

All that begins ...

السلام عليكم

peace be upon you

# Introduction to Linux

## CLI – Command Line Interface



**Abu Hasan 'ABDULLAH**

December 2019

## 1 Linux Command Shell

- ▶ What is a command shell?
- ▶ What is BASH?
- ▶ Accessing CLI via a Linux Console Terminal
- ▶ Accessing CLI via Graphical Terminal Emulation

## 2 Basic bash Shell Commands

- ▶ Using the Shell Prompt
- ▶ Navigating the Filesystem
- ▶ Traversing Directories
- ▶ Listing Files and Directories
- ▶ Filtering Listing Output
- ▶ Handling Files
- ▶ Viewing File Contents
- ▶ Managing Directories

## 3 Some Useful CLI Utilities

- ▶ Monitoring activities with `top` command
- ▶ Downloading files with `wget` command
- ▶ Multiplexing console with `screen` command

# Linux Command Shell

## What is a command shell?

- A program that interprets commands.
- A shell is **NOT** an operating system. It is a way to interface with the operating system and run commands.
- Allows a user to execute commands by typing them manually at a terminal, or automatically in programs called *shell scripts*.

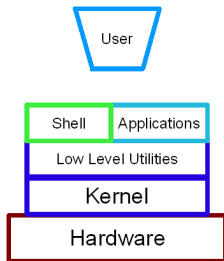


Figure 1: The different layers of the Linux operating system.\*

\* <http://www.penguintutor.com/linux/basic-shell-reference>

# Linux Command Shell

## What is BASH?

- BASH = **B**ourne **A**gain **S**hell or `bash`
- `/bin/bash` is a shell written as a free replacement to the standard Bourne Shell (`/bin/sh`) originally written by Steve Bourne for UNIX systems.
- It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.
- Since it is Free Software, it has been adopted as the default shell on most Linux systems, including **Ubuntu MATE 16.04 LTS** installed on **OSCAE.Initiative's** Linux workstations.

Table 1: Common Linux/Unix Command Shells

Name of Shell	Command Name	Description
Bourne shell	<code>/bin/sh</code>	The most basic shell available on all Linux/Unix systems.
Korn shell	<code>/bin/ksh</code>	Based on the Bourne shell with enhancements.
C shell	<code>/bin/csh</code>	Similar to the C programming language in syntax.
tcsh shell	<code>/bin/tcsh</code>	This is a different shell that emulates the C shell.
Bash shell	<code>/bin/bash</code>	Bourne Again Shell combines the advantages of the Korn Shell and the C Shell. The default on most Linux distributions.

# Linux Command Shell

## Accessing CLI via a Linux Console Terminal

- You can access one of the Linux virtual consoles using a simple keystroke combination by holding down the **Ctrl+Alt** key combination and then press a function key (**F1** through **F7**) for the virtual console you want to use.
- After logging into a virtual console, you are taken to the Linux CLI. Within the Linux virtual console, you do not have the ability to run any graphical programs.

```
Ubuntu 16.04.3 LTS belau tty2
belau login: theuser
Password:
Last login: Thu Oct 19 10:01:48 MYT 2017 on tty2
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-37-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/advantage

2 packages can be updated.
0 updates are security updates.

theuser@belau:~$ _
```

Figure 2: Linux virtual console login screen.

# Linux Command Shell

## Accessing CLI via Graphical Terminal Emulation

- From the menu system, click **Applications**, then select **System Tools**, and finally click **MATE Terminal**. Written in shorthand, the directions look like the following:  
**Applications** ▷ **System Tools** ▷ **MATE Terminal**

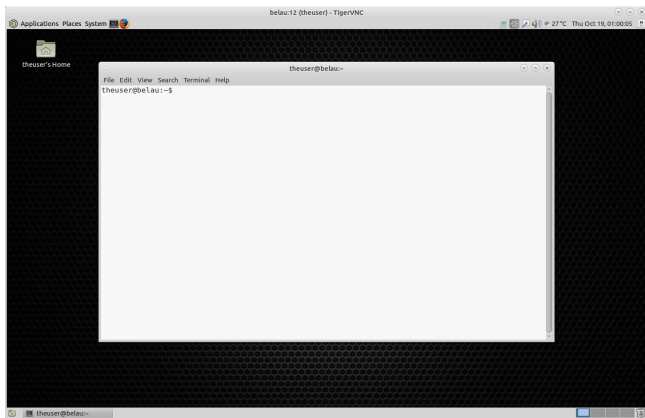


Figure 3: Linux CLI via MATE terminal.

# Basic bash Shell Commands

## Using the Shell Prompt

- After you log in to a Linux virtual console, Figure 2, or start a terminal emulation package, Figure 3, you get access to the shell CLI prompt.
- The prompt is your gateway to the shell.
- This is the place where you enter shell commands.
- The default prompt symbol for the bash shell is the dollar sign (\$). This symbol indicates that the shell is waiting for you to enter text.
- On Ubuntu Linux system, the shell prompt looks like this:

```
theuser@belau:~$
```



# Basic bash Shell Commands

## Navigating the Filesystem

- In Windows you may be used to seeing the file paths such as:

```
c:\Users\theuser\Documents\test.doc
```

**Note:** The Windows file path tells you exactly which physical disk partition contains the file named `test.doc`. If you saved `test.doc` on a flash drive, designated by the `J` drive, the file path would be `J:\test.doc`, indicating that the file is located at the root of the drive assigned the letter `J`.

- Linux, on the other hand, stores files within a **single directory structure**, called a **virtual directory**. In Linux, you will see file paths similar to the following:

```
/home/theuser/Documents/test.doc
```

**Note:** This indicates the file `test.doc` is in the directory `Documents`, under the directory `theuser`, which is contained in the directory `home`. Notice that the path doesn't provide any information as to which physical disk the file is stored on.

# Basic bash Shell Commands

## Navigating the Filesystem

Table 3: Common Linux Directory Names

Directory	Usage
/	root of the virtual directory, where normally, no files are placed
/bin	binary directory, where many GNU user-level utilities are stored
/boot	boot directory, where boot files are stored
/dev	device directory, where Linux creates device nodes
/etc	system configuration files directory
/home	home directory, where Linux creates user directories
/lib	library directory, where system and application library files are stored
/media	media directory, a common place for mount points used for removable media
/mnt	mount directory, another common place for mount points used for removable media
/opt	optional directory, often used to store third-party software packages and data files
/proc	process directory, where current hardware and process information is stored
/root	root home directory
/sbin	system binary directory, where many GNU admin-level utilities are stored
/run	run directory, where runtime data is held during system operation
/srv	service directory, where local services store their files
/sys	system directory, where system hardware information files are stored
/tmp	temporary directory, where temporary work files can be created and destroyed
/usr	user binary directory, where the bulk of GNU user-level utilities and data files are stored
/var	variable directory, for files that change frequently, such as log files

# Basic bash Shell Commands

## Traversing Directories: `cd` & `pwd` commands

- You use the change directory command (`cd`) to move your shell session to another directory in the Linux filesystem. To move to a specific location in the filesystem using the absolute directory reference, you just specify the full pathname in the `cd` command:

```
theuser@belau:~$ cd /usr/bin
theuser@belau:/usr/bin$
```

- The `pwd` command displays the shell session's current directory location, which is called the *present working directory*:

```
theuser@belau:/usr/bin$ pwd
/usr/bin
theuser@belau:/usr/bin$
```

# Basic bash Shell Commands

## Traversing Directories: `cd` & `pwd` commands

- You can move to any level within the entire Linux virtual directory structure from any level using the absolute directory reference:

```
theuser@belau:/usr/bin$ cd /var/log
theuser@belau:/var/log$
theuser@belau:/var/log$ pwd
/var/log
theuser@belau:/var/log$
```

- You can also quickly jump to your home directory from any level within the Linux virtual directory structure:

```
theuser@belau:/var/log$ cd
theuser@belau:~$
theuser@belau:~$ pwd
/home/theuser
theuser@belau:~$
```

# Basic bash Shell Commands

## Traversing Directories: `cd` & `pwd` commands

- A **relative directory reference** starts with either a directory name (if you're traversing to a directory under your current directory) or a special character. e.g. if you are in your home directory and want to move to your Documents subdirectory, you can use the `cd` command

```
theuser@belau:~$ pwd
/home/theuser
theuser@belau:~$
theuser@belau:~$ cd Documents
theuser@belau:~/Documents$ pwd
/home/theuser/Documents
theuser@belau:~/Documents$
```

- The two special characters used for relative directory references are:
  - ▶ The single dot (`.`) to represent the **current** directory
  - ▶ The double dot (`..`) to represent the **parent** directory

# Basic bash Shell Commands

## Traversing Directories: `cd` & `pwd` commands

- If you are in the `Documents` directory under your home directory and need to go to your `Downloads` directory, also under your home directory, you can do this:

```
theuser@belau:~/Documents$ pwd
/home/theuser/Documents
theuser@belau:~/Documents$ cd ../Downloads
theuser@belau:~/Downloads$ pwd
/home/theuser/Downloads
theuser@belau:~/Downloads$
```

- If you are in your home directory (`/home/theuser`) and want to go to the `/etc` directory, you could type the following:

```
theuser@belau:~$ cd ../../etc
theuser@belau:/etc$ pwd
/etc
theuser@belau:/etc$
```

# Basic bash Shell Commands

## Listing Files and Directories: `ls` command

- The `ls` command at its most basic form displays the files and directories located in your current directory:

```
$ ls
Arduino Documents draw Music Public Videos
Desktop Downloads gmon.out Pictures Templates
$
```

The `ls` command produces the listing in alphabetical order (in columns rather than rows).

- Use the `-F` parameter with the `ls` command to distinguish files from directories.

```
$ ls -F
Arduino/ Documents/ draw/ Music/ Public/ Videos/
Desktop/ Downloads/ gmon.out Pictures/ Templates/
$
```

# Basic bash Shell Commands

## Listing Files and Directories: `ls` command

- To display hidden files along with normal files and directories, use the `-a` parameter.

```
$ ls -a
.          .dbus      .gnome2    .mw        Videos
..         Desktop    .gvfs      Pictures    .vnc
Arduino    .dmrc      .ICEauthority .pki        .Xauthority
.bash_history Documents  .icons     .profile    .Xresources
.bash_logout Downloads  .java      Public      .xsession-errors
.bashrc    draw       .kshrc     .selected_editor
.cache     .flexlmrc .local     .ssh
.config    .gconf    .mozilla   Templates
.cxlayout.ini .gnome    Music      .themes
$
```

All the files beginning with a period, **hidden files**, are now shown.

- The `-R` parameter, called the recursive option, shows files that are contained within subdirectories in the current directory. Try it out:

```
$ ls -F -R
```

and see what happen!



# Basic bash Shell Commands

## Listing Files and Directories: `ls` command

- The `-l` parameter produces a long listing format, providing more information about each file in the directory:

```
$ ls -l
total 48
-rwx----- 1 theuser theuser  69 Oct 18 11:05 0-svnc
-rwx----- 1 theuser theuser  46 Oct 18 11:05 0-svncx
drwxrwxr-x  3 theuser theuser 4096 Jan 16  2017 Arduino
drwxr-xr-x  2 theuser theuser 4096 Oct 19 18:34 Desktop
drwxr-xr-x  3 theuser theuser 4096 Jan 11  2017 Documents
drwxr-xr-x  2 theuser theuser 4096 Jan 11  2017 Downloads
drwxrwxrwx  2 theuser theuser 4096 Jan 16  2017 draw
drwxr-xr-x  2 theuser theuser 4096 Jan 11  2017 Music
drwxr-xr-x  3 theuser theuser 4096 Jan 30  2017 Pictures
drwxr-xr-x  2 theuser theuser 4096 Jan 11  2017 Public
drwxr-xr-x  2 theuser theuser 4096 Jan 11  2017 Templates
drwxr-xr-x  2 theuser theuser 4096 Jan 11  2017 Videos
$
```

# Basic bash Shell Commands

## Listing Files and Directories: `ls` command

- In the long listing format, the first line shows the total number of blocks contained within the directory.
- Then each line contains the following information about each file (or directory):
  - ▶ The file type – such as directory (`d`), file (`-`), linked file (`l`), character device (`c`), or block device (`b`)
  - ▶ The file permissions
  - ▶ The number of file hard links
  - ▶ The file owner username
  - ▶ The file primary group name
  - ▶ The file byte size
  - ▶ The last time the file was modified
  - ▶ The filename or directory name

# Basic bash Shell Commands

## Filtering Listing Output: `ls` command

- The `ls` command also recognizes standard wildcard characters and uses them to match patterns within the filter:
  - ▶ A question mark (`?`) to represent one character
  - ▶ An asterisk (`*`) to represent any number of characters
- The question mark can be used to replace exactly one character anywhere in the filter string, e.g.

```
$ ls -l my_scr?pt
-rw-rw-r-- 1 theuser theuser 0 May 21 13:25 my_script
-rwxrw-r-- 1 theuser theuser 54 May 21 11:26 my_script
$
```

The filter `my_scr?pt` matched two files in the directory.

- Similarly, the asterisk can be used to match zero or more characters:

```
$ ls -l my*
-rw-rw-r-- 1 theuser theuser 0 May 21 13:25 my_file
-rw-rw-r-- 1 theuser theuser 0 May 21 13:25 my_script
-rwxrw-r-- 1 theuser theuser 54 May 21 11:26 my_script
$
```

- Using `*` and `?` in the filter is called *file globbing*. You can use more *metacharacter wildcards* for file globbing than just `*` and `?`.

# Basic bash Shell Commands

## Filtering Listing Output: ls command

- You can also use brackets:

```
$ ls -l my_scr[ai]pt
-rw-rw-r-- 1 theuser theuser 0 May 21 13:25 my_script
-rwxrw-r-- 1 theuser theuser 54 May 21 11:26 my_script
$
```

Here we used the brackets with two potential choices for a single character in that position, **a** or **i**.

- You can specify a range of characters, such as an alphabetic range **[a-i]**:

```
$ ls -l f[a-i]ll
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fall
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fell
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fill
$
```

- To specify what should not be included in the pattern match you use the exclamation point (!):

```
$ ls -l f[!a]ll
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fell
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fill
-rw-rw-r-- 1 theuser theuser 0 May 21 13:44 full
$
```

# Basic bash Shell Commands

Handling Files: `touch` command

- You can use the `touch` command to create an empty file:

```
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 theuser theuser 0 May 21 14:17 test_one
$
```

... or to change the modification time without changing the file contents:

```
$ ls -l test_one
-rw-rw-r-- 1 theuser theuser 0 May 21 14:17 test_one
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 theuser theuser 0 May 21 14:35 test_one
$
```

- You can also use the `touch` command to create multiple files at the same time:

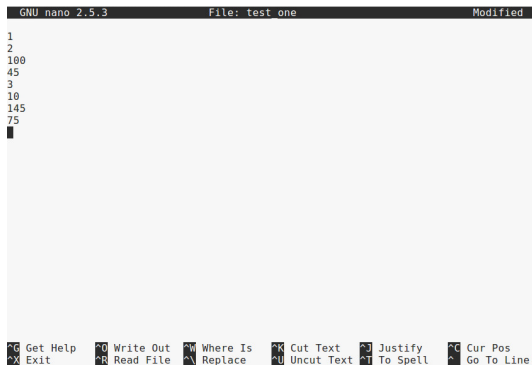
```
$ touch fall fell fill full
$ ls -l
total 0
-rw-rw-r-- 1 theuser theuser 0 Oct 20 19:20 fall
-rw-rw-r-- 1 theuser theuser 0 Oct 20 19:20 fell
-rw-rw-r-- 1 theuser theuser 0 Oct 20 19:20 fill
-rw-rw-r-- 1 theuser theuser 0 Oct 20 19:20 full
$
```

# Basic bash Shell Commands

Handling Files: `nano` text editor

- The `nano` text editor is installed on most Linux distributions by default. To open a file at the command line with `nano`:

```
$ nano test_one
```



```
GNU nano 2.5.3 File: test_one Modified
1
2
100
45
3
10
145
75
█

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

Figure 4: The `nano` editor window.

# Basic bash Shell Commands

Handling Files: [nano](#) text editor

Table 5: nano Control Commands

Command	Description
CTRL+C	Displays the cursor's position within the text editing buffer
CTRL+G	Displays <a href="#">nano</a> 's main help window
CTRL+J	Justifies the current text paragraph
CTRL+K	Cuts the text line and stores it in cut buffer
CTRL+O	Writes out the current text editing buffer to a file
CTRL+R	Reads a file into the current text editing buffer
CTRL+T	Starts the available spell checker
CTRL+U	Pastes text stored in cut buffer and places in current line
CTRL+V	Scrolls text editing buffer to next page
CTRL+W	Searches for word or phrases within text editing buffer
CTRL+X	Closes the current text editing buffer, exits nano, and returns to the shell
CTRL+Y	Scrolls text editing buffer to previous page

# Basic bash Shell Commands

## Handling Files: `cp` command

- The `cp` command is used to copy files and directories from one location in the filesystem to another and needs at least two parameters – **source** and **destination**:  
`cp source destination`
- When both the source and destination parameters are filenames, the `cp` command copies the source file to a new destination file. The new file acts like a brand new file, with an updated modification time:

```
$ cp test_one test_two
$ ls -l test_*
-rw-rw-r-- 1 theuser theuser 0 May 21 14:35 test_one
-rw-rw-r-- 1 theuser theuser 0 May 21 15:15 test_two
$
```

If the destination file already exists, the `cp` command may not prompt you to this fact. It is best to add the `-i` option to force the shell to ask whether you want to overwrite a file:

```
$ ls -l test_*
-rw-rw-r-- 1 theuser theuser 0 May 21 14:35 test_one
-rw-rw-r-- 1 theuser theuser 0 May 21 15:15 test_two
$
$ cp -i test_one test_two
cp: overwrite 'test_two'? n
$
```

If you don't answer **y**, the file copy does not proceed.



# Basic bash Shell Commands

## Handling Files: cp command

- To copy a file into a pre-existing directory:

```
$ cp -i test_one /home/theuser/Documents/  
$  
$ ls -l /home/theuser/Documents  
total 0  
-rw-rw-r-- 1 theuser theuser 0 May 21 15:25 test_one  
$
```

The new file is now under the Documents subdirectory, using the same filename as the original.

- You can also use a relative directory reference:

```
$ cp -i test_one Documents/  
cp: overwrite 'Documents/test_one'? y  
$  
$ ls -l Documents  
total 0  
-rw-rw-r-- 1 theuser theuser 0 May 21 15:28 test_one  
$
```

# Basic bash Shell Commands

## Handling Files: `cp` command

- Single dot (`.`) represents your present working directory. If you need to copy a file with a long source object name to your present working directory, `.` can simplify the task:

```
$ cp -i /etc/NetworkManager/NetworkManager.conf .
$
$ ls -l NetworkManager.conf
-rw-r--r-- 1 theuser theuser 76 May 21 15:55 NetworkManager.conf
$
```

- You can also use wildcard metacharacters in your `cp` command:

```
$ cp *script Mod_Scripts/
$ ls -l Mod_Scripts
total 26
-rwxrw-r-- 1 theuser theuser 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 theuser theuser  54 May 21 16:27 my_script
-rwxrw-r-- 1 theuser theuser 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 theuser theuser 243 May 21 16:16 SUID_search.sh
$
```

This command copied any files that ended with `script` to directory `Mod_Scripts`.

# Basic bash Shell Commands

## Handling Files: `mv` command

- In Linux, renaming files is called *moving files*. The `mv` command is available to move both files and directories to another location or a new name:

```
$ ls -li f?ll
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fall
296717 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 full
$
$ mv fall fzll
$
$ ls -li f?ll
296717 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 full
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fzll
$
```

Notice that moving the file changed the name from `fall` to `fzll`, but it kept the same inode number and timestamp value. This is because `mv` affects only a file's name.

# Basic bash Shell Commands

Handling Files: `mv` command

- You can use `mv` to change a file's location:

```
$ ls -li /home/theuser/fzll
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44
/home/theuser/fzll
$
$ ls -li /home/theuser/Pictures/
total 0
$ mv fzll Pictures/
$
$ ls -li /home/theuser/Pictures/
total 0
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44 fzll
$
$ ls -li /home/theuser/fzll
ls: cannot access /home/theuser/fzll: No such file or directory
$
```

Here we used `mv` command to move the file `fzll` from `/home/theuser` to `/home/theuser/Pictures`.

# Basic bash Shell Commands

## Handling Files: `mv` command

- You can also use the `mv` command to move a file's location and rename it, all in one easy step:

```
$ ls -li Pictures/fzll
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44
Pictures/fzll
$
$ mv /home/theuser/Pictures/fzll /home/theuser/fall
$
$ ls -li /home/theuser/fall
296730 -rw-rw-r-- 1 theuser theuser 0 May 21 13:44
/home/theuser/fall
$
$ ls -li /home/theuser/Pictures/fzll
ls: cannot access /home/theuser/Pictures/fzll:
No such file or directory
```

# Basic bash Shell Commands

## Handling Files: `rm` command

- In Linux, deleting is called *removing*. The command to remove files in the bash shell is `rm`. The basic form of the `rm` command is simple:

```
$ rm -i fall
rm: remove regular empty file 'fall'? y
$
$ ls -l fall
ls: cannot access fall: No such file or directory
$
```

- You can also use wildcard metacharacters to remove groups of files. However, again, use that `-i` option to protect yourself:

```
$ rm -i f?ll
rm: remove regular empty file 'fell'? y
rm: remove regular empty file 'fill'? y
rm: remove regular empty file 'full'? y
$
$ ls -l f?ll
ls: cannot access f?ll: No such file or directory
$
```

# Basic bash Shell Commands

Viewing File Contents: `file` and `cat` commands

- The `file` command allows you to determine what kind of file it is:

```
$ file my_file
my_file: ASCII text
$
```

OR

```
$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=d0bc0fb9b3f60f72bbad3c5a1d24c9e2a1fde775, stripped
$
```

- The `cat` command is a tool for displaying all the data inside a text file:

```
$ cat test1
Salam.
This is a test file.
It is used to test the cat command.
$
```

or try

```
$ cat /etc/bash.bashrc
```

and see what happen ...

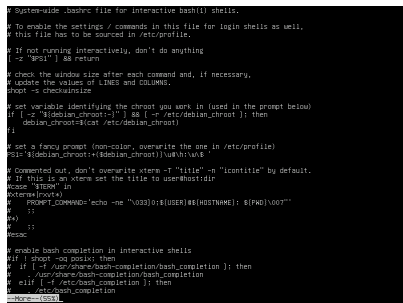
# Basic bash Shell Commands

Viewing File Contents: [more command](#)

- [more](#) is a command to view (but not modify) the contents of a text file one screen at a time. You can type

```
$ more /etc/bash.bashrc
```

to produce the sample [more](#) screen shown in Figure 5 ...



```
System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, overwrite the one in /etc/profile)
PS1='${debian_chroot:+($debian_chroot)}$# ?'

# Commented out, don't overwrite xterm -F "title" -n "icontitle" by default.
# If this is an xterm set the title to user@host:dir
#case "$TERM" in
#xterm*)
#    PROMPT_COMMAND='echo -ne "\033[0;${USER}@${HOSTNAME}: ${PWD}\007"'
#    ;;
#*)
#    ;;
#esac

# enable bash completion in interactive shells
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

==More==(55%)
```

Figure 5: The [more](#) command output.

... showing that you're still in the [more](#) application and how far along (55%) in the text file you are. Press SPACEBAR to view more of the file content.



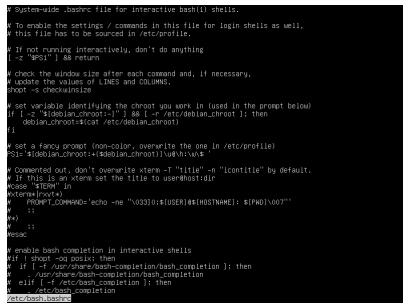
# Basic bash Shell Commands

## Viewing File Contents: `less` command

- `less` is similar to `more`, but has the extended capability of allowing both forward and backward navigation through the file:

```
$ less /etc/bash.bashrc
```

to produce the sample `less` screen shown in Figure 6 ...

A screenshot of a terminal window showing the output of the 'less' command applied to the file '/etc/bash.bashrc'. The text is displayed in a monospaced font with syntax highlighting. The content includes comments about enabling settings for login shells, window size checks, setting the chroot environment, and enabling bash completion. The prompt '\$' is visible at the bottom of the terminal window.

```
System-wide .bashrc file for interactive bash() shells.
# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.
# If not running interactively, don't do anything
[ -z "$PS1" ] && return
# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize
# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "$debian_chroot" ] && [ -n "/etc/debian_chroot" ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi
# set a fancy prompt (non-color, overwrite the one in /etc/profile)
PS1="debian_chroot:+${debian_chroot}|u@h:~$ "
# Commented out, don't overwrite xterm -T "title" -n "lcontitle" by default.
# If this is an xterm set the title to user@host:dir
unset $TERM" in
xterm)react)
# PROMPT_COMMAND="echo -ne "\033[0;${USER}@${HOSTNAME}: ${PWD}\007""
#
#
#
#
#
#
# enable bash completion in interactive shells
if ! shopt -oq posix; then
# if [ -f /usr/share/bash-completion/bash_completion ]; then
#     . /usr/share/bash-completion/bash_completion
# elif [ -f /etc/bash_completion ]; then
#     . /etc/bash_completion
# fi
/etc/bash.bashrc
```

Figure 6: The `less` command output.

Press **ARROW UP** or **ARROW DOWN** to navigate the file content.

# Basic bash Shell Commands

## Managing Directories: `mkdir` command

- To create a new directory in Linux, use the `mkdir` command:

```
$ mkdir New_Dir
$ ls -ld New_Dir
drwxrwxr-x 2 theuser theuser 4096 May 22 09:48 New_Dir
$
```

Notice in the new directory's long listing that the directory's record begins with a `d`.

- To create several directories and subdirectories at the same time, you need to add the `-p` parameter:

```
$ mkdir -p New_Dir/Sub_Dir/Under_Dir
$
$ ls -R New_Dir
New_Dir:
Sub_Dir
New_Dir/Sub_Dir:
Under_Dir
New_Dir/Sub_Dir/Under_Dir:
$
```

# Basic bash Shell Commands

## Managing Directories: `rmdir` command

- The basic command for removing a directory is `rmdir`:

```
$ touch New_Dir/my_file
$ ls -li New_Dir/
total 0
294561 -rw-rw-r-- 1 theuser theuser 0 May 22 09:52 my_file
$
$ rmdir New_Dir
rmdir: failed to remove 'New_Dir': Directory not empty
$
```

By default, the `rmdir` command works only for removing empty directories. Because we created a file, `my_file`, in the `New_Dir` directory, the `rmdir` command refuses to remove it.

- To fix this, we must remove the file first. Then we can use the `rmdir` command on the now empty directory.

```
$ rm -i New_Dir/my_file
rm: remove regular empty file 'New_Dir/my_file'? y
$
$ rmdir New_Dir
$
$ ls -ld New_Dir
ls: cannot access New_Dir: No such file or directory
```

# Basic bash Shell Commands

## Managing Directories: `tree` command

- Before executing the `rmdir` command, you might want to check with the `tree` command which allows you to see directories, subdirectories and files in every level of your directory tree structure:

```
$ tree
```

The output should be similar to the one shown in Figure 7.

```
├── 0-svnc
├── 0-svncx
├── Arduino
│   ├── libraries
│   └── readme.txt
├── Desktop
├── Documents
│   └── MATLAB
├── Downloads
├── draw
│   └── VariCAD Samples -> /opt/VariCAD/lib/sample/
├── fall
├── fell
├── fill
├── full
├── Music
├── Pictures
│   └── smplayer_screenshots
├── Public
├── Templates
├── test_one.save
└── Videos

14 directories, 8 files
```

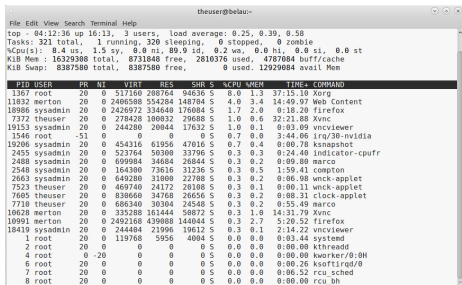
Figure 7: The `tree` command output.

# Some Useful CLI Utilities

## Monitoring activities with `top` command

- `top` command displays processor activity of your Linux box and also displays tasks managed by kernel in real-time.

```
$ top
```



```
theuser@belau:~$ top
top - 04:12:36 up 16:13, 3 users, load average: 0.25, 0.39, 0.58
Tasks: 321 total, 1 running, 320 sleeping, 0 stopped, 0 zombie
%Cpu(s):  8.4 us,  1.5 sy,  0.0 ni, 89.9 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16329360 total, 8731848 free, 2810376 used, 4787084 buff/cache
KiB Swap: 8387580 total, 8387580 free,  0 used, 12929884 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  %CPU  %MEM     TIME+ COMMAND
 1367 root        20   0 517160 208764 94636  0.0  1.3 37:15.10 Xorg
11832 nerton     20   0 2480598 554284 148794  4.0  3.4 14:49.97 Web Content
18986 sysadmin  20   0 2426972 334640 176084  1.7  2.0  0:18.20 firefox
 3732 theuser   20   0 278428 100032 29688  1.0  0.6 32:21.88 Xvnc
19153 sysadmin  20   0 244280 20044 17632  1.0  0.1  0:03.09 vncviewer
 1546 root       51   0  0 0 0  0.7  0.0  3:44.06 irq/20-nvidia
19206 sysadmin  20   0 454316 61956 47016  0.7  0.4  0:00.78 ksmaphot
2455 sysadmin  20   0 523764 50300 33796  0.3  0.3  0:24.40 indicator-cpufr
2488 sysadmin  20   0 609984 34684 26844  0.3  0.2  0:00.80 marco
2548 sysadmin  20   0 164300 73616 31236  0.3  0.5  1:59.41 compton
2063 sysadmin  20   0 649288 31800 22708  0.3  0.2  0:06.98 wnck-applet
 7523 theuser   20   0 469740 24172 20108  0.3  0.1  0:00.11 wnck-applet
 7605 theuser   20   0 838660 34768 26656  0.3  0.2  0:08.31 clock-applet
 7710 theuser   20   0 608340 30304 24548  0.3  0.2  0:55.49 marco
10628 nerton     20   0 335288 161444 58872  0.3  1.0 14:31.79 Xvnc
10991 nerton     20   0 2492168 439088 144044  0.3  2.7  5:20.52 firefox
18419 sysadmin  20   0 244484 21996 19612  0.3  0.1  2:14.22 vncviewer
  1 root       20   0 119768 5956 4004  0.0  0.0  0:03.44 systemd
  2 root       20   0  0 0 0  0.0  0.0  0:00.00 kthreadd
  4 root       20  -20  0 0 0  0.0  0.0  0:00.00 kworker/0:0H
  6 root       20   0  0 0 0  0.0  0.0  0:00.26 ksoftirqd/0
  7 root       20   0  0 0 0  0.0  0.0  0:06.52 rcu_sched
  8 root       20   0  0 0 0  0.0  0.0  0:00.00 rcu_bh
```

Figure 8: Linux's `top` command output.

- It shows processor and memory are being used and other information like running processes.

# Some Useful CLI Utilities

## Downloading files with `wget` command

- `wget` utility retrieves files from **World Wide Web** (WWW) using widely used protocols like `HTTP`, `HTTPS` and `FTP`.
- It shows download progress, size, date and time while downloading.
- To download single file and stores in a current directory:

```
$ wget https://winscp.net/download/WinSCP-5.11.2-Setup.exe
```

- To resume uncompleted download

```
$ wget -c http://mirrors.ctan.org/systems/texlive/Images/texlive2017-20170524.iso
```

It's good practice to add `-c` switch when you download big files because if the download happens to stop, we can resume download the same file where it was left off.

- If **URL** of the file you want to download contains “funny” characters, wrap it in quotes to avoid errors and save it into a different file name using the `-O` switch:

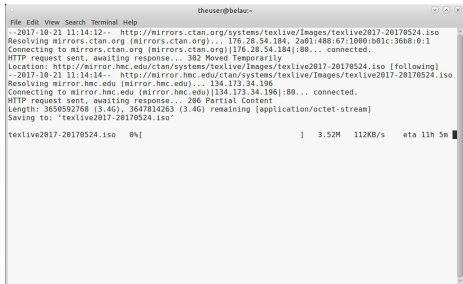
```
$ wget "http://search.yahoo.com/404handler?src=search&p=food+delicious" -O test.html
```

# Some Useful CLI Utilities

## Multiplexing console with `screen` command

- `screen` can be used to multiplex a physical console between several processes (typically interactive shells). Let's download a big file with:

```
$ screen wget -c "http://mirrors.ctan.org/systems/texlive/Images/texlive2017-20170524.iso"
```



```
theuser@belac:~  
--2017-10-21 11:14:12-- http://mirrors.ctan.org/systems/texlive/Images/texlive2017-20170524.iso  
Resolving mirrors.ctan.org (mirrors.ctan.org)... 176.28.54.184, 2a01:488:67:1080:b01c:36b8:0:1  
Connecting to mirrors.ctan.org (mirrors.ctan.org)[176.28.54.184]:80... connected.  
HTTP request sent, awaiting response... 302 Moved Temporarily  
Location: http://mirror.hmc.edu/ctan/systems/texlive/Images/texlive2017-20170524.iso [following]  
--2017-10-21 11:14:14-- http://mirror.hmc.edu/ctan/systems/texlive/Images/texlive2017-20170524.iso  
Resolving mirror.hmc.edu (mirror.hmc.edu)... 134.173.34.196  
Connecting to mirror.hmc.edu (mirror.hmc.edu)[134.173.34.196]:80... connected.  
HTTP request sent, awaiting response... 286 Partial Content  
Length: 3650592768 (3.4G), 3647814263 (3.4G) remaining [application/octet-stream]  
Saving to: 'texlive2017-20170524.iso'  
  
texlive2017-20170524.iso 0%| ] 3.52M 112KB/s eta 11h 5m
```

Figure 9: screening a big file download.

- One of the advantages of `screen` is that you can detach it. Later, you can restore it without losing anything you have done on the screen.

# Some Useful CLI Utilities

## Multiplexing console with `screen` command

- While downloading in progress, Figure 9, you can press `Ctrl-A` and `d` to detach it.

```
[detached from 3924.pts-6.belau]
$
```

- Couple of hours later, you start `ssh` again to your server and you want to see the progress of your download process. To do that, you need to restore the screen. You can run these commands:

```
$ screen -ls
There are screens on:
  3924.pts-6.belau (10/21/2017 11:25:42 AM) (Detached)
  3277.pts-6.belau (10/21/2017 10:55:11 AM) (Detached)
2 Sockets in /var/run/screen/S-theuser.
$
$ screen -R 3924.pts-6.belau
```



# Bibliography

- 1 UBUNTU MATE TEAM (2016): *Ubuntu MATE User Guide*, Creative Commons Attribution 4.0
- 2 RICHARD BLUM & CHRISTINE BRESNAHAN (2015): *Linux Command Line and Shell Scripting Bible, 3ed*, Wiley (ISBN: 978-1-118-98419-2 (ebk))
- 3 WILLIAM E. SHOTTS, JR. (2013): *The Linux Command Line, 2ed*, Copyright © 2008–2013 William E. Shotts, Jr.
- 4 VICTOR GEDRIS (2003): *An Introduction to the Linux Command Shell For Beginners, v1.2*, Copyright © 2003 Victor Gedris

... must end

- ... and I end my presentation with two supplications

رَبِّ زِدْنِي عِلْمًا

my Lord! increase me in knowledge

(TAA-HAA (20):114)

اللَّهُمَّ إِنَّا نَسْأَلُكَ عِلْمًا نَافِعًا

O Allah! We ask You for knowledge that is of benefit

(IBN MAJAH)