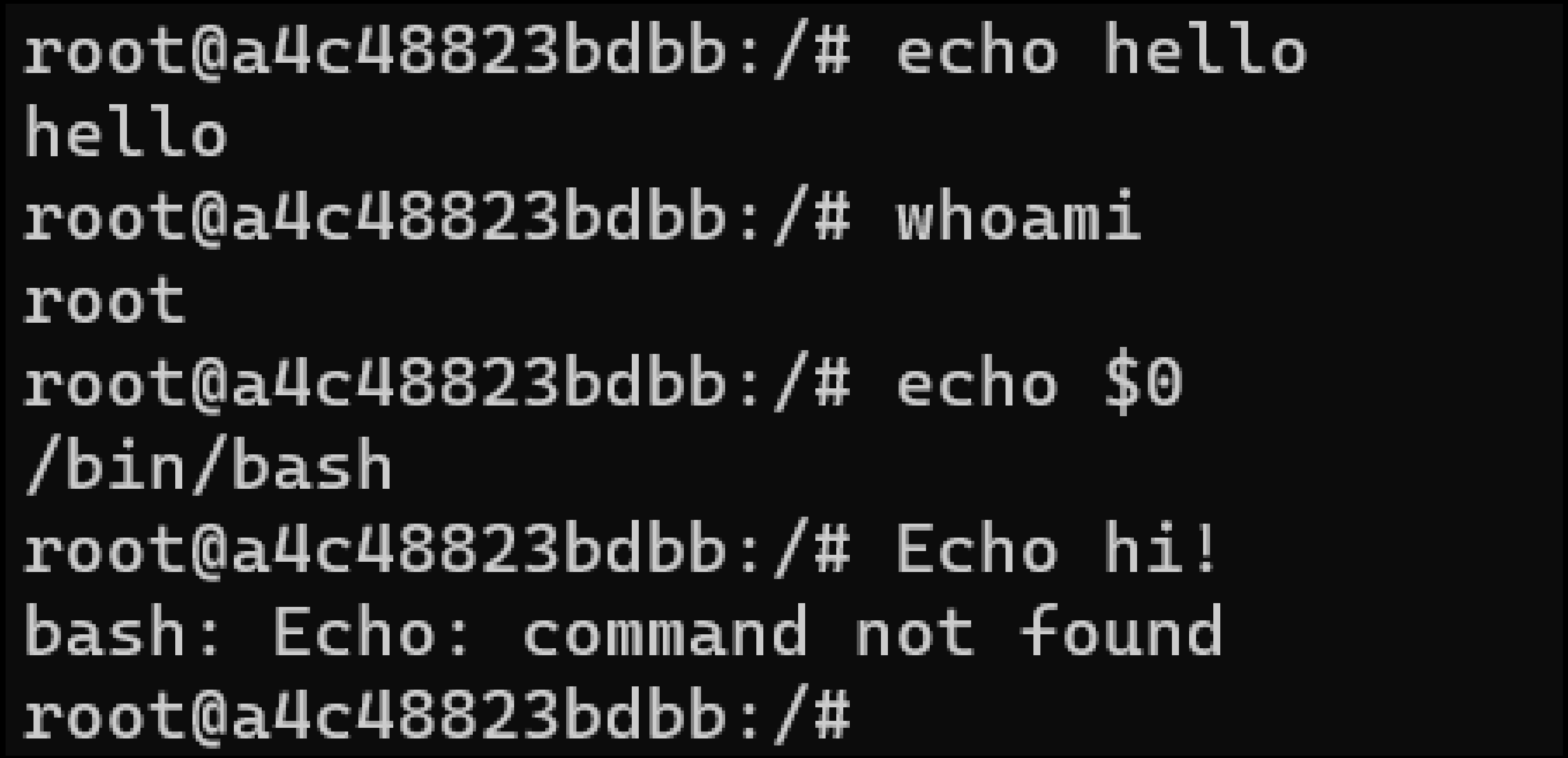
Quick reference

- Maintained by:

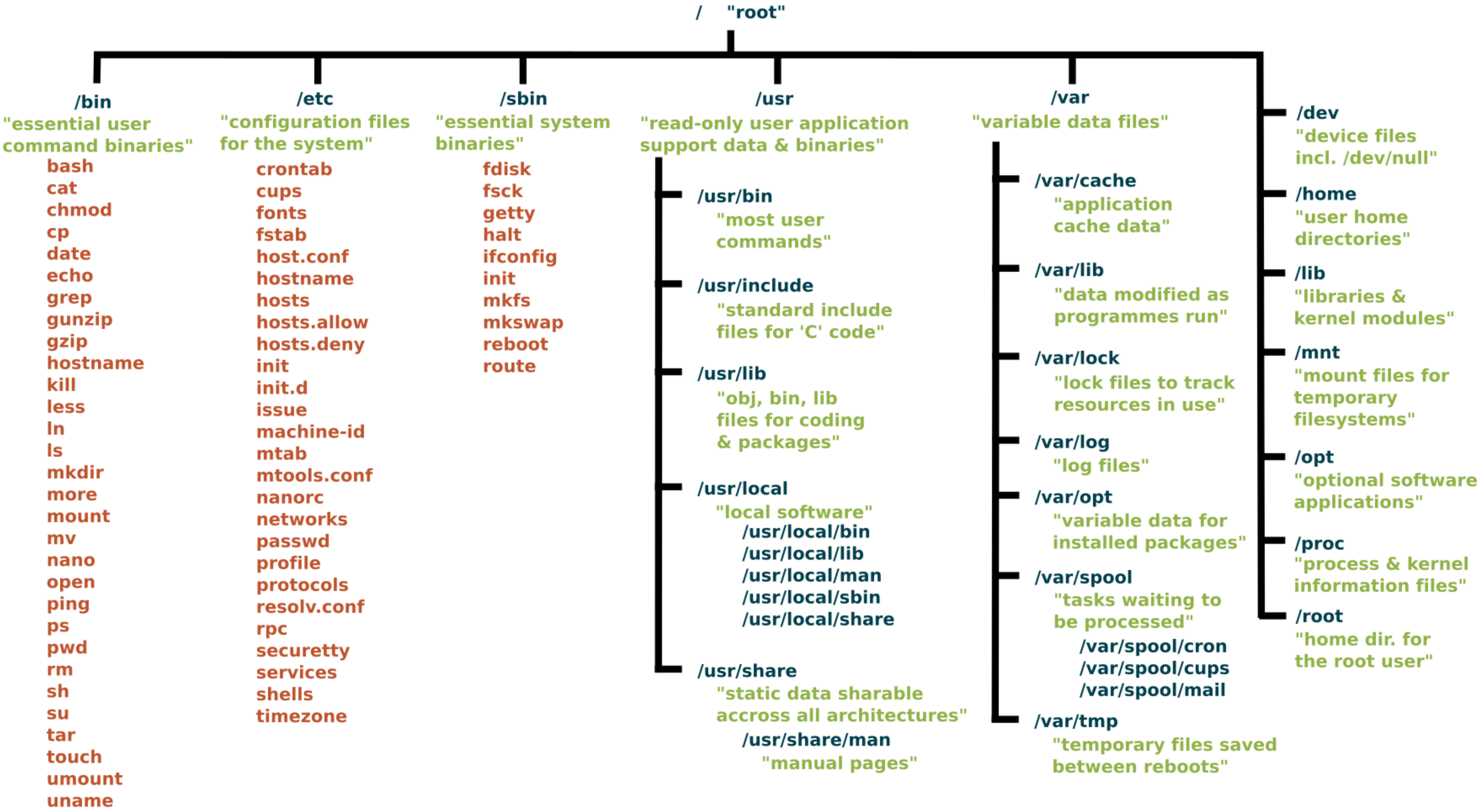
Use ↑ ↓ or up and down arrow keys to navigate between results



```
C:\Users\drbab>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\drbab>docker ps -a
CONTAINER ID   IMAGE           COMMAND       CREATED              STATUS                          PORTS     NAMES
b0bfdb7f3279   ubuntu:latest   "/bin/bash"   53 seconds ago       Exited (0) 52 seconds ago                 hungry_hoove
r
d3fd8644bdf8   ubuntu:latest   "/bin/bash"   About a minute ago   Exited (0) About a minute ago             crazy_antone
lli

C:\Users\drbab>docker run -it
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image

C:\Users\drbab>docker run -it ubuntu
root@a4c48823bdbb:/#
```



```
root@a4c48823bdbb:/# echo hello
hello
root@a4c48823bdbb:/# whoami
root
root@a4c48823bdbb:/# echo $0
/bin/bash
root@a4c48823bdbb:/# Echo hi!
bash: Echo: command not found
root@a4c48823bdbb:/#
```



/ "root"

/bin
"essential user command binaries"

bash
cat
chmod
cp
date
echo
grep
gunzip
gzip
hostname
kill
less
ln
ls
mkdir
more
mount
mv
nano
open
ping
ps
pwd
rm
sh
su
tar
touch
umount
uname

/etc
"configuration files for the system"

crontab
cups
fonts
fstab
host.conf
hostname
hosts
hosts.allow
hosts.deny
init
init.d
issue
machine-id
mtab
mtools.conf
nanorc
networks
passwd
profile
protocols
resolv.conf
rpc
securetty
services
shells
timezone

/sbin
"essential system binaries"

fdisk
fsck
getty
halt
ifconfig
init
mkfs
mkswap
reboot
route

/usr
"read-only user application support data & binaries"

/usr/bin
"most user commands"

/usr/include
"standard include files for 'C' code"

/usr/lib
"obj, bin, lib files for coding & packages"

/usr/local
"local software"

/usr/local/bin
/usr/local/lib
/usr/local/man
/usr/local/sbin
/usr/local/share

/usr/share
"static data sharable accross all architectures"

/usr/share/man
"manual pages"

/var
"variable data files"

/var/cache
"application cache data"

/var/lib
"data modified as programmes run"

/var/lock
"lock files to track resources in use"

/var/log
"log files"

/var/opt
"variable data for installed packages"

/var/spool
"tasks waiting to be processed"

/var/spool/cron
/var/spool/cups
/var/spool/mail

/var/tmp
"temporary files saved between reboots"

/dev
"device files incl. /dev/null"

/home
"user home directories"

/lib
"libraries & kernel modules"

/mnt
"mount files for temporary filesystems"

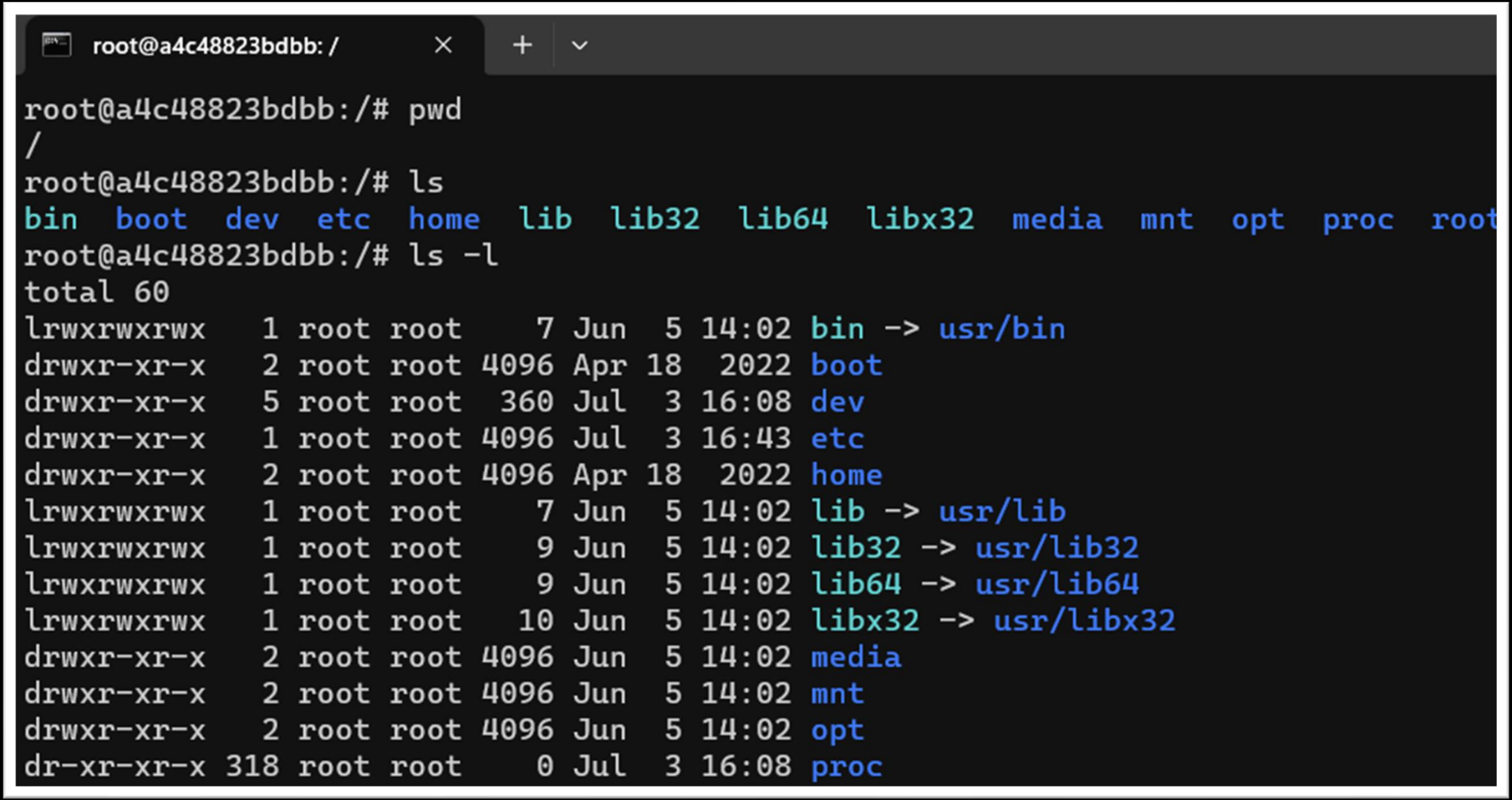
/opt
"optional software applications"

/proc
"process & kernel information files"

/root
"home dir. for the root user"

Navigating the file systems

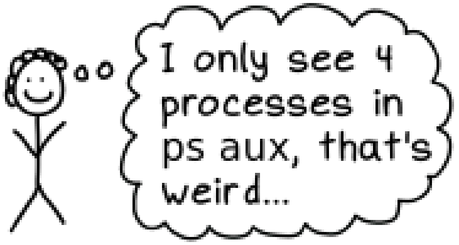
To see where we are in the file system:
We can use *pwd* (Print Working Directory)



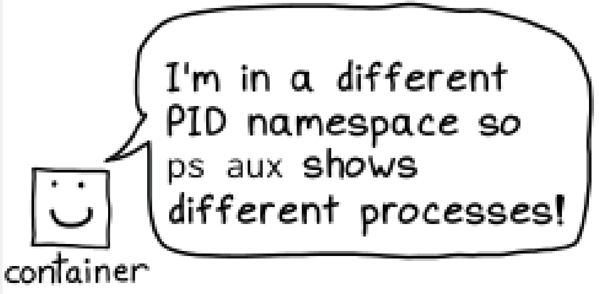
```
root@a4c48823bdbb:/# pwd
/
root@a4c48823bdbb:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root
root@a4c48823bdbb:/# ls -l
total 60
lrwxrwxrwx   1 root root    7 Jun  5 14:02 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 18  2022 boot
drwxr-xr-x   5 root root  360 Jul  3 16:08 dev
drwxr-xr-x   1 root root 4096 Jul  3 16:43 etc
drwxr-xr-x   2 root root 4096 Apr 18  2022 home
lrwxrwxrwx   1 root root    7 Jun  5 14:02 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Jun  5 14:02 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Jun  5 14:02 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Jun  5 14:02 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Jun  5 14:02 media
drwxr-xr-x   2 root root 4096 Jun  5 14:02 mnt
drwxr-xr-x   2 root root 4096 Jun  5 14:02 opt
dr-xr-xr-x 318 root root    0 Jul  3 16:08 proc
```

namespaces

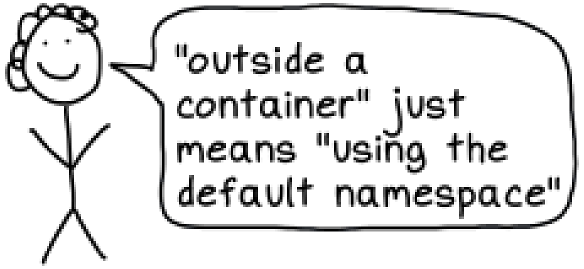
inside a container, things look different



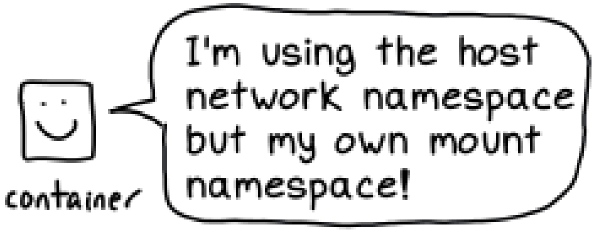
why things look different: namespaces


I'm in a different PID namespace so ps aux shows different processes!
container

there's a default ("host" namespace)


"outside a container" just means "using the default namespace"

processes can have any combination of namespaces


I'm using the host network namespace but my own mount namespace!
container

every process has 7 namespaces

```
              PID
$ lsns -p 273
          NS TYPE
4026531835    cgroup
4026531836    pid
4026531837    user
4026531838    uts
4026531839    ipc
4026531840    mnt
4026532009    net
     namespace ID
```

you can also see a process's namespace with:

```
$ ls -l /proc/273/ns
```

What Is a Namespace?

- Namespaces are a Linux kernel feature that isolates various aspects of a process.
- They provide a process with its own isolated view of the system, such as its own file system, network, hostname, and more.
- They allow us to create isolated environments for processes so that they can’t access or affect other processes or the host system.
- Containers make extensive use of Linux namespaces to be able to group processes and provide resource isolation for them.

The main advantages in the use of namespaces

- **Isolation of resources:** One troublesome process won’t be taking down the whole host, it’ll only affect those processes belonging to a particular namespace.

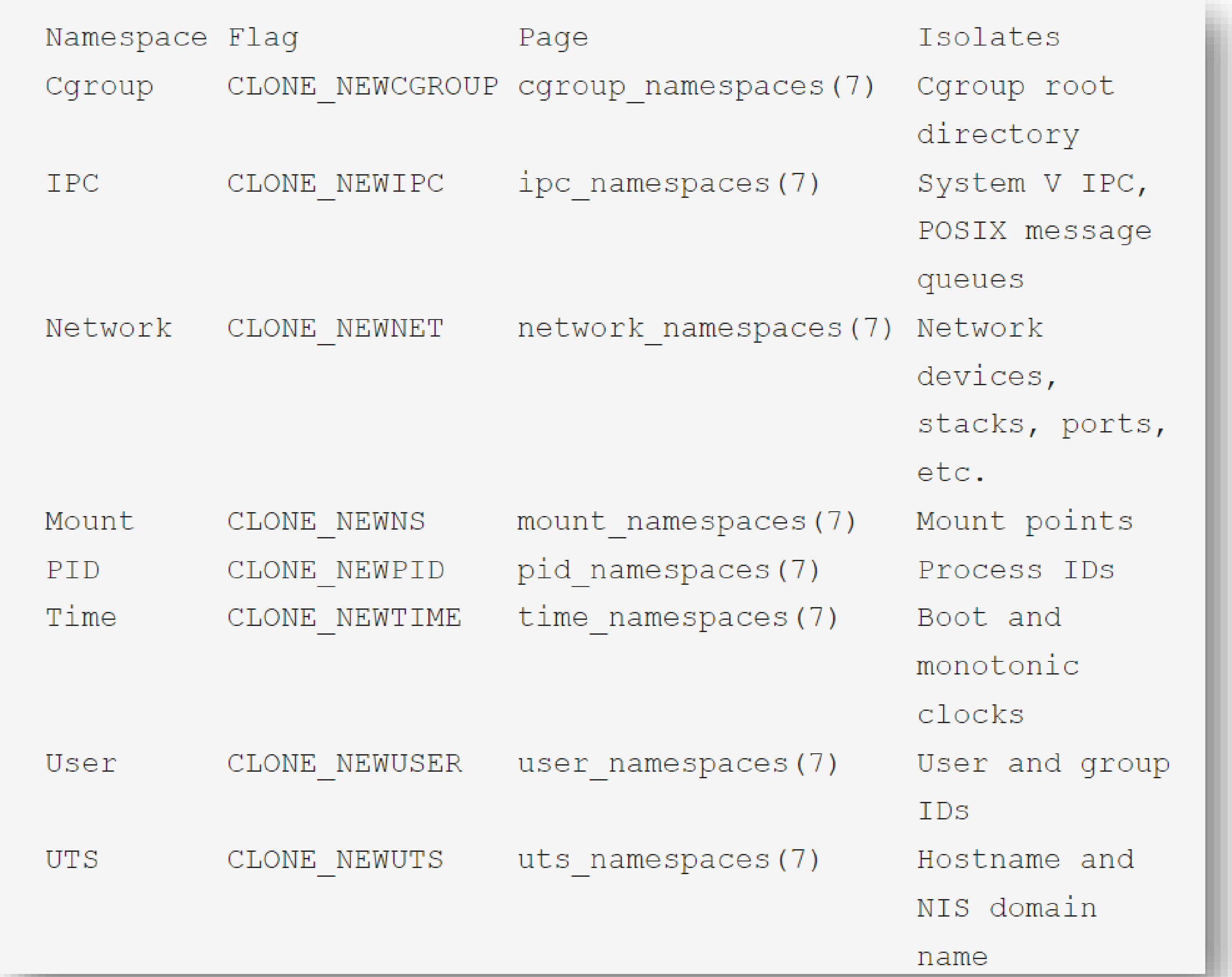
- **Security**: The other advantage is that a security flaw in the process or processes running under a given namespace, won’t give access to the attacker to the whole system.

Whatever the attacker could do, will always be contained within the boundaries of that namespace! This is why it’s also very important to avoid running our processes using privileged users whenever possible.

Types of Namespaces

- Each type of namespace is different, and it provides isolation for different resources in our system.

- If we check the namespaces in the Linux manual pages, we can see a list of namespace types:

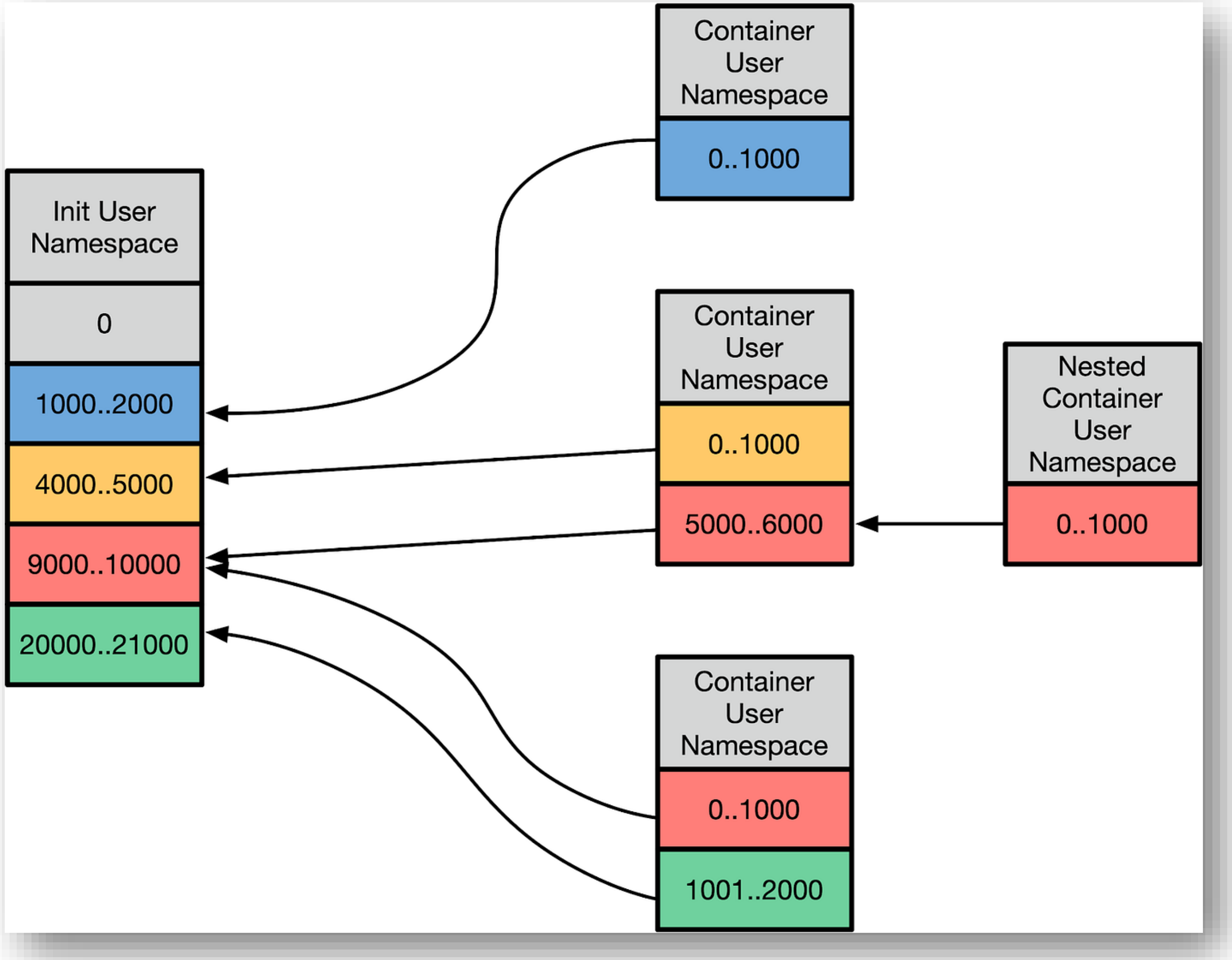


Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	mount_namespaces(7)	Mount points
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic clocks
User	CLONE_NEWUSER	user_namespaces(7)	User and group IDs
UTS	CLONE_NEWUTS	uts_namespaces(7)	Hostname and NIS domain name

User Namespace

- User namespaces is a Linux feature that allows to map users in the container to different users in the host.

- containers can run as root and be mapped to a non-root user on the host. Inside the container the process will think it is running as root (and therefore tools like apt, yum, etc. work fine), while in reality the process doesn't have privileges on the host.



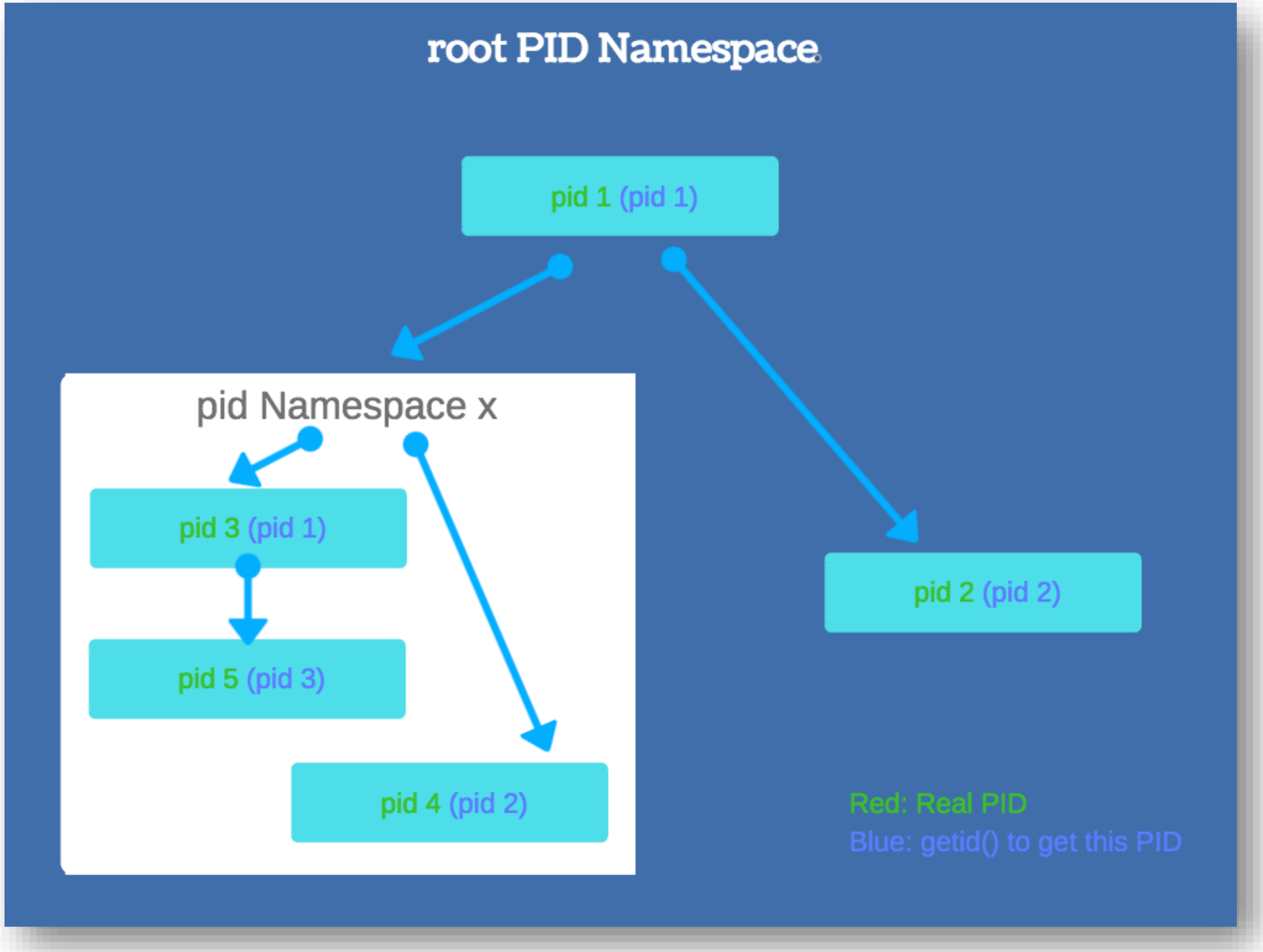
PID Namespace

- PID namespaces enable processes in different containers to have the same PID (process identifier).

- This means each container can have its own init (PID1) process that manages various system initialization tasks, as well as the container life cycle.

- Each container has its unique /proc directory.

- The container is only aware of its native processes

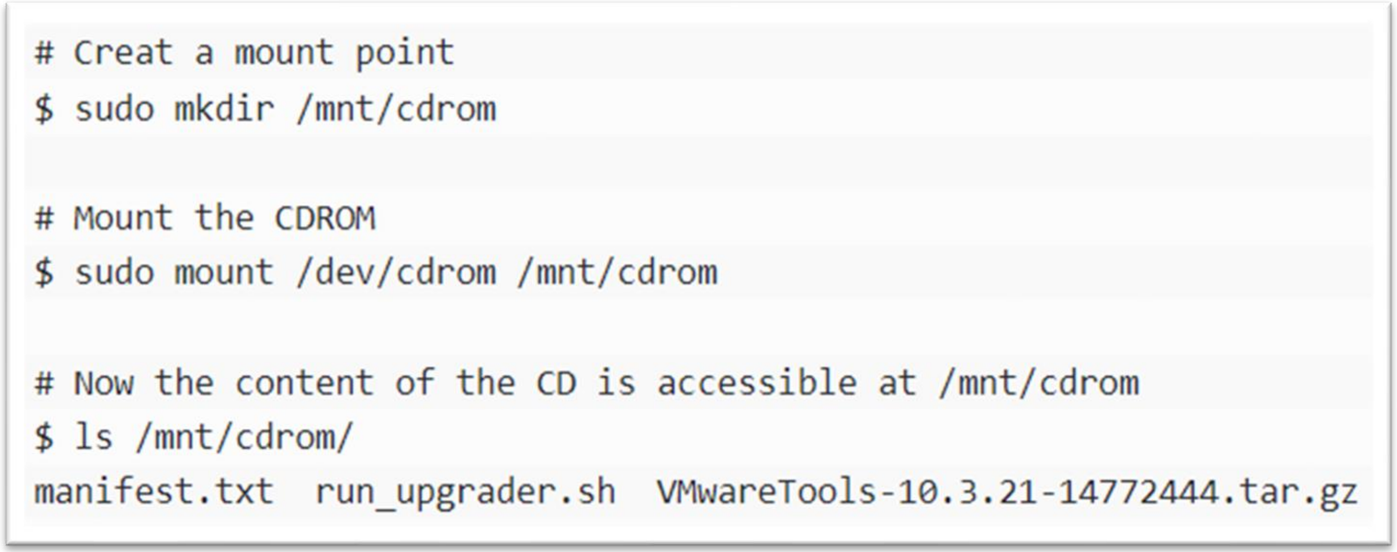


Mount Namespace

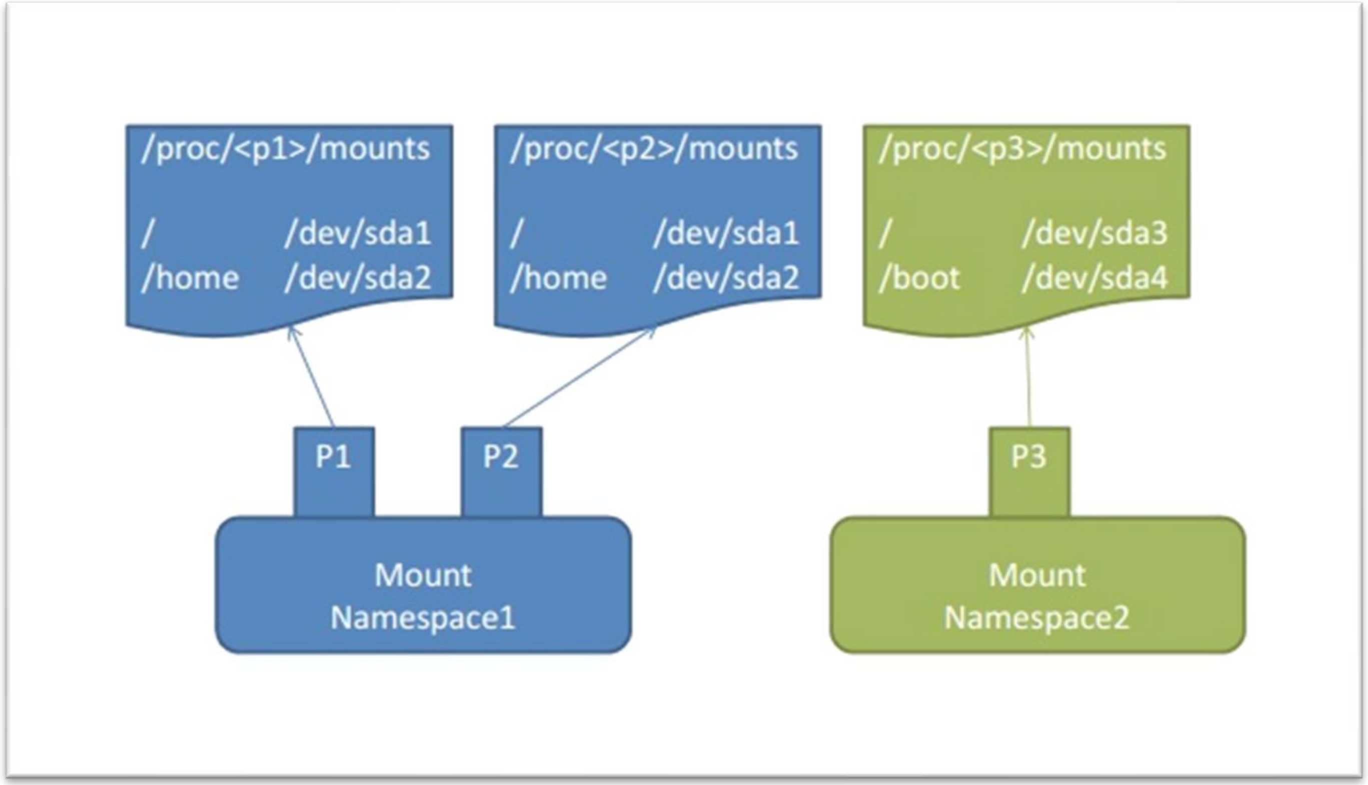
- *mount* is the operation of attaching the filesystem on a device to the host’s filesystem tree.

- mount namespaces isolate the list of mount points seen by the processes in a namespace.

- Processes in different namespaces have different views of mount points in the system.



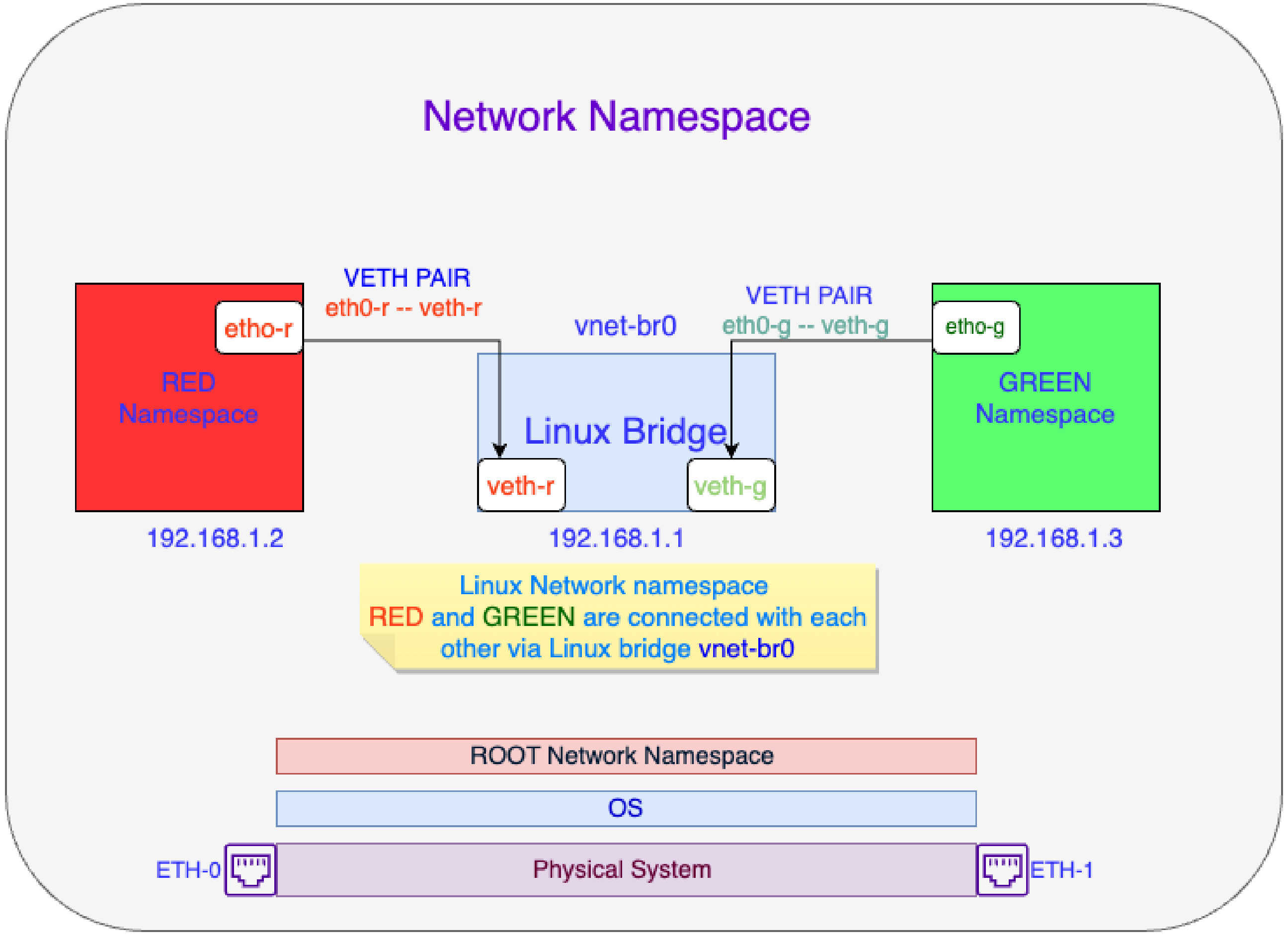
```
# Creat a mount point
$ sudo mkdir /mnt/cdrom

# Mount the CDROM
$ sudo mount /dev/cdrom /mnt/cdrom

# Now the content of the CD is accessible at /mnt/cdrom
$ ls /mnt/cdrom/
manifest.txt  run_upgrader.sh  VMwareTools-10.3.21-14772444.tar.gz
```



Network Namespace

- Each network namespace will have its own independent network configuration, interfaces, IP addresses, routing tables, and firewall rules.

- When a new network namespace is created, it starts with a completely isolated network stack, with no network interfaces except for the loopback interface.



Linux Network namespace
RED and GREEN are connected with each other via Linux bridge vnet-br0

Cgroup Namespace

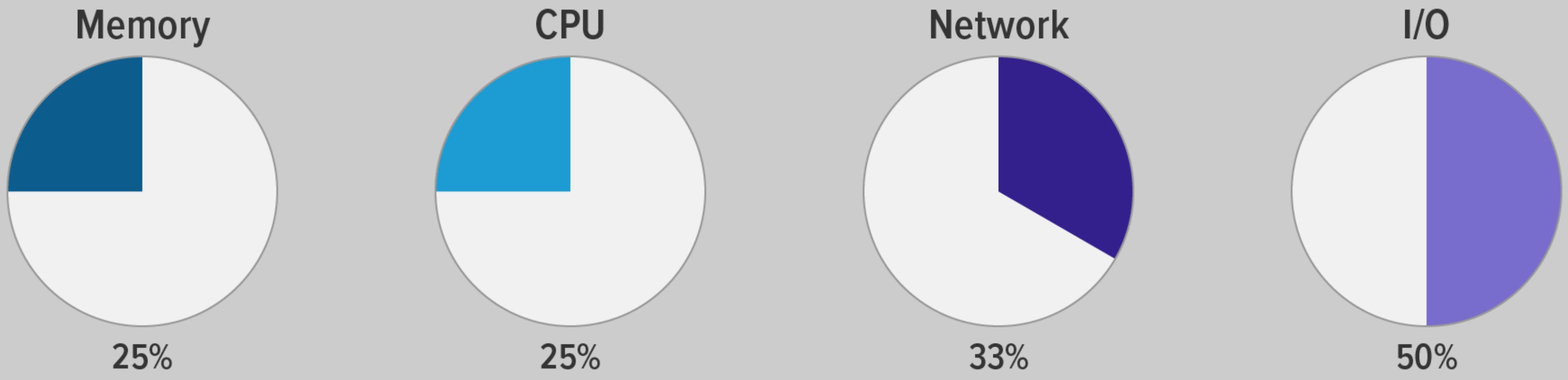
- Cgroups, short for control groups, are a kernel feature that allows organizing processes into hierarchical groups to manage and enforce limits on system resources.

- Cgroup namespaces virtualize the view of the cgroup hierarchy, so that processes running within a cgroup namespace have a different view of the hierarchy compared to processes running in the host or other namespaces.

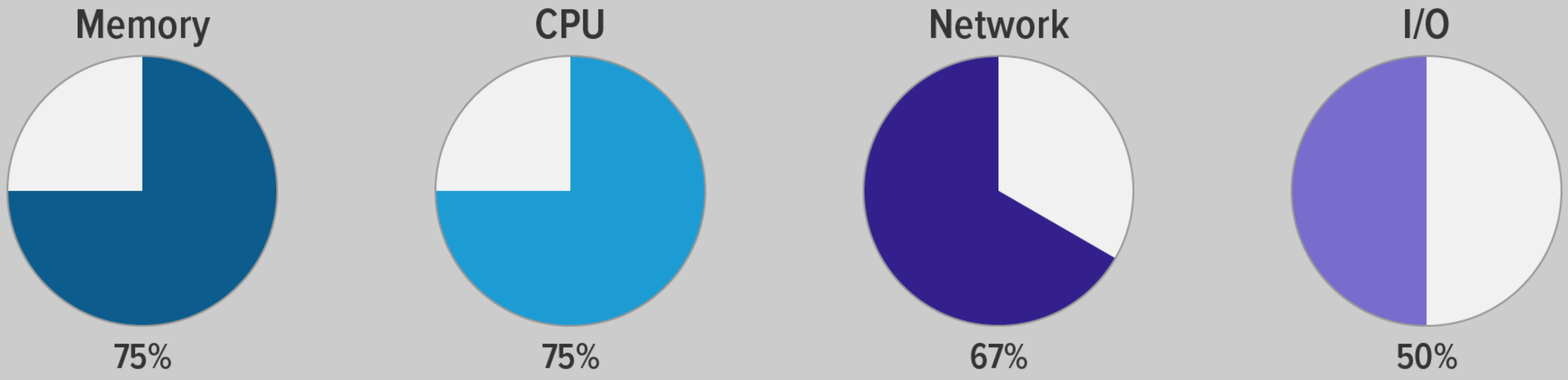
- **Resource limits** – You can configure a cgroup to limit how much of a particular resource (memory or CPU, for example) a process can use.

- **Prioritization** – You can control how much of a resource (CPU, disk, or network) a process can use compared to processes in another cgroup when there is resource contention.

cgroup-1

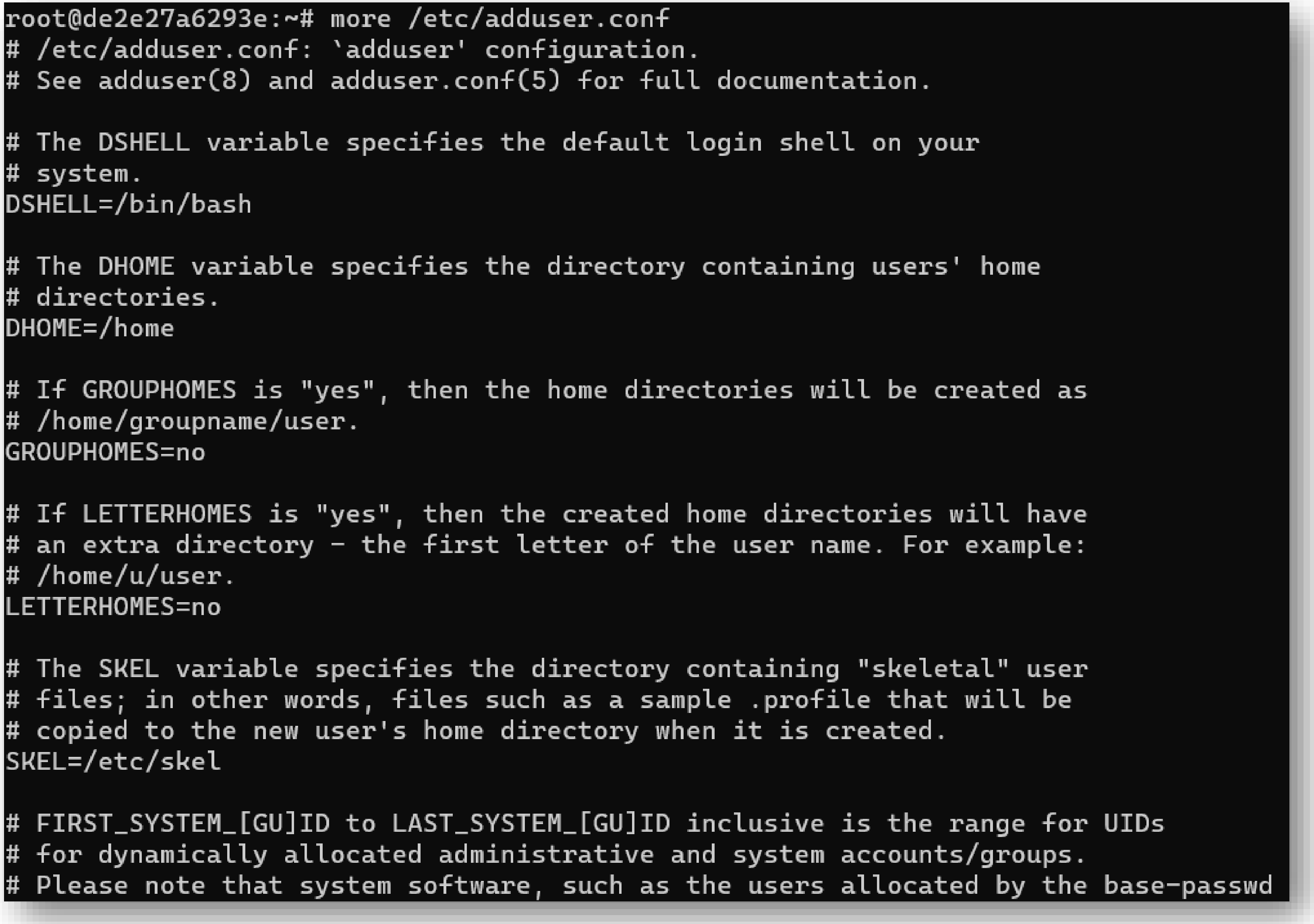


System Remainder After cgroup Allocation



Linux System Commands

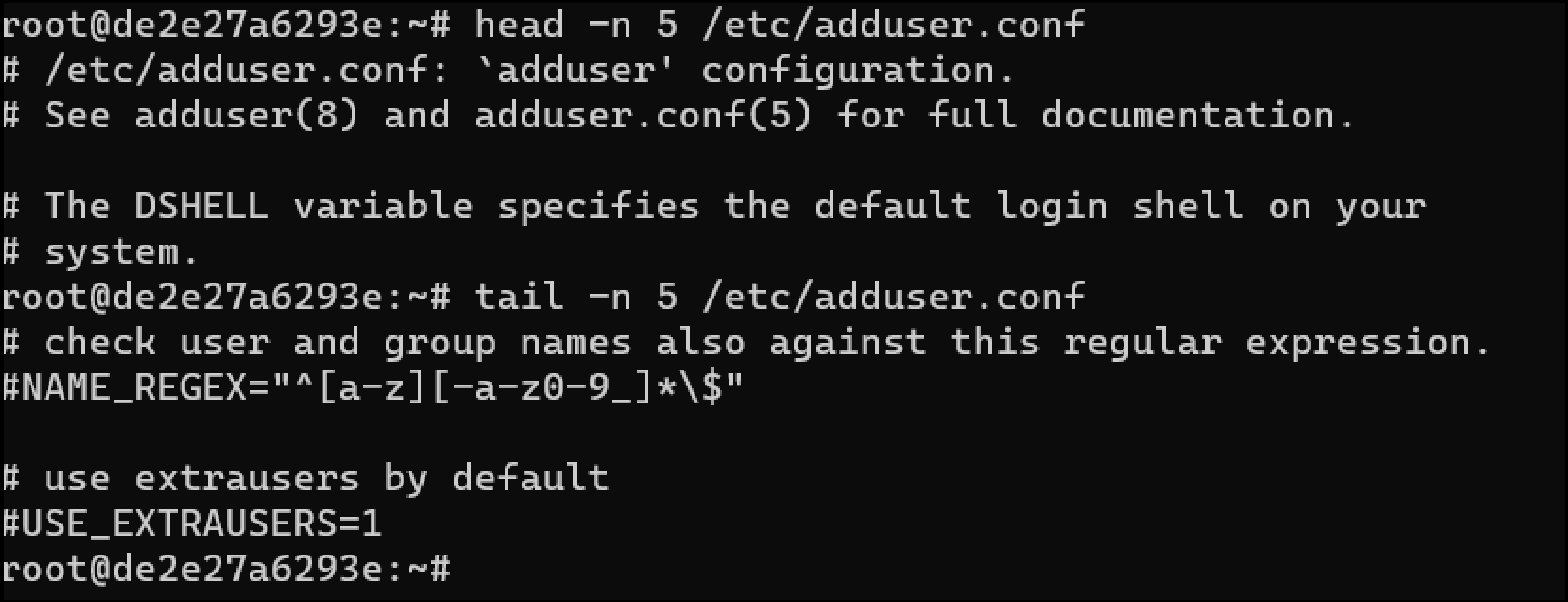
For long files we can use *more* instead of *cat*



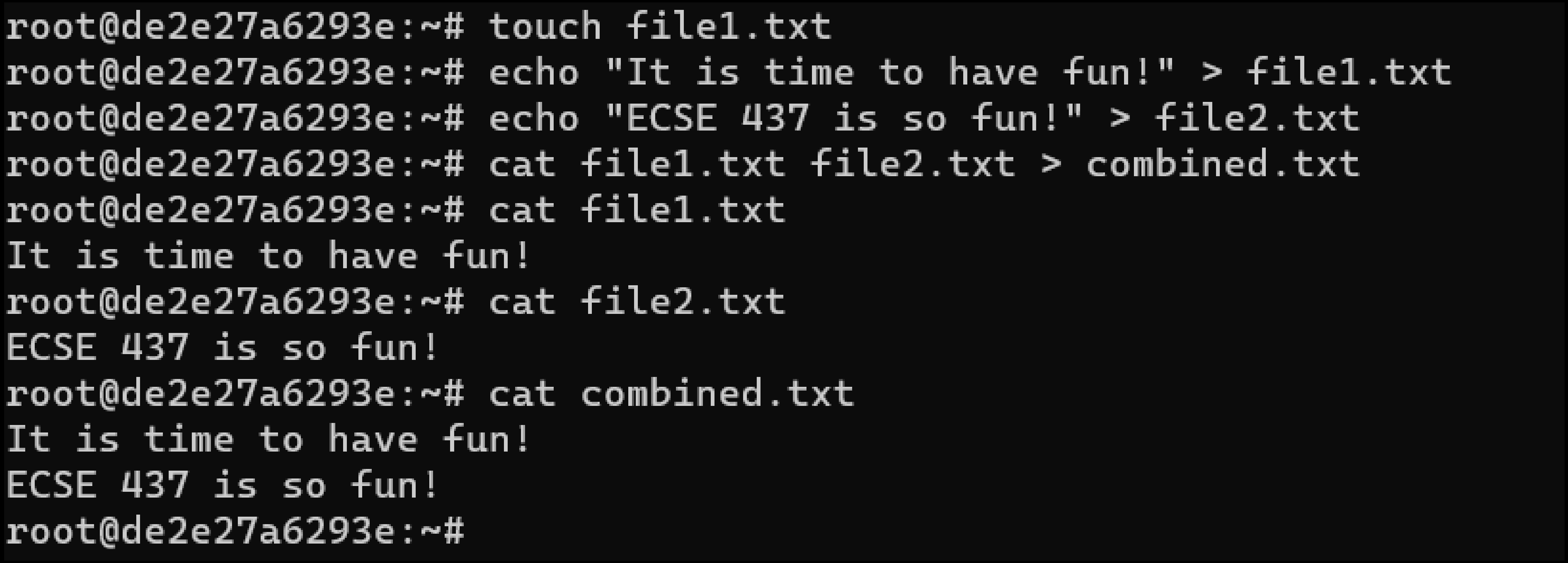
```
root@de2e27a6293e:~# more /etc/adduser.conf
# /etc/adduser.conf: `adduser' configuration.
# See adduser(8) and adduser.conf(5) for full documentation.

# The DSHELL variable specifies the default login shell on your
# system.
DSHELL=/bin/bash

# The DHOME variable specifies the directory containing users' home
# directories.
DHOME=/home

# If GROUPHOMES is "yes", then the home directories will be created as
# /home/groupname/user.
GROUPHOMES=no

# If LETTERHOMES is "yes", then the created home directories will have
# an extra directory - the first letter of the user name. For example:
# /home/u/user.
LETTERHOMES=no

# The SKEL variable specifies the directory containing "skeletal" user
# files; in other words, files such as a sample .profile that will be
# copied to the new user's home directory when it is created.
SKEL=/etc/skel

# FIRST_SYSTEM_[GU]ID to LAST_SYSTEM_[GU]ID inclusive is the range for UIDs
# for dynamically allocated administrative and system accounts/groups.
# Please note that system software, such as the users allocated by the base-passwd
```

With *more* you can only go down!
To go up you can use *less*

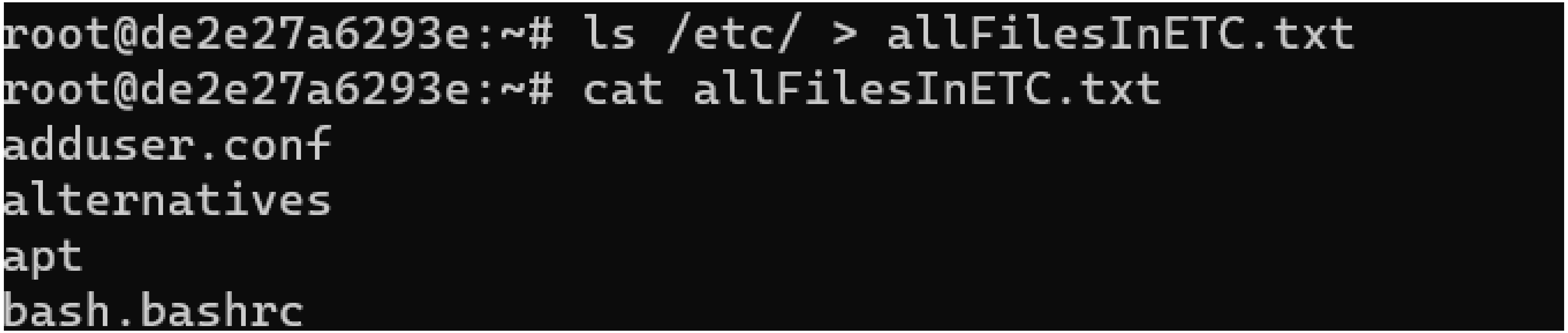
***Make sure to install it first!**



```
root@de2e27a6293e:~# head -n 5 /etc/adduser.conf
# /etc/adduser.conf: `adduser' configuration.
# See adduser(8) and adduser.conf(5) for full documentation.

# The DSHELL variable specifies the default login shell on your
# system.
root@de2e27a6293e:~# tail -n 5 /etc/adduser.conf
# check user and group names also against this regular expression.
#NAME_REGEX="^[a-z][-a-z0-9_]*\$"

# use extrausers by default
#USE_EXTRAUSERS=1
root@de2e27a6293e:~#
```



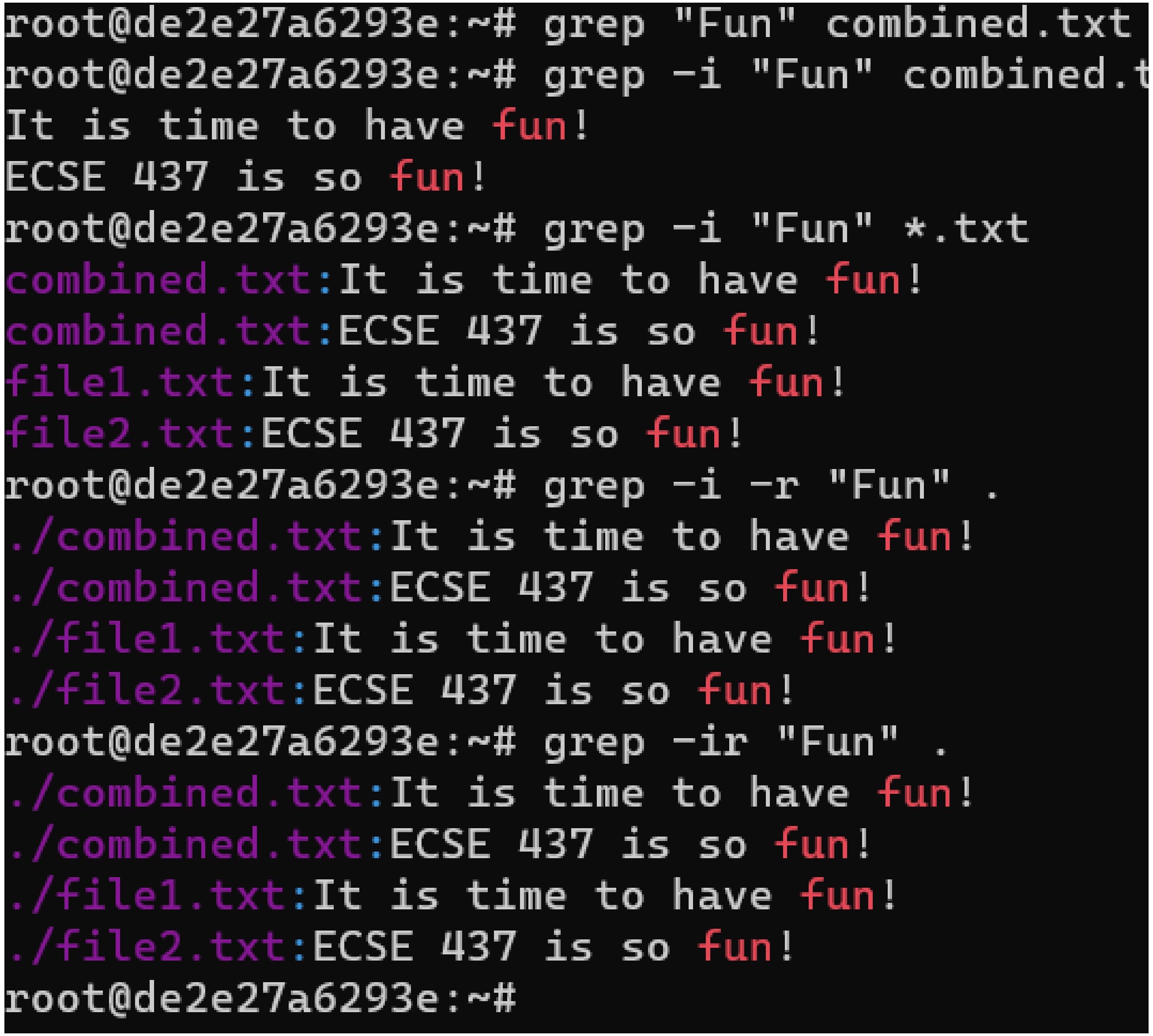
```
root@de2e27a6293e:~# touch file1.txt
root@de2e27a6293e:~# echo "It is time to have fun!" > file1.txt
root@de2e27a6293e:~# echo "ECSE 437 is so fun!" > file2.txt
root@de2e27a6293e:~# cat file1.txt file2.txt > combined.txt
root@de2e27a6293e:~# cat file1.txt
It is time to have fun!
root@de2e27a6293e:~# cat file2.txt
ECSE 437 is so fun!
root@de2e27a6293e:~# cat combined.txt
It is time to have fun!
ECSE 437 is so fun!
root@de2e27a6293e:~#
```



```
root@de2e27a6293e:~# ls /etc/ > allFilesInETC.txt
root@de2e27a6293e:~# cat allFilesInETC.txt
adduser.conf
alternatives
apt
bash.bashrc
```

Get the long list of files in etc directory and write the output to a file

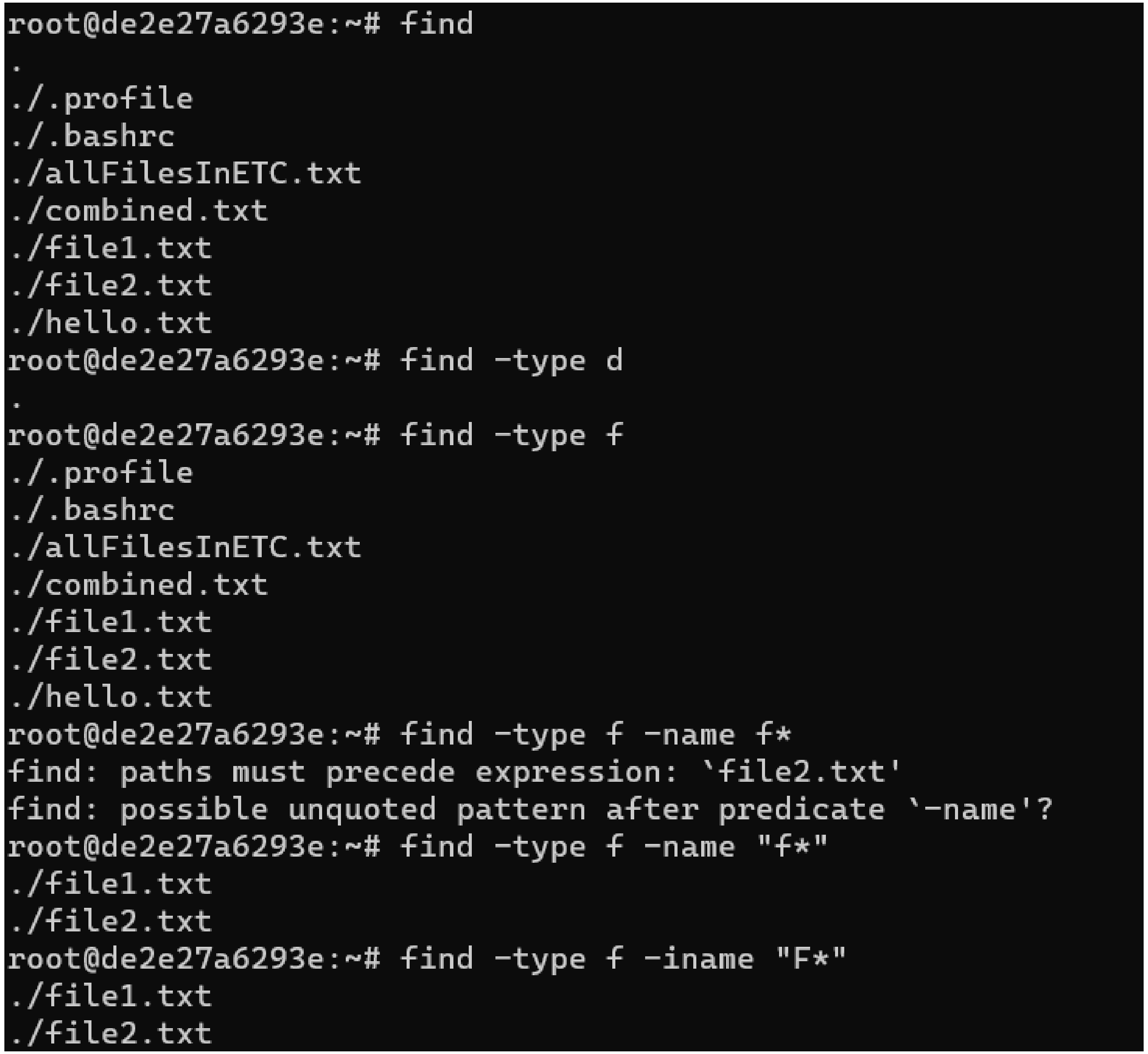
Search in a file



```
root@de2e27a6293e:~# grep "Fun" combined.txt
root@de2e27a6293e:~# grep -i "Fun" combined.t
It is time to have fun!
ECSE 437 is so fun!
root@de2e27a6293e:~# grep -i "Fun" *.txt
combined.txt:It is time to have fun!
combined.txt:ECSE 437 is so fun!
file1.txt:It is time to have fun!
file2.txt:ECSE 437 is so fun!
root@de2e27a6293e:~# grep -i -r "Fun" .
./combined.txt:It is time to have fun!
./combined.txt:ECSE 437 is so fun!
./file1.txt:It is time to have fun!
./file2.txt:ECSE 437 is so fun!
root@de2e27a6293e:~# grep -ir "Fun" .
./combined.txt:It is time to have fun!
./combined.txt:ECSE 437 is so fun!
./file1.txt:It is time to have fun!
./file2.txt:ECSE 437 is so fun!
root@de2e27a6293e:~#
```

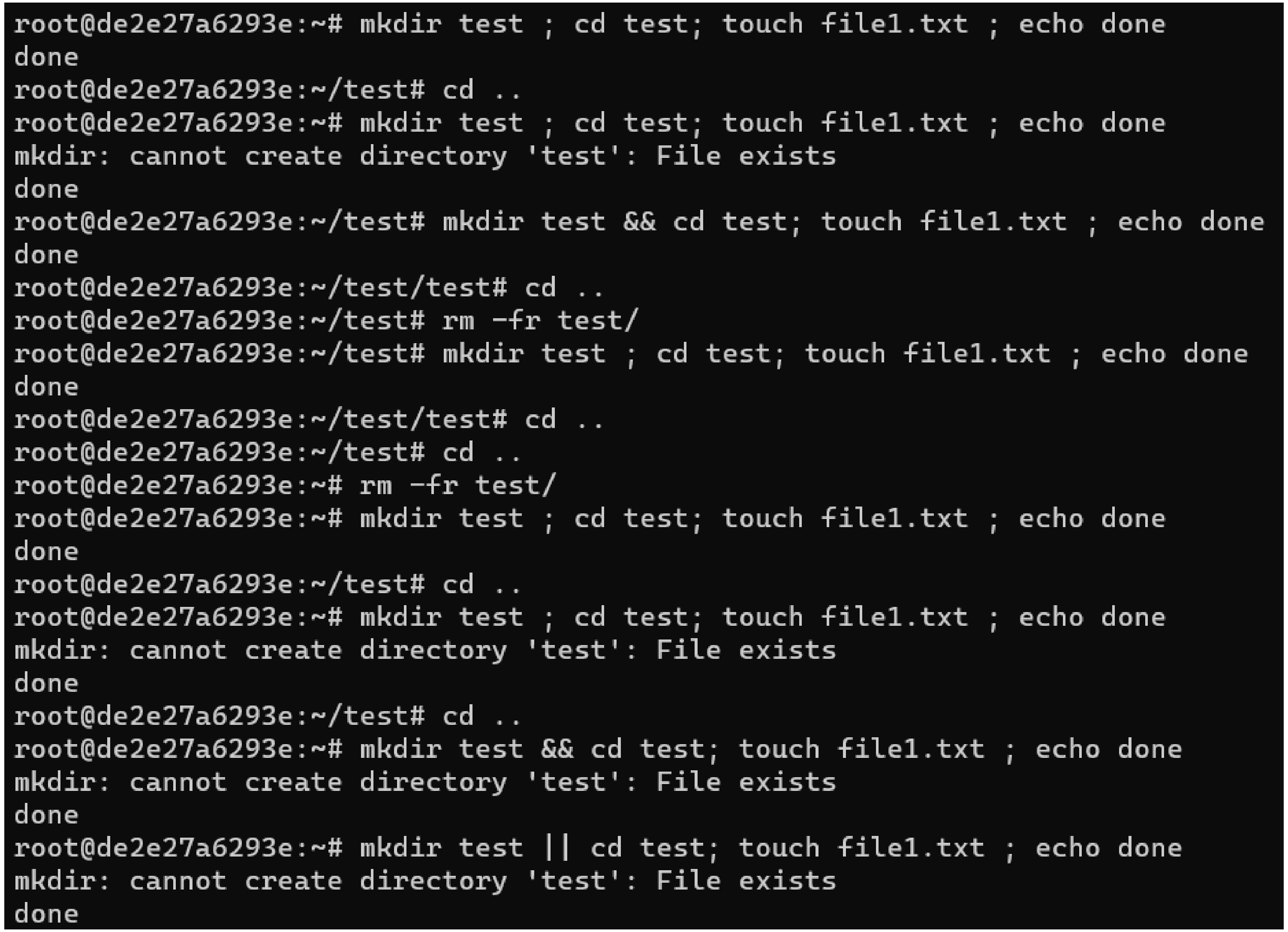
*In Linux and Unix Systems Grep, short for “**global regular expression print**”, is a command used in searching and matching text files contained in the regular expressions.*

Finding files and directories



```
root@de2e27a6293e:~# find
.
./.profile
./.bashrc
./allFilesInETC.txt
./combined.txt
./file1.txt
./file2.txt
./hello.txt
root@de2e27a6293e:~# find -type d
.
root@de2e27a6293e:~# find -type f
./.profile
./.bashrc
./allFilesInETC.txt
./combined.txt
./file1.txt
./file2.txt
./hello.txt
root@de2e27a6293e:~# find -type f -name f*
find: paths must precede expression: `file2.txt'
find: possible unquoted pattern after predicate `-name'?
root@de2e27a6293e:~# find -type f -name "f*"
./file1.txt
./file2.txt
root@de2e27a6293e:~# find -type f -iname "F*"
./file1.txt
./file2.txt
```

Chaining commands

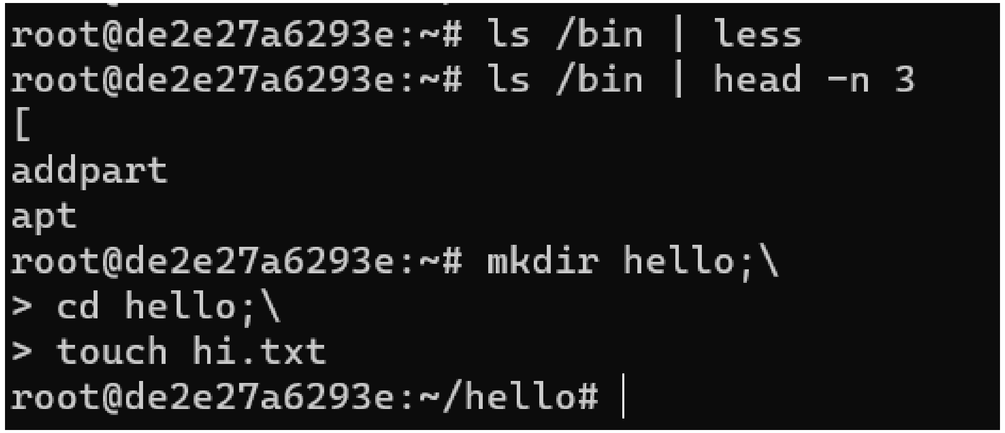


```
root@de2e27a6293e:~# mkdir test ; cd test; touch file1.txt ; echo done
done
root@de2e27a6293e:~/test# cd ..
root@de2e27a6293e:~# mkdir test ; cd test; touch file1.txt ; echo done
mkdir: cannot create directory 'test': File exists
done
root@de2e27a6293e:~/test# mkdir test && cd test; touch file1.txt ; echo done
done
root@de2e27a6293e:~/test/test# cd ..
root@de2e27a6293e:~/test# rm -fr test/
root@de2e27a6293e:~/test# mkdir test ; cd test; touch file1.txt ; echo done
done
root@de2e27a6293e:~/test/test# cd ..
root@de2e27a6293e:~/test# cd ..
root@de2e27a6293e:~# rm -fr test/
root@de2e27a6293e:~# mkdir test ; cd test; touch file1.txt ; echo done
done
root@de2e27a6293e:~/test# cd ..
root@de2e27a6293e:~# mkdir test ; cd test; touch file1.txt ; echo done
mkdir: cannot create directory 'test': File exists
done
root@de2e27a6293e:~/test# cd ..
root@de2e27a6293e:~# mkdir test && cd test; touch file1.txt ; echo done
mkdir: cannot create directory 'test': File exists
done
root@de2e27a6293e:~# mkdir test || cd test; touch file1.txt ; echo done
mkdir: cannot create directory 'test': File exists
done
```

You can use ;

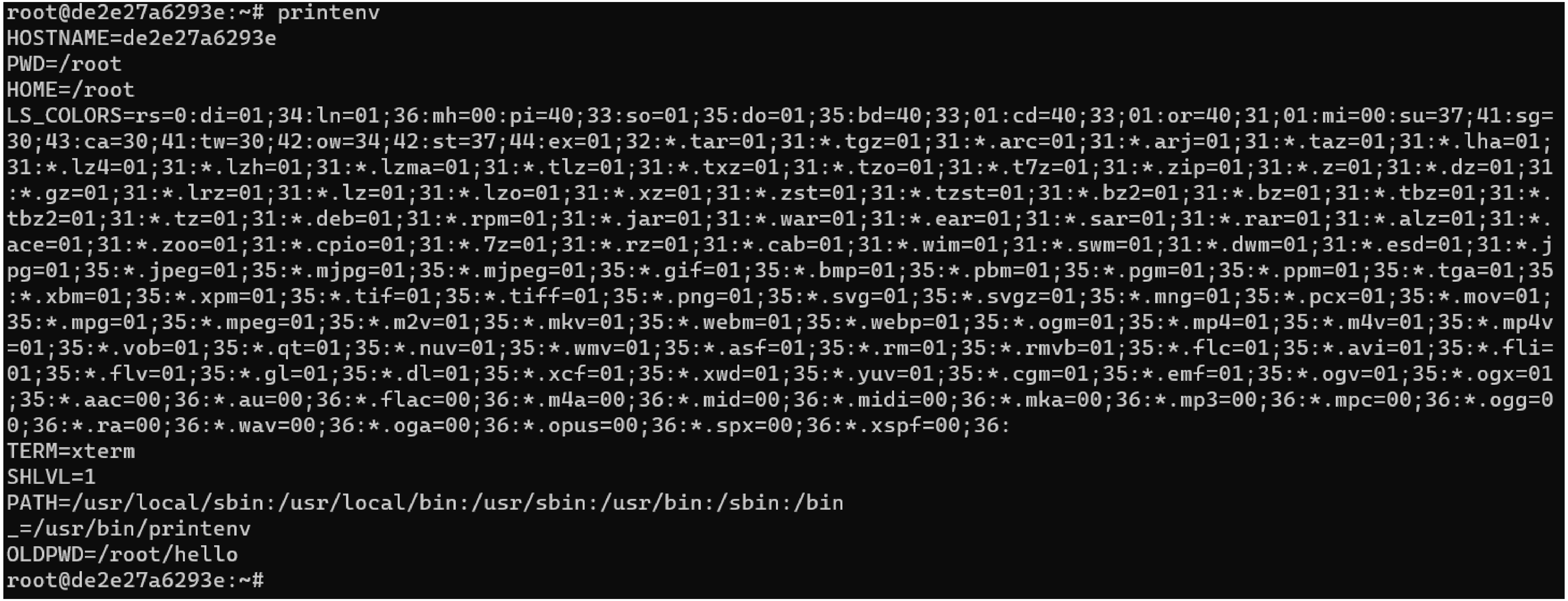
But if you want to stop if any commands failed you can use ***&&***

Also, to apply OR condition you can use this: ||



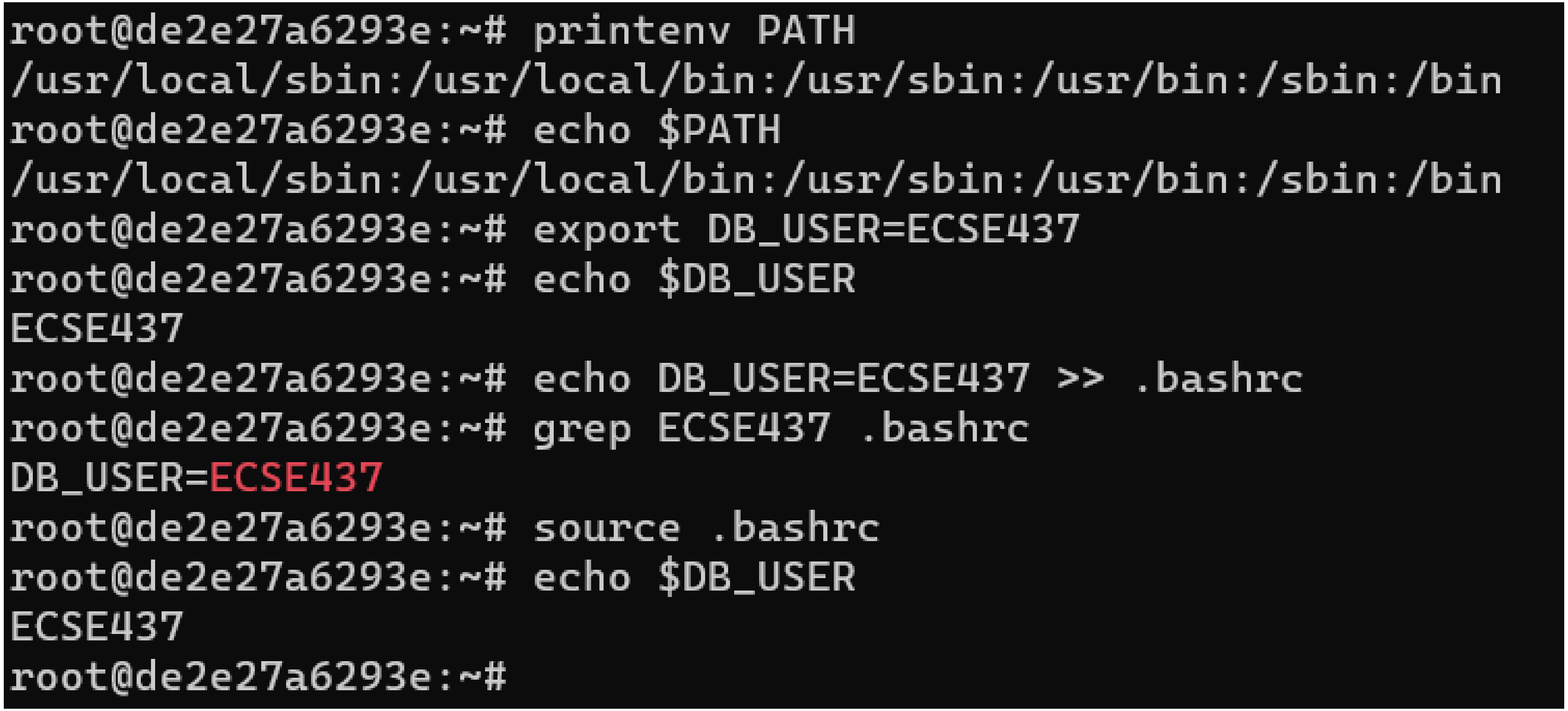
```
root@de2e27a6293e:~# ls /bin | less
root@de2e27a6293e:~# ls /bin | head -n 3
[
addpart
apt
root@de2e27a6293e:~# mkdir hello;\
> cd hello;\
> touch hi.txt
root@de2e27a6293e:~/hello# 
```

Finding files and directories



```
root@de2e27a6293e:~# printenv
HOSTNAME=de2e27a6293e
PWD=/root
HOME=/root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=
30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31
:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.
tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.
ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.j
pg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35
:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;
35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v
=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=
01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01
;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=0
0;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
TERM=xterm
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/printenv
OLDPWD=/root/hello
root@de2e27a6293e:~#
```

Modifying PATH



```
root@de2e27a6293e:~# printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root@de2e27a6293e:~# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root@de2e27a6293e:~# export DB_USER=ECSE437
root@de2e27a6293e:~# echo $DB_USER
ECSE437
root@de2e27a6293e:~# echo DB_USER=ECSE437 >> .bashrc
root@de2e27a6293e:~# grep ECSE437 .bashrc
DB_USER=ECSE437
root@de2e27a6293e:~# source .bashrc
root@de2e27a6293e:~# echo $DB_USER
ECSE437
root@de2e27a6293e:~#
```