## **Linux Home Directory**

The Linux home directory is a directory for a particular user of the system and consists of individual files. It is also referred to as the **login directory**. This is the first place that occurs after logging into a Linux system. It is automatically created as **"/home"** for each user in the directory'. It is a standard subdirectory of the root directory. The root directory contains all other directories, subdirectories, and files on the system. It is denoted by a forward slash (/).

The home directory can be said as a personal working space for all the users except root. There is a separate directory for every user.

## **Hidden files inside home directory**

* + .bash\_history
  + .bash\_profile
  + .bashrc
  + .bash\_logout
  + .viminfo
  + .cache

**Precedence**

* /etc/profile
* ~/.bash\_profile
* ~/.bashrc
* ~/.bash\_login
* ~/.profile
* ~/.bash\_logout

## **Login Shell vs Non Login Shell**

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

**Login Shell**

Login shell is started after a successful login, using ***/bin/login***, by reading the ***/etc/passwd*** file. Login shell is the first process that executes under our user ID when we log in to a session. The login process tells the shell to behave as a login shell with a convention: passing argument ***0***, which is normally the name of the shell executable, with a “**-**” character prepended. For example, for [Bash](https://gnu.org/software/bash/) shell it will be ***-bash***.

A **Login Shell** executes following scripts:

* Login shell executes /etc/profile
* /etc/profile executes all scripts in /etc/profile.d/
* Then executes users ~/.bash\_profile
* ~/.bash\_profile executes users ~/.bashrc
* ~/.bashrc executes /etc/bashrc

Shells created by explicitly telling to login.

**examples:** # **su -** | # **su -l** | # **su --login** | # **su USERNAME -** | # **su -l USERNAME** | # **su --login USERNAME** | # **sudo -i**

**Non Login Shell**

* Sub shell
  + Sub shell, also called a non-login shell is a shell started after the login process without the -l or --login option and without an extra dash before the command name

A Non login shell is started by a program without a login. In this case, the program just passes the name of the shell executable. For example, for a [Bash](https://gnu.org/software/bash/) shell it will be simply ***bash***.

**When bash is invoked as a Non login shell**

→Non-login process(shell) calls ***~/.bashrc***  
→~/.bashrc calls ***/etc/bashrc***  
→/etc/bashrc calls the scripts in ***/etc/profile.d/***

* Non login shells include the following.

•Shells created using the below command syntax.  
**examples: # su | # su USERNAME**  
• Graphical terminals  
• Executed scripts  
• Any other bash instances

#### **What Shell Type Do We Have?**

In order to find out what type of shell we are working at, we can type echo $0 into the terminal and get the following output:

Interactive login

-bash or -su

Interactive non-login

bash or /bin/bash

Non-interactive non-login (scripts)

<name\_of\_script>

Alternate method

$ shopt login\_shell

**Script will check login or non login shell**

$ shopt -q login\_shell && echo "Login shell" || echo "Not login shell"

## **Interactive Login shell vs Non Interative shell**

In general, a shell has two modes of operation. It can run as an interactive login shell and as a non-interactive batch shell. The mode of operation defines the Bash startup and which configuration files are read [7]. The mode of operation can be differentiated as follows [6] — interactive login shell, interactive non-login-shell, non-interactive login shell, and non-interactive (batch) non-login shell.

Interactive Login shell

An interactive shell reads commands from user input on a tty. Among other things, such a shell reads startup files on activation, displays a prompt, and enables job control by default. The user can interact with the shell.

To put it simply, an interactive shell reads and writes to a user’s terminal.

* /etc/profile: if it exists, it runs the commands listed in the file.
* ~/.bash\_profile, ~/.bash\_login, and ~/.profile (in that order). It executes the commands from the first readable file found from the list. Each individual user can have their own set of these files.

Example

$ ssh user@remote-system

$ ssh user@remote-system remote-command

The first command connects to the remote system and opens an interactive shell only. In contrast, the second command connects to the remote system, executes the given command in a non-interactive login shell, and terminates the ssh connection

Interactive Non Login shell

This mode describes opening a new terminal, for example, xterm or Gnome Terminal, and executing a shell in it. In this mode, the two files /etc/bashrc and ~/.bashrc are read.

* /etc/bash.bashrc: global Bash configuration. It executes the commands if that file exists, and it is readable. Only available in Debian GNU/Linux, Ubuntu, and Arch Linux.
* ~/.bashrc: local Bash configuration. It executes the commands if that file exists, and it is readable.

Non-Interactive Non Login shell

A shell running a script is always a non-interactive shell. This mode is in use when executing a shell script. The shell script runs in its own subshell. It is classified as non-interactive unless it asks for user input. The shell only opens to execute the script and closes it immediately once the script has terminated.

Non-Interactive Login shell

This mode covers logging into a computer from a remote, for example, via Secure Shell (ssh). The shell script local-script.sh is run locally, first, and its output is used as the input of ssh

$ ./local-script.sh | ssh user@remote-system

## **Linux Bash History [.bash\_history ]**

* The Linux bash comes with a very powerful command called “history”. This command is a built-in bash command, used to extract the history information about commands that have been executed by Linux users in all the previous sessions.
* Linux stores the command history for a certain user in a file ~/.bash\_history by default.

**Viewing the Entire History**

$ history #display entire history with number

**Filtering the History Output**

$ history | grep [keyword]

$ history | grep ping

**List All Commands with Date and Timestamp**

$ export HISTTIMEFORMAT='%F, %T '

**Running Commands from History**

$ !# # represents the number

$ !92

**Clearing History**

$ history -c

**Setting History Defaults**

Bash allows you to adjust the number of previous commands that it stores in history. It actually has two separate options for this: the HISTFILESIZE parameter configures how many commands are kept in the history file, while the HISTSIZE controls the number stored in memory for the current session.

This means you can set a reasonable cap for the size of history in memory for the current session, and have an even larger history saved to disk that you can examine at a later time. By default, bash sets very conservative values for these options, so we’ll expand them to take advantage of a larger history. Some distributions already increase the default bash history settings with slightly more generous values

Open your ~/.bashrc file with an editor to change these settings

Search for both the HISTSIZE and HISTFILESIZE parameters. If they are set, feel free to modify the values. If these parameters aren’t in your file, add them now

HISTSIZE=5000

HISTFILESIZE=10000

By default, bash writes its history at the end of each session, overwriting the existing file with an updated version. This means that if you are logged in with multiple bash sessions, only the last one to exit will have its history saved.

We can work around this by setting the histappend setting, which will append instead of overwrite the history. This may be set already, but if it is not, you can enable this by adding this line:

shopt -s histappend

If we want to have bash immediately add commands to our history instead of waiting for the end of each session (to enable commands in one terminal to be instantly be available in another)

export PROMPT\_COMMAND="history -a; history -c; history -r; $PROMPT\_COMMAND"

source ~/.bashrc

## **[.bash\_logout ]**

When exiting **bash**, it can execute **~/.bash\_logout**.

This is the individual login shell cleanup file, executed when a [login](https://bash.cyberciti.biz/guide/Login) shell exits. This file exists in users home directory i.e. $HOME/.bash\_logout. This is useful if you want to run task or another script or command automatically at [logout](https://bash.cyberciti.biz/guide/Logout). For example, [clear the mysql command line] history stored in ~/.mysql\_history or make backup of files you can use this file.

Example #1

**if** **[** "$SHLVL" = 1 **]**; **then**

*#clear screen*

**[** -x **/**usr**/**bin**/**clear\_console **]** **&&** **/**usr**/**bin**/**clear\_console -q

*# delete mysql history*

**[** -f $HOME**/**.mysql\_history **]** **&&** **/**bin**/rm** $HOME**/**.mysql\_history

*# Update ip accounting*

**[** -x **/**usr**/**bin**/**ip\_accouting **]** **&&** **/**usr**/**bin**/**ip\_accouting -u "$USER" -a

*# call your script here*

**[** -x **/**usr**/**local**/**bin**/**timesheet\_client.pl **]** **&&** **/**usr**/**local**/**bin**/**timesheet\_client.pl

**fi**

Example #2

clean

*# clear history*

/bin/rm $HOME/.mysql\_history

*# Backup files to NAS server*

~/bin/backup.sh

**The .bash\_logout provides a way to execute commands when the user logs out of the system.** One useful application for this in a user’s home directory is that an administrator can provide a way to kill all of the user’s applications when logging out just in case an application was not closed correctly or that hangs.

Some ideas for your logout file are: A command like fortune to give you something fun to think about when you log out. A command to list a "reminder" file -- for example, work to take home. A script that prompts you for the hours you've worked on projects so you can make a timesheet later. The command clear to erase your screen. docstore.mik.ua/orelly/unix3/upt/ch04\_17.htm

## **[.bashrc ]**

* The Linux bash comes with a very powerful command called “history”. This command is a built-in bash command, used to extract the history information about commands that have been executed by Linux users in all the previous sessions.
* Linux stores the command history for a certain user in a file ~/.bash\_history by default.

**Viewing the Entire History**

$ history #display entire history with number

**Filtering the History Output**

$ history | grep [keyword]

$ history | grep ping

**List All Commands with Date and Timestamp**

$ export HISTTIMEFORMAT='%F, %T '

**Running Commands from History**

$ !# # represents the number

$ !92

**Clearing History**

$ history -c

**Setting History Defaults**

Bash allows you to adjust the number of previous commands that it stores in history. It actually has two separate options for this: the HISTFILESIZE parameter configures how many commands are kept in the history file, while the HISTSIZE controls the number stored in memory for the current session.

This means you can set a reasonable cap for the size of history in memory for the current session, and have an even larger history saved to disk that you can examine at a later time. By default, bash sets very conservative values for these options, so we’ll expand them to take advantage of a larger history. Some distributions already increase the default bash history settings with slightly more generous values

Open your ~/.bashrc file with an editor to change these settings

Search for both the HISTSIZE and HISTFILESIZE parameters. If they are set, feel free to modify the values. If these parameters aren’t in your file, add them now

HISTSIZE=5000

HISTFILESIZE=10000

By default, bash writes its history at the end of each session, overwriting the existing file with an updated version. This means that if you are logged in with multiple bash sessions, only the last one to exit will have its history saved.

We can work around this by setting the histappend setting, which will append instead of overwrite the history. This may be set already, but if it is not, you can enable this by adding this line:

shopt -s histappend

If we want to have bash immediately add commands to our history instead of waiting for the end of each session (to enable commands in one terminal to be instantly be available in another)

export PROMPT\_COMMAND="history -a; history -c; history -r; $PROMPT\_COMMAND"

source ~/.bashrc