



Linux Command Line

Linux Fundamentals

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Welcome to Linux Command Line.

What you will learn



At the core of the lesson

You will learn how to:

- Describe the login workflow
- Explain the Linux command syntax
- Perform basic operations at the command line
- Explain standard input, standard output, and standard error

In this lesson, you will learn how to:

- Describe the login workflow
- Explain the Linux command syntax
- Perform basic operations at the command line
- Explain standard input, standard output, and standard error



Linux login workflow

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In this section, you'll learn about the Linux login workflow.

The login prompt

- Enter the user name and password (verified against what the system has on file).

password:

- Access is either granted or denied.

```
CentOS Linux 7 (Core)  
Kernel 3.10.0-957.1.3.el7.x86_64 on an x86_64  
server00 login: _
```

After a network connection is made, you can connect by using a program like Putty or by using the terminal on Mac OS. You will encounter the login prompt. All Linux sessions begin with the login process (default authentication process). Linux sessions start with the user entering their user name at the prompt. The login prompt is used to authenticate (prove the user's identity) before using a Linux system. When the password is typed, it does not echo (a line of text isn't displayed).

The user name is checked against the `/etc/.passwd` file, which is stored in the `/etc` directory. The file represents an individual user account and contains the following fields separated by colons (:).

1. User name or login name
2. Encrypted password
3. User ID
4. Group ID
5. User description
6. User's home directory
7. User's login shell

The login workflow

The user name is checked against the `/etc/passwd` file.

```
login: c_salazar
```

The password is checked against the `/etc/shadow` file.

```
server00 login: userA
Password:
Last login: Thu Mar  7 21:28:52 on :0
userA@server00 ~1$ pwd
/home/userA
```

The `pwd` command is a utility for printing the current working directory.

During the login workflow, the name is checked against the `/etc/passwd` file, and the password is checked against the `/etc/shadow` file.

The user name field has a maximum of 32 characters. To avoid any confusion, avoid using initial capitalization in user name. For example, "salazar" is not the same as "Salazar." Linux has moved the user's password into a separate file `/etc/shadow`. Because the user's password is stored in the `/etc/shadow` file, a placeholder is used in `/etc/.passwd` as a reference.

The login workflow

The user's profile is loaded from files that are stored in the user's home directory:

- `/home/username/.bashrc`
- `/home/username/.bash_history`
- `/home/username/.bash_profile`

```
server00 login: userA
Password:
Last login: Thu Mar  7 21:28:52 on :0
[userA@server00 ~]$ pwd
/home/userA
```

The user's present working directory will be their own home directory.

The user is presented with a command prompt. The user's current working directory will be their home directory.



Linux command prompt

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In this section, you'll learn about the Linux command prompt.

Anatomy of the command prompt

```
login: c_salazar
```

```
Password:
```

```
[c_salazar@hostname ~]$ pwd
```




```
/home/c_salazar
```

```
[username@hostname ~]$
```

The default shell in Linux is Bash, which provides the command prompt. The Linux command prompt or command line is a text interface to your Linux computer. It is commonly referred to as the shell, terminal, console, or prompt. The default shell contains the user name, and the user's home directory has the same name as the user (which it does by default). Although this format is common, it is changeable and not always consistent in different Linux distributions.

Command syntax example

Linux command example:

man	-i	whoami
 man command shows the man page	-i option is to perform a case-insensitive search	whoami argument is what is being searched for
 Command: What you want Linux to do	Option: Modifies the command	Argument: What the command acts on
 Command is drive	Option is to turn on headlights	Argument is a car

The command syntax is case-sensitive.

Most commands in Linux follow syntax rules. Depending on the command, the syntax comprises the command itself, an option, and an argument. If you review the example above, this structure is reflected.

The best way to describe the command syntax is as follows:

- Command: What you want Linux to do
- Option: Modifies the command
- Argument: What the command acts on

For example, consider the analogy of a car. If you drive a car, drive would be the command. An option could be noted as turn on the headlights, and the argument is a car.

Again, to reinforce this concept, a car was used as an example:

- The command is drive.
- An option is to turn on the headlights.
- The argument is a car.

Reference: Image of Tux made by Larry Ewing (lewing@isc.tamu.edu) by using [The GIMP](#)



Useful commands

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This section describes some useful vocabulary that you can use at the Linux command prompt.

The `whoami` command

```
[c_salazar@hostname ~]$ whoami
```

```
c_salazar
```

```
[c_salazar@hostname ~]$
```

The concatenation of the strings *who*, *am*, and *i* is *whoami*. In Linux, it's used to show the current user's user name when the command is invoked. A use case for this command would be to use it after you log in as UserA. Then, switch users and run commands as another user to see the context.

The `id` command

```
[c_salazar@hostname ~]$ id
```

```
[userA@server00 ~]$ id  
uid=1003(userA) gid=1003(userA) groups=1003(userA) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023  
[userA@server00 ~]$
```

The `id` command is used to print real and effective user and group IDs.

This command helps identify the user and group name and numeric IDs (UID or group ID) of the current user or any other user on the server. This command displays the user and group information for each specified USER or (when USER is omitted) for the current user.

The hostname command

```
[c_salazar@hostname ~]$ hostname
```

```
[userA@server00 ~]$ hostname  
server00  
[userA@server00 ~]$ █
```

The `hostname` command is used to either set or display the current host, domain, or node name of the system.

This command displays the TCP/IP hostname. The `hostname` is used to either set or display the system's current host, domain, or node name. Many networking programs use hostnames to identify the system.

The uptime command

```
[c_salazar@hostname ~]$ uptime
```

```
[userA@server00 ~]$ uptime  
21:39:05 up 11 min.  2 users,  load average: 0.00, 0.12, 0.15  
[userA@server00 ~]$
```

The `uptime` command indicates how long the system has been up since the last boot.

This command indicates how long the system has been up since the last boot.

The date command

```
[c_salazar@hostname ~]$ date
```

```
[userA@server00 ~]$ date  
Thu Mar  7 21:37:34 GMT 2019  
[userA@server00 ~]$ █
```

The date command provides the current date and time.

This command can display the current time in a given format. It can also set the system date.

The cal command

```
[c_salazar@hostname ~]$ cal
```

```
[userA@server00 ~]$ cal
      March 2019
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
[userA@server00 ~]$
```

The `cal` command is used to display a simple calendar. If no arguments are specified, the current month is displayed.

The `cal` command is used to display a calendar. If no arguments are specified, the current month is displayed.

Note: The month can be specified as a number (1–12), a month name, or an abbreviated month name according to the current locales.

The `clear` command

```
[c_salazar@hostname ~]$ clear
```

```
[c_salazar@hostname ~]$
```

The `clear` command is a system command used to clear the terminal screen.

The `clear` command is used to clear the terminal screen. This command, when ran, clears all text on the terminal screen and displays a new prompt.

The echo command

```
[c_salazar@hostname ~]$ echo 'Hi!'
```

```
[c_salazar@hostname ~]$ Hi!
```

The `echo` command outputs the string *Hi!* within the quotation marks to the terminal screen.

```
[root@server00 ~]# echo 'Hello World!'  
Hello World!  
[root@server00 ~]#
```

The `echo` command places specified text on the screen. It is useful in scripts to provide users with information as the script runs. It directs selected text to standard output or in scripts to provide descriptions of displayed results.

The history command

```
[c_salazar@hostname ~]$ history
```

Bash keeps a history of each user's commands in a file in that user's home directory. The `history` command views the history file.

- History is stored in `/home/username/.bash_history`

```
142 whoami
143 hostname
144 clear
145 man date
146 man cal
147 man cat
148 cat -n /proc/partitions
149 history
[userA@server00 ~]$ !143
hostname
server00
[userA@server00 ~]$
```

! and the command number from the history file repeats a specific command

Displays the current user's history file

Bash keeps a history of each user's commands in a file in that user's home directory. The `history` command views the history file.

- It displays the current user's history file.
- Up and down arrow keys cycle through the output or the history file.
- This command can be run by using an event number: for example, `!143`.

Note: If you make a mistake when writing a command, don't reenter it. Use the `history` command to call the command back to the prompt, and then edit the command to correct the mistake.

You should use the `history` command in the following use cases:

- Accessing history between sessions
- Removing sensitive information from the history: for example, a password that is entered into a command argument

The touch command

```
[c_salazar@hostname ~]$ touch file_example_1
```

```
[userA@server00 Documents]$ ls  
[userA@server00 Documents]$ touch file1.txt  
[userA@server00 Documents]$ ls  
file1.txt  
[userA@server00 Documents]$ █
```

The **touch** command is used to update the access and modification times of each file to the current time.

The **touch** command can be used to create, change, or modify timestamps on existing files.

Also, the **touch** command is used to create a new empty file in a directory. To create a new file by using the **touch** command, enter *touch file_example_1*

You can use the **touch** command to generate more than one new file at a time, by writing the syntax *touch file_example_1 file_example_2 file_example_3*

Note: You see the **ls** command in the screenshot. You'll learn more about the **ls** command in the next lesson.

The **cat** command

```
[c_salazar@hostname ~]$ cat
```

```
[root@server00 etc]# cat hosts.allow
#
# hosts.allow This file contains access rules which are used to
#             allow or deny connections to network services that
#             either use the tcp_wrappers library or that have been
#             started through a tcp_wrappers-enabled xinetd.
#
#             See 'man 5 hosts_options' and 'man 5 hosts_access'
#             for information on rule syntax.
#             See 'man tcpd' for information on tcp_wrappers
#
[root@server00 etc]#
```

The **cat** (concatenate) command reads data from the file and provides the content as output within the terminal window.

The **cat** command is used to show contents of files. This command is useful to administrators because Linux configurations are held in text files. In the example that is shown, you can use this command to view the contents of the `hosts.allow` file.

Additional commands

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Standard streams are interconnected input and output communication channels. There are three input/output (I/O) streams. The standard streams are #0 Standard in (`stdin` command), #1 Standard out (`stdout` command), and #2 Standard error (`stderr` command). These communication channels handle incoming data or outgoing data from an input device or write data from applications. In modern operating systems, the standard error stream is redirected to log files.

Standard input – stdin command

Standard input (stdin)

Standard input is the device through which input is normally received: for example, a keyboard or a scanner.

Standard input: 0

0 is the value that is used for standard input.

< redirects the content of `myfirstscript` to the standard.



```
[c_salazar@hostname ~]$ cat 0<myfirstscript
```

Tells `cat` to take input information from `myfirstscript`

Standard input is the *file handle* that your process reads to get information from you. It can be from the user who provides the information or from a file.

Standard input: **0**

The example `cat 0<myfirstscript` tells `cat` to take input information from `myfirstscript`.

0 is the value that is used for standard input.

< redirects the content of `myfirstscript` to the standard.

Standard output – stdout command

Standard output (stdout)

Standard output is the device through which output is normally delivered: for example, the display monitor or a mobile device screen.

Standard output: 1

1 is the value that is used for standard output.



```
[c_salazar@hostname ~]$ ls -l 1>folder.txt
```

`ls -l` lists the content of the current folder. Instead of displaying it on the console, the output of `ls` is directed to the file `folder.txt`.

Standard output consists of the standardized streams of data that flow out of command line interface (CLI) commands. It simplifies the use of Linux on different devices and in different programs. It reduces the need to adjust output depending on the device that is used.

1 is the value that is used for standard output.

Consider the command: `ls -l 1>folder.txt`

`ls -l` lists the content of the current folder. Instead of displaying it on the console, the output of `ls` command is directed to the file `folder.txt`.

Standard error – stderr command

Standard error (stderr)

Standard error is where a process can write error messages.

Standard error: 2

2 is the value that is used for standard error.



```
[c_salazar@hostname ~]$ find / -name "*" -print 2> / dev/null
```

Note: You can discard any errors that the `find` command generates to keep your CLI tidy.

Standard error (stderr) is used to print the error messages on the output screen or window terminal.

2 is the value that is used for standard error.

For example, consider the command:

```
find / -name "*" -print 2> / dev/null
```

You can discard any errors that the `find` command generates to keep your CLI tidy.

Bash tab completion

```
[userA@server00 Documents]$ cat file  
file1 file2 file3 file4  
[userA@server00 Documents]$ cat file
```

Press **tab** twice
where you left off
to select the
specific file that
you want.

Pressing the tab key twice will show all matching options.

In Bash tab completion, the tab key automatically completes commands and file or directory names. The Bash tab saves time and provides greater accuracy. To use this feature, you must enter enough of the command or file name for it to be unique.

Best practice: History and tab completion



Get in the habit of using both of these features.

For history and Bash tab completion, best practices include the following:

- Develop the habit of using both of these features to make usage of command line faster.
- Use the features of Bash to work smarter, not harder.

Checkpoint questions



What are the three main components of the Bash shell command syntax?



How do you run a command from your Bash history?



What happens if you attempt tab completion on a command when you don't enter enough characters to make the command string unique?

Be sure that you can answer these key questions about this content.

1. Command, Option, Argument.
2. The history command. Example: `[c_salazar@hostname ~]$ history`
3. You will be shown all possible options if you press tab twice.

Key takeaways



- The Linux login workflow consists of the following three main steps:
 - The user is prompted to **authenticate** with a user name and password.
 - The user's session settings are loaded from the user's **profile** files.
 - The user is presented with a **command prompt** in the user's **home directory**.
- You can use the **tab** key to **automatically complete commands** at the command prompt.
- Some useful Linux commands include **history**, **whoami**, and **hostname**.

Some key takeaways from this lesson include the following:

- The Linux login workflow consists of the following three main steps:
 - The user is prompted to authenticate with a user name and password.
 - The user's session settings are loaded from the user's profile files.
 - The user is presented with a command prompt in the user's home directory.
- You can use the *tab* key to automatically complete commands at the command prompt.
- Some useful Linux commands include *history*, *whoami*, and *hostname*.



Thank you

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. Corrections, feedback, or other questions? Contact us at <https://support.aws.amazon.com/#/contacts/aws-training>. All trademarks are the property of their owners.

