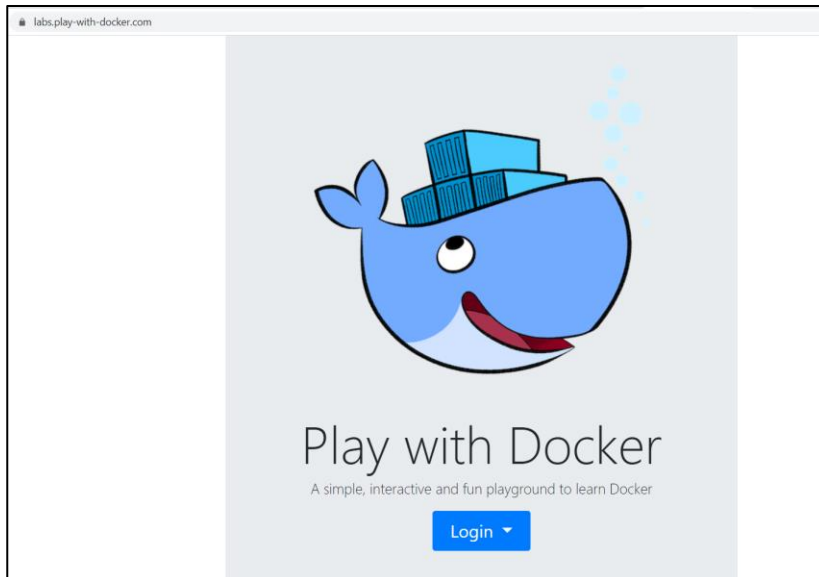


# Exercises: Linux Console Commands

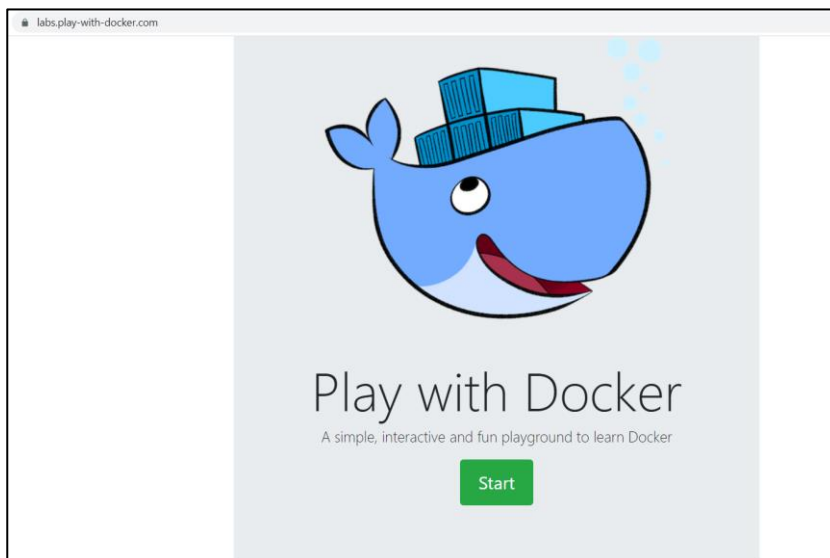
Problems for exercises for the ["Containers and Clouds" course @ SoftUni](#)

## 1. Configure Connectivity of the VM

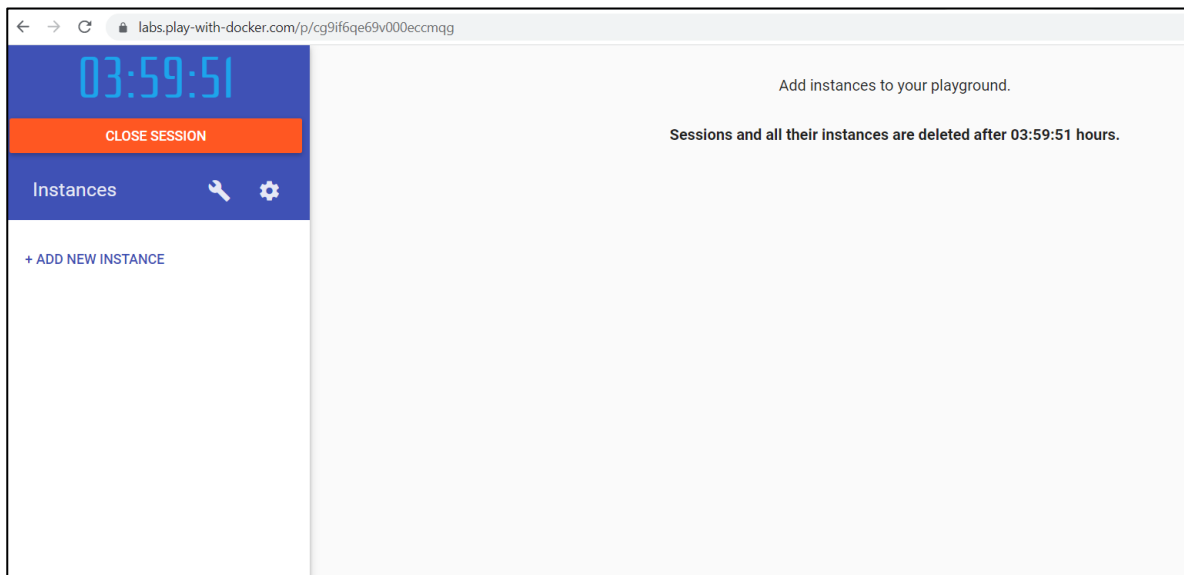
The first step is to open the link - <https://labs.play-with-docker.com/>.



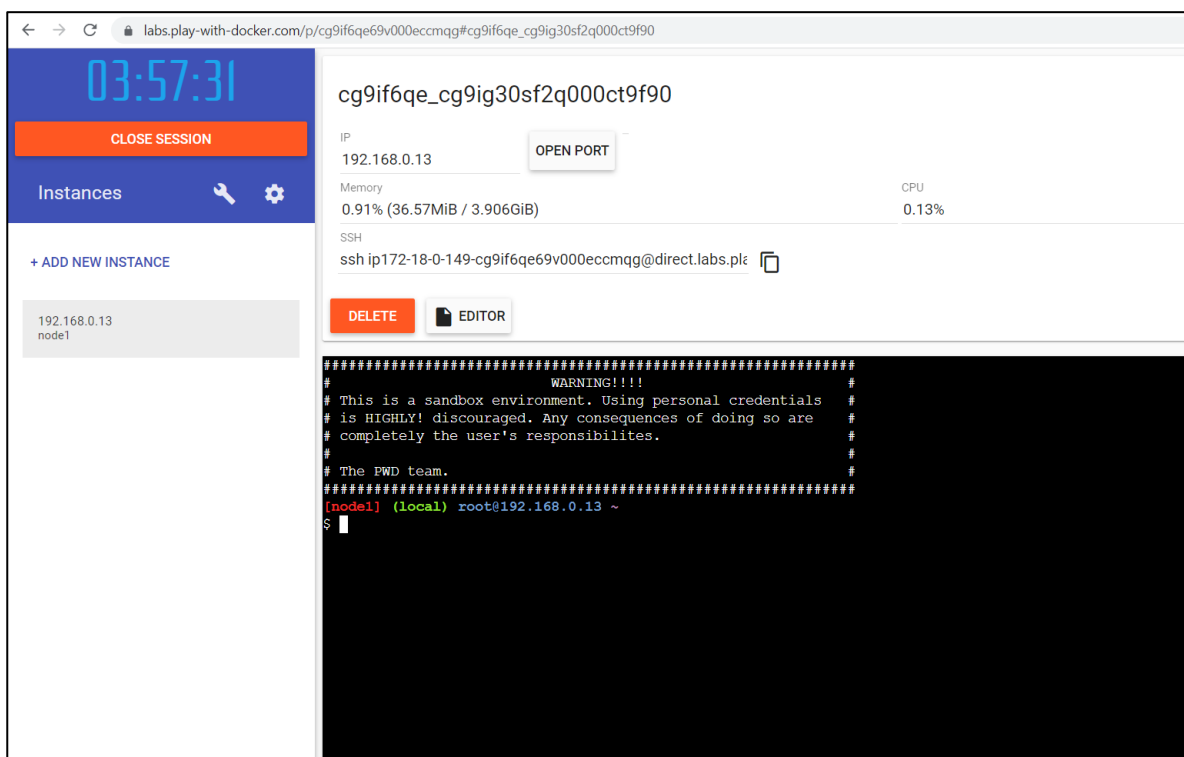
Then press the **[Login]** button and click on **"docker"**. A new dialog box opens, which is for **docker registration**. If you don't have a registration click on **[Sign Up]**. You have to create an account with a username, password and email. Then sign in to your account. It takes you to a page to select the plan you want – click on **"Continue to Free"** (Personal plan). Log in to the email you registered with and **confirm your account**. You may need to reload the page until you see this:



Click on the **[Start]** button.



You should see this. Click on [+ add new instance].



We are ready for work!

## 2. Getting to Know the Console

Let's start writing the command in the docker playground.

On the prompt, we can **input the user and password** that we created during the installation. Of course, we can use the **root user**, but it is not considered a good practice.

Now, let's check where (in which **folder**) we are with the **pwd** command:

```
[node1] (local) root@192.168.0.13 ~  
pwd  
/root
```

It appears that we are in our **home folder (/home/1sauser)**.

Check what we have here:

```
[node1] (local) root@192.168.0.13 ~  
$ ls
```

There is **nothing** or at least it appears to be this way. Of course, what we will see here on a **clean new installation** depends on what **distribution we chose** and **what settings** there are by default.

Even though we are not familiar with all options on the **ls** command yet, let's check if there are **any hidden files and folders** with:

```
[node1] (local) root@192.168.0.13 ~  
$ ls -al  
total 16  
drwx-----  1 root    root    18 Jul 25  2022 .  
drwxr-xr-x   1 root    root    57 Mar 16 14:30 ..  
-rw-rw-r--   1 root    root    43 Jan 17  2018 .gitconfig  
-rw-rw-r--   1 root    root   1865 Jan 17  2018 .inputrc  
-rw-rw-r--   1 root    root    207 Oct 22  2019 .profile  
drwxr-xr-x   2 root    root    61 Jul 25  2022 .ssh  
-rw-rw-r--   1 root    root    85 Jan 17  2018 .vimrc
```

Note that you can **autocomplete commands** by clicking the [Tab] button twice (Tab-Tab).

So, **there are some files** after all. Those are considered **hidden files**, because their name starts with the **dot symbol** (.). The same rule applies to **folders** as well.

Now, let's execute this:

```
[node1] (local) root@192.168.0.13 ~  
$ ls -a  
.  
..  
.gitconfig  .profile    .vimrc  
.inputrc    .ssh
```

We can also **give arguments**, not only options. For example, we can check what we have in the **main (root) folder**. The "/" symbol is used to state that we want to **access the root of our file system**:

```
[node1] (local) root@192.168.0.13 ~
$ ls -al /
total 336
drwxr-xr-x  1 root    root      57 Mar 16 14:30 .
drwxr-xr-x  1 root    root      57 Mar 16 14:30 ..
-rwxr-xr-x  1 root    root        0 Mar 16 14:30 .dockerenv
drwxr-xr-x  1 root    root     166 Jul 13  2022 bin
drwxrwxrwt  3 root    root      20 Jun 27  2022 certs
drwxr-xr-x 13 root    root    3660 Mar 16 14:30 dev
-rw-r--r--  1 root    root   340217 Mar 16 15:23 docker.log
drwxr-xr-x  1 root    root      95 Mar 16 14:30 etc
drwxr-xr-x  1 root    root     23 Jun 27  2022 home
drwxr-xr-x  1 root    root    208 Jul 13  2022 lib
drwxr-xr-x  2 root    root      34 May 27  2022 lib64
drwxr-xr-x  5 root    root      44 May 23  2022 media
drwxr-xr-x  2 root    root        6 May 23  2022 mnt
drwxr-xr-x  1 root    root     24 Mar 16 14:30 opt
dr-xr-xr-x 1637 root    root        0 Mar 16 14:30 proc
drwx----- 1 root    root      18 Jul 25  2022 root
drwxr-xr-x  1 root    root    117 Mar 16 14:30 run
drwxr-xr-x  1 root    root      22 Jul 13  2022 sbin
drwxr-xr-x  2 root    root        6 May 23  2022 srv
```

```
drwxrwxrwx 13 root    root        0 Jan 24 19:50 sys
drwxrwxrwt  2 root    root        6 May 23  2022 tmp
drwxr-xr-x  1 root    root     19 Jul 13  2022 usr
drwxr-xr-x  1 root    root     41 Jul 13  2022 var
```

We can note that there is a **folder** named **"/root"**. This is the **home folder for the root user**.

Let's **change the folder**. For example, go to the **"/etc"** folder. This is the place where most of the **configuration files** are stored. Then we can check if indeed we **changed the folder**:

```
[node1] (local) root@192.168.0.13 ~
$ cd /etc
[node1] (local) root@192.168.0.13 /etc
$ pwd
/etc
```

As we can see, there is **no need** to execute **pwd**. The **prompt reflects or shows** where in the **file system tree** we are currently.

When we want to **address all files which name starts with something**, no matter what, and how many symbols their name contains, we can use the **"\*"** symbol. For example, ask for **all files** starting with **"os\*"**:

```
[node1] (local) root@192.168.0.13 /etc
$ ls os*
os-release
```

Okay, now that we know that this **file exists**, let's check **what it contains** (the actual output may be different):

```
[node1] (local) root@192.168.0.13 /etc
$ cat os-release
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.16.0
PRETTY_NAME="Alpine Linux v3.16"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"
```

It appears, that this file contains **detailed information about our distribution**.

**Similar or additional information** about the distribution we can get by executing:

```
[node1] (local) root@192.168.0.13 /etc
$ uname -a
Linux node1 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86_64 Linux
```

Beside the information about the kernel, we can extract information about the **name of the host**:

```
[node1] (local) root@192.168.0.13 /etc
$ hostname
node1
```

Now, let's return to our home folder:

```
[node1] (local) root@192.168.0.13 /etc
$ cd
[node1] (local) root@192.168.0.13 ~
$ pwd
/root
```

As we can see, if we execute the **cd** command **without any arguments**, the result is that we are "**back home**". There is also a **special symbol** that we can use – it is again the tilde symbol – "~".

Now, let's try a different approach. In general, no matter what distribution we use, there is a **common set of commands** that is always available.

```
lsauser@ubuntu:~$ cat /etc/hostname
softuni.lsa.lab
lsauser@ubuntu:~$ cat /etc/machine-info
PRETTY_HOSTNAME="SoftUni Lab Server"
```

In order the **changes to be reflected in the prompt**, we must **close the session**, and **open a new one**. So, type **logout** and **log back in**:

```
[node1] (local) root@192.168.0.13 ~
$ logout
#####
#                               WARNING!!!!                               #
# This is a sandbox environment. Using personal credentials               #
# is HIGHLY! discouraged. Any consequences of doing so are               #
# completely the user's responsibilities.                                  #
#                                                                           #
# The PWD team.                                                            #
#####
```

We can see that the **new name is applied**.

Now, we can check **what date is today** and **what is the time now**:

```
[node1] (local) root@192.168.0.13 ~  
$ date  
Thu Mar 16 15:32:34 UTC 2023
```

There is a way to **modify the output** of the **date** command:

```
[node1] (local) root@192.168.0.13 ~  
$ date +%Y-%m-%d  
2023-03-16
```

This way, we will receive the **current date**, represented in "YYYY-MM-DD" format.

Should we need a **calendar** on the command line, we can have it easily with:

```
[node1] (local) root@192.168.0.13 ~  
$ cal -3  
      February 2023      March 2023      April 2023  
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  
      1  2  3  4          1  2  3  4          1  
  5  6  7  8  9 10 11   5  6  7  8  9 10 11   2  3  4  5  6  7  8  
12 13 14 15 16 17 18   12 13 14 15 16 17 18   9 10 11 12 13 14 15  
19 20 21 22 23 24 25   19 20 21 22 23 24 25   16 17 18 19 20 21 22  
26 27 28              26 27 28 29 30 31       23 24 25 26 27 28 29  
                          30
```

If we need to know **since when or how long our system is operating**, we can do it with:

```
[node1] (local) root@192.168.0.13 ~  
$ uptime  
15:33:58 up 56 days, 23:41,  0 users,  load average: 13.56, 21.57, 23.02
```

At any point we can ask for the **history of executed commands**:

```
[node1] (local) root@192.168.0.13 ~
$ history
 1  pwd
 2  ls
 3  ls -al
 4  ls -a
 5  ls -al /
 6  cd /etc
 7  pwd
 8  ls os*
 9  cat os-release
10  uname -a
11  hostname
12  cd
13  pwd
14  hostnamectl
15  hostnamectl
16  sudo hostnamectl
17  systemd
18  hostnamectl
19  sudo hostnamectl set-hostname --pretty 'SoftUni Lab Server'
20  logout
21  date
```

```
21  date
22  date +%Y-%m-%d
23  cal -3
24  uptime
25  history
```

If we want to **end our session**, we can do it with either **exit** or **logout**. Issuing any of these will close our session but will leave the machine up and running. Let's type **exit** and press the **[Enter]** key. Our **session is closed** now:

```
[node1] (local) root@192.168.0.13 ~
$ exit
logout
#####
#                                     WARNING!!!!                                #
# This is a sandbox environment. Using personal credentials                    #
# is HIGHLY! discouraged. Any consequences of doing so are                    #
# completely the user's responsibilities.                                       #
#                                     #                                           #
# The PWD team.                                                                #
#####
```

Now **log in back again** and **ask for all files** (including hidden ones) in our **home folder**:

```
[node1] (local) root@192.168.0.13 ~
$ ls -a
.          .bash_history  .inputrc    .ssh
..         .gitconfig    .profile    .vimrc
```

It appears that there is a **special file** that takes care for our **history** – **.bash\_history**. Let's check its **contents**:

```
[node1] (local) root@192.168.0.13 ~
$ cat .bash_history
pwd
ls
ls -al
ls -a
ls -al /
cd /etc
pwd
ls os*
cat os-release
uname -a
hostname
cd
pwd
hostnamectl
hostnamectl
sudo hostnamectl
systemd
hostnamectl
sudo hostnamectl set-hostname --pretty 'SoftUni Lab Server'
logout
date
```

```
date +%Y-%m-%d
cal -3
uptime
history
exit
```

As we can see the **last few commands are not here** and there is a perfect explanation for this. The reason is that they are **kept in a buffer** and are stored on the disk only when certain events occur, like session end.

Now let's continue with the console command explanations and transfer files.

### 3. Getting to Know Environment Variables

Let's start with the **environment exploration process**. Try out the **set** command and look at its result:

```
[node3] (local) root@192.168.0.11 ~
$ set | less
```

↓



```

BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_ignore_globasciiranges:interactive_comments:log
in_shell:progcomp:promptvars:sourcepath
BASHPID=17
BASH_ALIASES=()
BASH_ARGC=([0]="0")
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSION=([0]="2" [1]="11")
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSION=([0]="5" [1]="1" [2]="16" [3]="1" [4]="release" [5]="x86_64-alpine-linux-musl")
BASH_VERSION='5.1.16(1)-release'
CHARSET=UTF-8
COLUMNS=135
COMPOSE_PLUGIN_PATH=/usr/lib/docker/cli-plugins/docker-compose
COMPOSE_VERSION=2.6.1
DIND_COMMIT=42b1175eda071c0e9121e1d64345928384a93df1
DIRSTACK=()
DOCKER_BUILDX_VERSION=0.8.2
DOCKER_CLI_EXPERIMENTAL=enabled
DOCKER_COMPOSE_VERSION=2.6.1
standard input

```

When executed **without parameters**, **set** returns **information about the environment – variables and functions**. Again, the information will differ from distribution to distribution and between versions of distribution.

We added **| less** to the **set** command to **display the command output, one page at a time**. It is **optional**, but it's a good idea to use it when the **output is longer**. You can go to the **next pages** with **[Enter]** or **quit** with **[q]**.

The **set** command can be used to **modify the parameters** that are driving the environment. In order to see what **parameters** are there, we can execute:

```

[node1] (local) root@192.168.0.12 ~
$ set -o
allexport            off
braceexpand         on
emacs               on
errexit             off
errtrace            off
functrace           off
hashall             on
histexpand          on
history             on
ignoreeof           off
interactive-comments on
keyword             off
monitor             on
noclobber           off
noexec              off
noglob              off
nolog               off
notify              off
nounset             off
onecmd              off

```

We can **see the same information** but prepared for **re-use** with:

```
[node1] (local) root@192.168.0.12 ~
$ set +o
set +o allexport
set -o braceexpand
set -o emacs
set +o errexit
set +o errtrace
set +o functrace
set -o hashall
set -o histexpand
set -o history
set +o ignoreeof
set -o interactive-comments
set +o keyword
set -o monitor
set +o noclobber
set +o noexec
set +o noglob
set +o nolog
set +o notify
set +o nounset
set +o onecmd
```

Let's **change the flag** that **controls the amount of information** shown during commands execution:

```
[node1] (local) root@192.168.0.12 ~
$ set -x
++ docker-prompt
++ hostname -i
```

You should note that the minus ("-") is used to **turn on a flag**, while the plus ("+") is used to **deactivate a flag**. Now, we can execute few more commands:

```
[node1] (local) root@192.168.0.12 ~
$ pwd
+ pwd
/root
++ docker-prompt
++ hostname -i
```

```
[node1] (local) root@192.168.0.12 ~
$ ls -al
+ ls -al
total 16
drwx-----  1 root    root      18 Jul 25  2022 .
drwxr-xr-x   1 root    root     57 Mar 16 15:51 ..
-rw-rw-r--   1 root    root     43 Jan 17  2018 .gitconfig
-rw-rw-r--   1 root    root    1865 Jan 17  2018 .inputrc
-rw-rw-r--   1 root    root     207 Oct 22  2019 .profile
drwxr-xr-x   2 root    root     61 Jul 25  2022 .ssh
-rw-rw-r--   1 root    root     85 Jan 17  2018 .vimrc
++ docker-prompt
++ hostname -i
```

We can see that immediately **after each command** there is **information aboutse what is going to be executed**. This way we can see that instead **ls**, in fact something else is being executed – its **alias** or the statement it specifies.

It is time to **deactivate** the **xtrace** mode:

```
[node1] (local) root@192.168.0.12 ~  
$ set +x  
+ set +x
```

Now you will not be able to see what commands are executed.

## 4. Getting Help

Now it is time to explore ways of **getting help** about different commands.

For the **shell built-in commands**, we can use the **help** command. If we execute it **without parameters**, it will return **all built-in commands**:

```
[node1] (local) root@192.168.0.12 ~  
$ help | less
```

↓

```
GNU bash, version 5.1.16(1)-release (x86_64-alpine-linux-musl)  
These shell commands are defined internally. Type 'help' to see this list.  
Type 'help name' to find out more about the function 'name'.  
Use 'info bash' to find out more about the shell in general.  
Use 'man -k' or 'info' to find out more about commands not in this list.  
  
A star (*) next to a name means that the command is disabled.  
  
job_spec [&]  
(( expression ))  
. filename [arguments]  
:  
[ arg... ]  
[[ expression ]]  
alias [-p] [name[=value] ... ]  
bg [job_spec ...]  
bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u name] [>  
break [n]  
builtin [shell-builtin [arg ...]]  
caller [expr]  
case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;)... esac  
cd [-L|[-P [-e]] [-@]] [dir]  
standard input  
  
history [-c] [-d offset] [n] or history -anrw [filename] or his>  
if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]...>  
jobs [-lnprs] [job_spec ...] or jobs -x command [args]  
kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or k>  
let arg [arg ...]  
local [option] name[=value] ...  
logout [n]  
mapfile [-d delim] [-n count] [-O origin] [-s count] [-t] [-u f>  
popd [-n] [+N | -N]  
printf [-v var] format [arguments]  
pushd [-n] [+N | -N | dir]  
pwd [-LP]  
read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nch>  
readarray [-d delim] [-n count] [-O origin] [-s count] [-t] [-u>
```

In order to **ask for a command**, we must execute it like:

```
[node1] (local) root@192.168.0.12 ~  
$ help cd | less
```

↓

```
cd: cd [-L|[-P [-e]] [-@]] [dir]
Change the shell working directory.

Change the current directory to DIR. The default DIR is the value of the
HOME shell variable.

The variable CDPATH defines the search path for the directory containing
DIR. Alternative directory names in CDPATH are separated by a colon (:).
A null directory name is the same as the current directory. If DIR begins
with a slash (/), then CDPATH is not used.

If the directory is not found, and the shell option `cdable_vars' is set,
the word is assumed to be a variable name. If that variable has a value,
its value is used for DIR.

Options:
  -L      force symbolic links to be followed: resolve symbolic
          links in DIR after processing instances of `..'
  -P      use the physical directory structure without following
          symbolic links: resolve symbolic links in DIR before
          processing instances of `..'
  -e      if the -P option is supplied, and the current working
```

standard input

Most **external commands** offer integrated help. The ways to **ask for this information** vary, but typically we can use:

```
[node1] (local) root@192.168.0.28 ~
$ ls --help
BusyBox v1.35.0 (2022-05-09 17:27:12 UTC) multi-call binary.

Usage: ls [-lAaCxdLHRFpplinshrSXvctu] [-w WIDTH] [FILE]...

List directory contents

  -l      One column output
  -a      Include names starting with .
  -A      Like -a, but exclude . and ..
  -x      List by lines
  -d      List directory names, not contents
  -L      Follow symlinks
  -H      Follow symlinks on command line
  -R      Recurse
  -p      Append / to directory names
  -F      Append indicator (one of */=@|) to names
  -l      Long format
  -i      List inode numbers
  -n      List numeric UIDs and GIDs instead of names
  -s      List allocated blocks
```

When using **man**, it is good to know the following **set of keys**:

- Key **[h]** shows **help information**
- Key **[q]** either **exits the help mode** or **the man**
- If we want to **search for something** from the current cursor position onwards, then we can press **[/]**, then enter the string and then press the **[Enter]** key. Once in this mode, we can press **[n]** key to **move forward** over the matches, or the **[N]** key to **move backward**
- Searching backwards works the same, but it is initiated with the **[?]** key

A **quick search** in **man** can be done on the command line with:

```
lsausser@softuni:~$ man -k passwd
chpasswd (8)      - update group passwords in batch mode
chpasswd (8)      - update passwords in batch mode
gpasswd (1)       - administer /etc/group and /etc/gshadow
grub-mkpasswd-pbkdf2 (1) - generate hashed password for GRUB
openssl-passwd (1ssl) - compute password hashes
pam_localuser (8) - require users to be listed in /etc/passwd
passwd (1)        - change user password
passwd (1ssl)     - compute password hashes
passwd (5)        - the password file
update-passwd (8) - safely update /etc/passwd, /etc/shadow and /etc/group
```

Similar effect can be achieved by using the **apropos** command:

```
lsausser@softuni:~$ apropos passwd
chpasswd (8)      - update group passwords in batch mode
chpasswd (8)      - update passwords in batch mode
gpasswd (1)       - administer /etc/group and /etc/gshadow
grub-mkpasswd-pbkdf2 (1) - generate hashed password for GRUB
openssl-passwd (1ssl) - compute password hashes
pam_localuser (8) - require users to be listed in /etc/passwd
passwd (1)        - change user password
passwd (1ssl)     - compute password hashes
passwd (5)        - the password file
update-passwd (8) - safely update /etc/passwd, /etc/shadow and /etc/group
```

Beside **man**, usually there is an alternative and modern help system available – **info**. We can use it the same way:

```
lsausser@softuni:~$ info ls
```

↓

```
Next: dir invocation, Up: Directory listing
```

```
10.1 'ls': List directory contents
```

```
=====
```

```
The 'ls' program lists information about files (of any type, including
directories). Options and file arguments can be intermixed arbitrarily,
as usual.
```

```
For non-option command-line arguments that are directories, by
default 'ls' lists the contents of directories, not recursively, and
omitting files with names beginning with '.'. For other non-option
arguments, by default 'ls' lists just the file name. If no non-option
argument is specified, 'ls' operates on the current directory, acting as
```

```
-----Info: (coreutils)ls invocation, 57 lines --Top-----
```

```
Welcome to Info version 6.7. Type H for help, h for tutorial.
```

For **help** inside the tool, we can press the **[h]** key. The help screen can be **closed** with the **[x]** key. No matter where we are in the tool, we can **exit** with the **[q]** key.