

RHSA (Admin 1) Red Hat Enterprise Linux 8 Essentials

Prepared by Eng. Taghreed Elsayed

Lecture 2

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

Access the Command Line



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THE bash SHELL A command line is:

- ❑ a text-based interface which can be used to input instructions to a computer system. The Linux command line is provided by a program called the shell.
- ❑ Over the long history of UNIX-like systems, many shells have been developed. The default shell for users in Red Hat Enterprise Linux is the GNU Bourne-Again Shell (bash). Bash is an improved version of one of the most successful shells used on UNIX-like systems, the Bourne Shell (sh).
- ❑ This is called the shell prompt. When a regular user starts a shell, the default prompt ends with a \$ character.

```
[student@desktopX ~]$
```

- ❑ The \$ is replaced by a # if the shell is running as the super user, root. This makes it more obvious that it is a super user shell, which helps to avoid accidents and mistakes in the privileged account.

```
[root@desktopX ~]#
```

- ❑ # Using bash to execute commands can be powerful. The bash shell provides a scripting language that can support automation of tasks. The shell has additional capabilities that can simplify or make possible operations that are hard to accomplish efficiently with graphical tools.
- ❑ **VIRTUAL CONSOLES** Users access the bash shell through a terminal. A terminal provides a keyboard for user input and a display for output.
- ❑ On text-based installations, this can be the Linux machine's physical console, the hardware keyboard and display. Terminal access can also be configured through serial ports.
- ❑ Another way to access a shell is from a virtual console. A Linux machine's physical console supports multiple virtual consoles which act like separate terminals. Each virtual console supports an independent login session.

SHELL BASICS Commands entered at the shell prompt have three basic parts:

- Command to run
 - Options to adjust the behavior of the command
 - Arguments, which are typically targets of the command
- The command is the name of the program to run.
- It may be followed by one or more options, which adjust the behavior of the command or what it will do. Options normally start with one or two dashes (-a or --all, for example) to distinguish them from arguments.
 - Commands may also be followed by one or more arguments, which often indicate a target that the command should operate on. For example, the command line `usermod -L morgan` has a command (`usermod`), an option (`-L`), and an argument (`morgan`).

- The effect of this command is to lock the password on user morgan's account. To use a command effectively, a user needs to know what options and arguments it takes and in what order it expects them (the syntax of the command). Most commands have a --help option. This causes the command to print a description of what it does, a "usage statement" that describes the command's syntax, and a list of the options it accepts and what they do.
- Usage statements may seem complicated and difficult to read. They become much simpler to understand once a user becomes familiar with a few basic conventions: • Square brackets, [], surround optional items. • Anything followed by ... represents an arbitrary-length list of items of that type. • Multiple items separated by pipes, |, means only one of them can be specified. • Text in angle brackets, <>, represents variable data. For example, means "insert the filename you wish to use here". Sometimes these variables are simply written in capital letters (e.g., FILENAME). Consider the first usage statement for the date command:

```
[student@desktopX ~]$ date --help  
date [OPTION]... [+FORMAT]
```

- This indicates that date can take an optional list of options ([OPTION]...), followed by an optional format string, prefixed with a plus character, +, that defines how the current date should be displayed ([+FORMAT]). Since both of these are optional, date will work even if it is not given options or arguments (it will print the current date and time using its default format).

Access the Command Line

LOCAL CONSOLE ACCESS TERMS

DESCRIPTION	TERM
The interpreter that executes commands typed as strings.	Shell
The visual cue that indicates an interactive shell is waiting for the user to type a command.	Prompt
The name of a program to run.	Command
The part of the command line that adjusts the behavior of a command.	Option
The part of the command line that specifies the target that the command should operate on.	Argument
The hardware display and keyboard used to interact with a system.	Physical console
One of multiple logical consoles that can each support an independent login session.	Virtual console
An interface that provides a display for output and a keyboard for input to a shell session.	Terminal

EXECUTING COMMANDS USING THE BASH SHELL

- ❑ BASIC COMMAND SYNTAX The GNU Bourne-Again Shell (bash) is a program that interprets commands typed in by the user. Each string typed into the shell can have up to three parts:
- ❑ the command, options (that begin with a - or --), and arguments.
- ❑ Each word typed into the shell is separated from each other with spaces. Commands are the names of programs that are installed on the system. Each command has its own options and arguments.
- ❑ The Enter key is pressed when a user is ready to execute a command. Each command is typed on a separate line and the output from each command displays before the shell displays a prompt.
- ❑ If a user wants to type more than one command on a single line, a semicolon, ;, can be used as a command separator. A semicolon is a member of a class of characters called meta characters that has special meanings for bash.

- ❖ **EXAMPLES OF SIMPLE COMMANDS** The date command is used to display the current date and time. It can also be used by the super user to set the system clock. An argument that begins with a plus sign (+) specifies a format string for the date command.

```
[student@desktopX ~]$ date
Sat Apr  5 08:13:50 PDT 2014
[student@desktopX ~]$ date +%R
08:13
[student@desktopX ~]$ date +%x
04/05/2014
```

- ❖ The pass word command changes a user's own password. The original password for the account must be specified before a change will be allowed. By default, pass word is configured to require a strong password, consisting of lowercase letters, uppercase letters, numbers, and symbols, and is not based on a dictionary word. The super user can use the pass word command to change other users' passwords.

Access the Command Line

```
[student@desktopX ~]$ passwd
Changing password for user student.
Changing password for student.
(current) UNIX password: old_password
New password: new_password
Retype new password: new_password
```

Pass word: all authentication tokens updated successfully.

Linux does not require file name extensions to classify files by type. The file command scans the beginning of a file's contents and displays what type it is. The files to be classified are passed as arguments to the command.

Access the Command Line

```
[student@desktopX ~]$ file /etc/passwd
/etc/passwd: ASCII text
[student@desktopX ~]$ file /bin/passwd
/bin/passwd: setuid ELF 64-bit LSB shared object, x86-64, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=0x91a7160a019b7f5f754264d920e257522c5bce67, stripped
[student@desktopX ~]$ file /home
/home: directory
```

- ❑ The head and tail commands display the beginning and end of a file respectively. By default, these commands display 10 lines, but they both have a -n option that allows a different number of lines to be specified. The file to display is passed as an argument to these commands.

Access the Command Line

```
[student@desktopX ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
[student@desktopX ~]$ tail -n 3 /etc/passwd
gdm:x:42:42:./var/lib/gdm:/sbin/nologin
gnome-initial-setup:x:993:991:./run/gnome-initial-setup:/sbin/nologin
tcpdump:x:72:72:./:/sbin/nologin
```

The **wc** command counts lines, words, and characters in a file. It can take a **-l**, **-w**, or **-c** option to display only the lines, words, or characters, respectively.

```
[student@desktopX ~]$ wc /etc/passwd
 39   70 2005 /etc/passwd
[student@desktopX ~]$ wc -l /etc/passwd ; wc -l /etc/group
 39 /etc/passwd
 63 /etc/group
[student@desktopX ~]$ wc -c /etc/group /etc/hosts
843 /etc/group
227 /etc/hosts
1070 total
```

TAB COMPLETION

- ❑ Tab completion allows a user to quickly complete commands or file names once they have typed enough at the prompt to make it unique. If the characters typed are not unique, pressing the Tab key twice displays all commands that begin with the characters already typed.

```
[student@desktopX ~]$ pas<Tab><Tab>
passwd      paste      pasuspender
[student@desktopX ~]$ pass<Tab>
[student@desktopX ~]$ passwd
Changing password for user student.
Changing password for student.
(current) UNIX password:
```

Tab completion can be used to complete file names when typing them as arguments to commands. When **Tab** is pressed, it will complete the file name as much as it can. Pressing **Tab** a second time causes the shell to list all of the files that are matched by the current pattern. Type additional characters until the name is unique, then use tab completion to finish off the command line.

```
[student@desktopX ~]$ ls /etc/pas<Tab>[student@desktopX ~]$ ls /etc/passwd<Tab>
passwd      passwd-
```

Arguments and options can be matched with tab completion for many commands. The **useradd** command is used by the superuser, **root**, to create additional users on the system. It has many options that can be used to control how that command behaves. Tab completion following a partial option can be used to complete the option without a lot of typing.

Access the Command Line

```
[root@desktopX ~]# useradd --<Tab><Tab>
--base-dir      --groups      --no-log-init   --shell
--comment       --help        --non-unique    --skel
--create-home   --home-dir    --no-user-group --system
--defaults      --inactive    --password      --uid
--expiredate    --key         --root          --user-group
--gid           --no-create-home --selinux-user
[root@desktopX ~]# useradd --
```

COMMAND HISTORY

The **history** command displays a list of previously executed commands prefixed with a command number.

The exclamation point character, **!**, is a metacharacter that is used to expand previous commands without having to retype them. **!number** expands to the command matching the number specified. **!string** expands to the most recent command that begins with the string specified.

```
[student@desktopX ~]$ history
...Output omitted...
 23  clear
 24  who
 25  pwd
 26  ls /etc
 27  uptime
 28  ls -l
 29  date
 30  history
[student@desktopX ~]$ !ls
ls -l
total 0
drwxr-xr-x. 2 student student 6 Mar 29 21:16 Desktop
...Output omitted...
```

Access the Command Line

```
[student@desktopX ~]$ !26
ls /etc
abrt                hosts                pulse
adjtime             hosts.allow          purple
aliases             hosts.deny            qemu-ga
...Output omitted...
```

- ❑ The arrow keys can be used to navigate through previous command lines in the shell's history. Up Arrow edits the previous command in the history list. Down Arrow edits the next command in the history list. Use this key when the Up Arrow has been pressed too many times. Left Arrow and Right Arrow move the cursor left and right in the current command line being edited.
- ❑ The Esc+. key combination causes the shell to copy the last word of the previous command on the current command line where the cursor is. If used repeatedly, it will continue to go through earlier commands.

- ❑ **EDITING THE COMMAND LINE** When used interactively, bash has a command line-editing feature. This allows the user to use text editor commands to move around within and modify the current command being typed.
- ❑ Using the arrow keys to move within the current command and to step through the command history was introduced earlier in this session. More powerful editing commands are introduced in the following table.

Editing Command Line

Useful command line-editing shortcuts

SHORTCUT	DESCRIPTION
Ctrl+a	Jump to the beginning of the command line.
Ctrl+e	Jump to the end of the command line.
Ctrl+u	Clear from the cursor to the beginning of the command line.
Ctrl+k	Clear from the cursor to the end of the command line.
Ctrl+Left Arrow	Jump to the beginning of the previous word on the command line.
Ctrl+Right Arrow	Jump to the end of the next word on the command line.
Ctrl+r	Search the history list of commands for a pattern.

There are several other command line-editing commands available, but these are the most useful commands for beginning users. The other commands can be found in the **bash(1)** man page.



Chapter 02

BASH COMMANDS AND KEYBOARD SHORTCUTS

DESCRIPTION	SHELL COMMAND
Jump to the beginning of the previous word on the command line.	Ctrl+Left Arrow
Separate commands on the same line.	;
Clear from the cursor to the end of the command line.	Ctrl+k
Re-execute a recent command by matching the command name.	!<i>string</i>
Shortcut used to complete commands, file names, and options.	Tab
Re-execute a specific command in the history list.	!<i>number</i>
Jump to the beginning of the command line.	Ctrl+a
Display the list of previous commands.	history
Copy the last argument of previous commands.	Esc+.





جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

