

# LINUX FUNDAMENTALS



**Emanuel Shmuelov**

## **Linux Fundamentals**

2<sup>rd</sup> Edition, Nov 2010

Author: *Emanuel Shmuelov*

Cover Image & Design: *Eshed Yam/Dprint*

***Copyright © 2010. All rights reserved.***

*No part of this work may be used, accessed, reproduced, transmitted or distributed in any form or by any means, electronic or mechanic, stored in a database or any retrieval system, without the prior written permission.*

### **Legal Disclaimer**

*This book is intended to provide fundamental information about Linux operating system. Although we endeavor to keep this book as up-to-date as possible, we appreciate readers to share with us the concern for accuracy in the text and examples.*

## TABLE OF CONTENT

<b>1.</b>	<b>GENERAL NOTES.....</b>	<b>5</b>
<b>2.</b>	<b>LINUX BASICS .....</b>	<b>7</b>
2.1.	INTRODUCTION.....	7
2.2.	LOGGING IN AND OUT .....	8
<b>3.</b>	<b>THE FILE SYSTEM .....</b>	<b>11</b>
3.1.	THE KERNEL .....	11
3.2.	FILESYSTEM CONCEPTS .....	12
3.3.	NAVIGATING THE FILE SYSTEM TREE .....	13
3.4.	FILE TYPES, NAMING & DIRECTORY STRUCTURE .....	14
3.5.	MANIPULATING FILES AND DIRECTORIES.....	17
<b>4.</b>	<b>THE SHELL.....</b>	<b>21</b>
4.1.	WHAT IS A SHELL?.....	21
4.2.	SHELL AS A COMMAND LINE INTERPRETER .....	21
4.3.	COMMANDS AND ARGUMENTS .....	23
4.4.	WILDCARDS .....	24
4.5.	QUOTES .....	25
4.6.	GETTING HELP .....	26
<b>5.</b>	<b>THE SHELL ENVIRONMENT .....</b>	<b>29</b>
5.1.	SHELL VARIABLES.....	29
5.2.	THE SEARCH PATH .....	32
5.3.	SHELL ALIASES .....	32
5.4.	ENVIRONMENT CONFIGURATION FILES.....	33
5.5.	CONFIGURING SHELL PROMPT .....	33
<b>6.</b>	<b>COMMAND INPUT AND OUTPUT.....</b>	<b>37</b>
<b>7.</b>	<b>PROCESSES .....</b>	<b>41</b>
7.1.	PROGRAMS AND PROCESSES .....	41
7.2.	JOB CONTROL AND SIGNALS .....	44
7.3.	EXIT STATUS.....	47
<b>8.</b>	<b>ORGANIZING FILES .....</b>	<b>49</b>
8.1.	FILE OWNERSHIP & PERMISSION.....	49
8.2.	HARD AND SOFT LINKS .....	52
<b>9.</b>	<b>POWER TOOLS.....</b>	<b>55</b>

9.1.	USING DIFF, FIND AND WHICH.....	55
9.2.	USING <b>grep</b> .....	57
9.3.	REGULAR EXPRESSIONS.....	57
<b>10.</b>	<b>THE VI TEXT EDITOR .....</b>	<b>63</b>
10.1.	CONCEPTS OF THE VI EDITOR.....	63
10.2.	USING VI.....	63
<b>11.</b>	<b>WRITING SHELL SCRIPTS .....</b>	<b>67</b>
11.1.	SIMPLE SCRIPTING .....	67
<b>12.</b>	<b>BASIC COMMUNICATIONS.....</b>	<b>69</b>
12.1.	LINUX IN A NETWORKED ENVIRONMENT.....	69
12.2.	NETWORKING UTILITIES .....	70
<b>13.</b>	<b>LOOKING AFTER YOUR SYSTEM .....</b>	<b>73</b>
13.1.	STARTING AND STOPPING THE SYSTEM.....	73
13.2.	MANAGING SERVICES .....	74
13.3.	USER MANIPULATION.....	76
13.4.	OTHER OPERATIONS .....	78
<b>14.</b>	<b>APPENDIX I - SHELL COMMANDS EXAMPLES .....</b>	<b>79</b>
<b>15.</b>	<b>APPENDIX II – COMMON LINUX COMMANDS .....</b>	<b>83</b>

## 1. General Notes

- ✓ Linux operating system supports two working modes: GUI (Graphical User Interface) and Text based. This book focuses on the text based mode.
- ✓ Linux operating system supports more than one way to execute many requested operations. This book describes the most common ways used to execute operations/requests.
- ✓ The shell used at the examples of this book is BASH.
- ✓ Lines starting with “[user@localhost ~]#” meaning command line prompt.
- ✓ At the command line explanation, content within [] square parenthesis indicates optional, and content within <> indicates mandatory.
- ✓ The Linux operating system provides a long list of command line utilities. Each utility usually supports many features, hence provides a range of properties. Prior to using a command line utility, read its manual by typing:

```
[user@localhost ~]# man <command name>
```



## 2. Linux Basics

### 2.1. Introduction

At the beginning of the 1990s, home PCs began to be powerful enough to run UNIX operating system, unfortunately, UNIX was not freely distributed.

Linux, a UNIX-like operating system, was introduced to the world on September 1991, developed by Linus Torvalds, a young computer science student from the university of Helsinki, who thought it would be a good idea to have a free sort of UNIX operating system for home users, as he said:

"This isn't yet the 'mother of all operating systems', and anyone who hoped for that will have to wait for the first real release (1.0), and even then you might not want to change from minix. This is a source release for those that are interested in seeing what Linux looks like, and it's not really supported yet."

On March 31<sup>st</sup> 1994 the first release 1.0 was introduced, supporting 32bit i386 architecture. A year later, 32-bit MIPS, 32-bit SPARC, and the 64-bit Alpha were supported too. Jumping forward to the Linux 2.6 kernel, first released in 2004, it has been and continues to be ported to numerous additional architectures.

The rest is history...

Exposing to Linux operating system, here are several common terms you will keep facing:

**Open Source** - the open-source philosophy permits users to study, change, and improve the software by providing the source code of the developed software. The main advantages behind it are: enabling unlimited tuning and improvement of a software product, making it possible to port the code to new hardware and extending the lifetime of the application. In fact, no binary-only application survive more than 10 years in unmodified form, while several open source software systems from the 1980s are still in widespread use.

**GNU** - a recursive acronym that stands for "GNU's Not UNIX".

**GPL** - General Public License is the most widely used free software license model.

**POSIX** - Portable Operating System Interface [for UNIX], is the standard for UNIX operating systems.

**FreeBSD** - Berkeley Software Distribution, a free UNIX-like operating system descended from AT&T UNIX.

Since 1991, many distributions of UNIX like operating system, based on Linux kernel, were released. Below you can find names and logos of most of the distributors:



## 2.2. Logging in and out

Linux is a multitasking and multiuser operating system, enabling two basic modes for working with its interface: GUI mode and DOS-like text based mode.

It is possible to work with both modes while working directly with the PC or by remote connection (requires enabling some services for availability).

There are several graphical interfaces (desktops) supported by the Linux operating system, two most famous are: KDE and Gnome. Both can be installed and activated by setting user configuration.

There is a system built-in super user called “root”, allowed to run any setup, update or configuration change within the system.



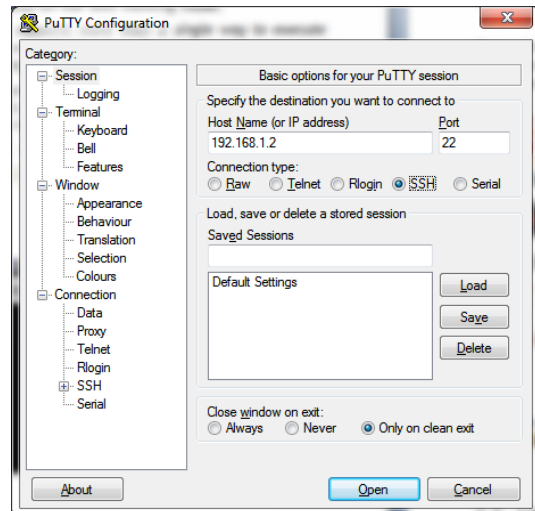
**Tip:** Do not login as “root” user unless you need to run setup or configuration updates.

In order to start working with the operating system, you must have username and password (refer to chapter 13.2 for user management) to log in into the system.

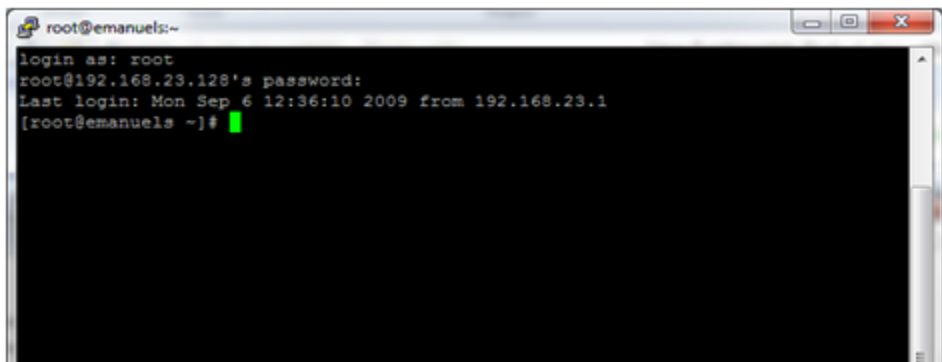


In order to login, either use the PC terminal or a remote connection application like PuTTY (requires Linux to be connected to the network) to remotely connect to the terminal.

Select connection type (SSH, Telnet etc.) and provide host's IP (ask the network system administrator for the login credentials):



Once connected, a prompt window will ask for username and password credentials. Once logged in, the terminal window will show a command prompt, for example:



**Congratulations!** You are now logged-in.

In order to logout from a working session follow one of these options:

1. Type "logout" or "exit" and press enter
2. Press the keyboard sequence "Ctrl+d"



## 3. The File System

### 3.1. The kernel

The kernel is the central component of an operating system, acting as a bridge between the applications and the actual data processing done at the hardware level.

There are different kernels used by different operating systems and different hardware types (i.e.: PC with a single CPU needs a different kernel type than PC with more than one CPU).



In order to verify the kernel's Version, Type and Architecture currently installed on the system, run the following command:

```
[user@localhost ~]# uname -pro  
2.6.9-78.ELsmp i686 GNU/Linux
```

**i686** - Indicates the architecture of the CPU. There are different kernels compilations for different CPU architectures, hence notice which kernel is needed for your machine.

There are 3 different kernel types for the Linux operating system:

**Standard Kernel** - Generic Linux kernel.

**Kernel SMP** - Symmetric Multiprocessing version, compiled for SMP machines

**Kernel Hugemem** - Dedicated version for machines with more than 4 Gigabyte of physical memory.

Like the “blue screen” for Windows, Linux kernel also might hang due to unrecoverable system error detected by the kernel (contrary to similar errors detected by user space code). This situation is called “**kernel panic**” and usually demands system reboot.

### 3.2. Filesystem concepts

In Linux filesystem everything is either file or process - this describes Linux’s filesystem the best, since a directory is actually a file listing the names of other files, devices are represented by files enabling communication with the device and so on.

Linux supports numerous file system types; most common are:

*Ext2*: This is like UNIX file system. It has the concepts of blocks, inodes<sup>1</sup> and directories.

*Ext3*: Enhanced ext2 filesystem with journaling capabilities (journaling allows fast file system recovery).

*Swap*: Special filesystem type reserved for paging data with the machine’s physical memory.

*Isofs*: Used by CDRom file system.

*Sysfs*: RAM-based filesystem initially based on ramfs. It is used to export kernel objects so that the end-user can use it easily.

*usbfs*: Supporting USB mass storage devices connected to the machine.

*Procfs*: The proc file system acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime using sysctl command. For example, you can find out CPU info using following command:

```
[user@localhost ~]# cat /proc/cpuinfo
```

*NFS*: Network File System allows users or systems to share the same files by using a client/server methodology.

---

<sup>1</sup> An inode stores basic information about a regular file, directory, or other file system object

In addition to the list above, Linux supports Microsoft NTFS, vfat and many other file systems.

You can find out what type of file systems is currently mounted using “mount” command:

```
[user@localhost ~]# mount
```



**Note:** In order to mount new devices you need to login as *root*.

In order to find out the filesystem’s **disk space usage** use:

```
df - report filesystem disk space usage
```

Common options	Description
-h	Print sizes in human readable format
-a	Include all filesystems
-l	Limit listing to local filesystems

Example:

```
[user@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol100
                  4.8G  4.1G  528M  89% /
/dev/sda1        99M   14M   81M  14% /boot
none            125M    0  125M   0% /dev/shm
```

In order to find out the **directory space usage** use:

```
du - estimate file space usage
```

Common options	Description
-h	Print sizes in human readable format
-a	Include all files not just directories
-s	Display only total for each argument

### 3.3. Navigating the file system tree

The following commands are common commands for navigating the file system:

<code>ls</code>	- List files and directories
<code>cd</code>	- Changes the current directory location
<code>pwd</code>	- Prints path of current working directory
<code>pushd</code>	- Adds a directory to the top of the directory stack
<code>popd</code>	- Removes entries from the directory stack

Linux filesystem is built of a list of standard directories naming (sometimes residing as a standalone partitions) as follows:

<code>/</code>	- The filesystem's tree root
<code>/bin</code>	- Contains system commands used both by the system administrators and non-privileged users
<code>/dev</code>	- Stores all special device files
<code>/etc</code>	- Stores system's and applications' configuration files
<code>/initrd (/boot)</code>	- Contains images to provide the capability to load a RAM disk and the kernel (vmlinuz) by the boot loader (either LILO or GRUB)
<code>/lib</code>	- Contains kernel modules and shared library images needed to boot the system and run the commands in the root filesystem
<code>/lost+found</code>	- Stores files recovered due to improper system shutdown
<code>/mnt</code>	- Path to system's mounted devices
<code>/proc</code>	- Interface to kernel's data structure
<code>/root</code>	- Home directory of root user
<code>/tmp</code>	- Default location of temporary files
<code>/usr</code>	- Stores application programs
<code>/var</code>	- Stores log files, mails and other data

Every user in the system usually has a home directory, which he or she is referred to on login.

### 3.4. File types, Naming & Directory structure

Everything you encounter on a Linux filesystem is a file, hence there are several types of file in Linux filesystem:

- ✓ **Regular file:** Stream of binary or ASCII bytes
- ✓ **Directory:** Files which list other files

- ✓ **Special file and Block device:** The mechanism used for input and output to/from hardware peripherals
- ✓ **Link:** A way to make a file or a directory visible in multiple parts of filesystem's tree
- ✓ **(Domain) socket:** A special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.
- ✓ **Named pipe:** Act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

Example of directory listing:

```
[user@localhost ~]# ls -l
total 28340
drwxr-xr-x  2   root   4096   Jan 22 2010   dir1
---x---x---x  1   root   0      Jan 22 2010   1.sh
-rw-r--r--  1   root  1263   Nov 22 2008   anaconda-ks.cfg
-rw-r--r--  1   root  56155  Nov 22 2008   install.log
lrwxrwxrwx  1   root   28     Nov 22 2008   vfontcap -> /etc/vfontcap
```

The list above contains 6 columns, each contains the following information:

Column	Content
drwxr-xr-x	Indicates the file type and permissions
2	Indicates number of hard links (for more information please refer to chapter 8.2). For directories, indicates number of internal directories reside
1263	Indicates File size. Notice that directories usually have fixed size of 4096 bytes
Jan 22 2010	Indicates file Update date
install.log	Indicates the file/directory name

The permissions column is described in more details in chapter 8.1.

The first letter at the first column indicates the file type:

- Regular file
- d Directory
- l Soft link
- c Special file
- s (Domain) Socket
- p Named pipe
- b Block device

**Example:**

```
drwxr-xr-x  2 root root 4.0K Oct 25 23:29 dir1
-rwxr-xr-x  1 root root  80K Oct 25 23:28 ls
lrwxrwxrwx  1 root root   11 Oct 25 23:29 my.cnf -> /etc/my.cnf
crw-rw----  1 root uucp  4, 109 Oct 25 03:02 ttyS45
```

In Linux filesystem, the file naming is case sensitive, i.e.: two files named “test” and “Test” can be stores in same directory since the filesystem refers to them as two different names.

```
-rw-r--r--  1      root    0      Sep  2 00:26  test
-rw-r--r--  1      root    0      Sep  2 00:26  Test
```

Like in other operating systems, Linux’s filesystem also supports hidden files and directories. In order to change file/directory from regular to hidden, add dot (“.”) as prefix to its name.

In order to list all files in a directory including hidden files add the “a” parameter to the command as follows:

```
[user@localhost ~]# ls -la
total 28340
drwxr-x---  9      root    4096    Sep  2 00:31  .      #Hidden directory
drwxr-xr-x 26      root    4096    Sep  1 16:27  ..     #Hidden directory
drwxr-xr-x  2      root    4096    Jan 22 2010  dir1
-rw-r--r--  1      root    1263    Nov 22 2008  anaconda-ks.cfg
-rw-r--r--  1      root   56155    Nov 22 2008  install.log
-rw-r--r--  1      root    555     Nov 22 2008  .test #Hidden file
```

Notice that while listing hidden files, two hidden directories appear: “.” & “..” which act as follows:

- . - A special directory referencing to the current directory
- .. - A special directory referencing to the upper directory

**Example:**

```
[user@localhost ~]# ls -l .      # List current directory. Equals to
                                # using "ls -l"
[user@localhost ~]# ls -l ..     # List upper directory
[user@localhost ~]# ls -l ../    # Same as previous command
```



The special character “~” is a shortcut indicating the shell to consider the logged user’s home directory as the relative lookup path, for example:

```
[user@localhost /]# ls -l ~           # List current user's home directory
[user@localhost /]# ls -l ~/scripts  # List "scripts" directory under
                                     current user's home directory
```



**Tip:** In order to quickly change the current directory to your home directory use either of the options:

```
[user@localhost ~]# cd ~
```

OR JUST

```
[user@localhost ~]# cd
```

### 3.5. Manipulating files and directories

Linux contains a long list of commands and utilities to control and manipulate files and directories.

Below you can find introduction to some commonly used commands/utilities. For more details on their variant options use “*man <command name>*” (refer to chapter 4.3 for more details about using Linux help):

**ls** - List files and directories

```
[user@localhost ~]# ls [options] [path]
```

Common options	Description
-l	List using a long listing format including: owner, size, update date etc.
-a	List files including hidden
--color	Use colorized list according to listed file types
-h	Print file size in a human format
-R	List subdirectories recursively
-S	Sort list by size
-t	Sort list by modification date

**mkdir** - Make directory

```
[user@localhost ~]# mkdir [options] [dir name]
```

Common options	Description
-p	Makes hierarchical directory tree; ignores errors if directory exists

Example:

```
# Creates directory "1", inside it creates directory "2" and so on.
[user@localhost ~]# mkdir -p 1/2/3/4
rmkdir -      Remove empty directory
[user@localhost ~]# rmkdir [options] <directory name>
```

Common options	Description
-p	Remove directory and each directory component of that path name

Example:

```
# Removes directory "4", then removes directory "3" and so on.
[user@localhost ~]# rmdir -p 1/2/3/4
```

```
mv -      Move file or directory location
[user@localhost ~]# mv [options] <src> <dst>
```

Common options	Description
-i	Prompt before overwrite, for example: # mv -i file1 / mv: overwrite `/file1'?

Example:

```
# Moves file "file1" to "/" using interactive mode
[user@localhost ~]# mv -i file1 /
mv: overwrite `/file1'?
```

```
rm -      Remove file or directory
[user@localhost ~]# rm [options] <destination name>
```

Common options	Description
-f	Force removing file (without prompting)
-i	Ask before removing
-r or -R	Remove the contents of directories recursively

Examples:

```
# Removes file "file.txt" without verification
[user@localhost ~]# rm -f /tmp/file.txt
```

```
# Removes file "file.txt" with user approval
[user@localhost ~]# rm -i /tmp/file.txt
rm: remove regular file `file.txt'?

# Prompt for internal content remove
[user@localhost ~]# rm -r dir1
rm: descend into directory `dir1'?

# Force remove without prompting
[user@localhost ~]# rm -rf dir1

cp      -      Copy file or directory
[user@localhost ~]# cp [options] <src> <dst>
```

Common options	Description
-f	If an existing destination file cannot be opened, remove it and try again
-i	Prompt before overwrite
-p	Preserve the specified attributes (default: mode, ownership, timestamps) and security contexts
-r OR -R	Copy directories recursively

Examples:

```
# Asks user approval before copying
[user@localhost ~]# cp -i file1 /
cp: overwrite `/file1'?

# Copies directory "dir1" recursively to "/tmp"
[user@localhost ~]# cp -r dir1 /tmp
```

```
cat      -      Concatenate files and print to the standard output
[user@localhost ~]# cat <file name>
```

Example:

```
[user@localhost ~]# cat /etc/passwd          # Print to terminal the
                                              content of "passwd" file
```

```
file     -      Determine file type
[user@localhost ~]# file <file name>
```

Example:

```
[user@localhost ~]# file /tmp/test.pdf
/tmp/test.pdf: PDF document, version 1.2
tail -          Output the last part of file
[user@localhost ~]# tail [options] <file name>
```

Common options	Description
-f	Output appended data as the file grows

#### Examples:

```
# Output last 10 sentences of the file "file1" and keep updating the
terminal as data appended to the file
[user@localhost ~]# tail -f /var/log/boot.log
Sep  6 09:12:02 emanuels sendmail: sendmail -HUP succeeded
Sep  6 09:12:02 emanuels sendmail: sm-client -HUP succeeded
Sep  6 10:12:03 emanuels sendmail: sendmail -HUP succeeded
Oct 25 23:16:32 localhost sysctl: kernel.sysrq = 0
Oct 25 23:16:32 localhost sysctl: kernel.core_uses_pid = 1
Oct 25 23:16:32 localhost network: Setting network parameters:  succeeded
Oct 25 23:16:37 localhost ifup:
Oct 25 23:16:37 localhost ifup: Determining IP information for eth0...
Oct 25 23:16:45 localhost ifup:  done.
Oct 25 23:16:56 localhost network: Bringing up interface eth0:  succeeded
```

## 4. The Shell

### 4.1. What is a shell?

Sometimes called *command shell*, a shell is the command processor interface. The command processor is the program that executes operating system commands. The shell, therefore, is the part of the command processor that accepts commands. After verifying that the commands are valid, the shell sends them to another part of the command processor to be executed.

Linux offers a choice between several different shells. The most popular are *Cshell* (*CSH*), *Bourne shell* (*BASH*), *TCSH* and the *Korn shell* (*KSH*). Each offers a somewhat different command language.

### 4.2. Shell as a command line interpreter

Linux command line is a very extensive command line interpreter, including built-in system variables, naming completion (commands, files/directories names and more), supporting strong scripting capabilities (conditioning, loops, functions etc.) and more.

The Shell may be used interactively or non-interactively.

In interactive mode, the shell accepts input typed from the keyboard. When executing non-interactively, the shell executes commands read from a file or other input sources.

Linux shell provides a set of built-in commands implementing functionality impossible or inconvenient to obtain via separate utilities. For example, `cd`, `break`, `continue`, and `exec` cannot be implemented outside of the shell since they directly manipulate the shell itself.

While executing commands is essential, most of the power (and complexity) of the shell is derived by its embedded programming languages. Like any high-level language, the shell provides variables, flow control constructs, quoting and functions.

#### 4.2.1. Controlling the cursor

Linux shell is an interactive shell, and the command-line input can be edited using key sequences. The shell binds the keys stroke as follows:

Key	Action
down	Down-history
up	Up-history

Key	Action
left	Backward-char
right	Forward-char
Ctrl+a	Moving cursor to the beginning of the line
Ctrl+e	Moving cursor to the end of the line
Ctrl+c	Cancel command input
Ctrl+l	Clear screen
Ctrl+h	Equal to backspace key
Ctrl+s	Freeze the key input
Ctrl+q	Unfreeze key input
Ctrl+u	Delete all line
Ctrl+p	Print previous command from history
Ctrl+z	Suspend the program
Esc+d	Delete current word
Ctrl+u	Delete from beginning to current position

#### 4.2.2. Name completion

The shell is often able to complete words when given a unique abbreviation. Type part of a word (for example 'ls /ro') and hit the **tab** key to run the *complete-word* editor command. The shell completes the name '/ro' to '/root/', replacing the incomplete word with the complete word in the input buffer (Note the terminal completion adds a '/' to the end of completed directories and a space to the end of other completed words, in order to speed typing and provide a visual indicator of successful completion).

Example:

```
[user@localhost ~]# ls /ro<tab>
[user@localhost ~]# ls /root/
```

If no match is found (perhaps '/root' doesn't exist) or more than one match found, the terminal bell rings. If the word is already complete (perhaps there is a '/root' on your

system, or perhaps you were thinking too far ahead and typed the whole thing) a ‘/’ or space is added to the end, if it isn’t already there.



**Tip:** Completion works anywhere in the line, not just at the end.

Completed text pushes the rest of the line to the right. Completion in the middle of a word often results in leftover characters to the right of the cursor that needs to be deleted.

Commands and variables can be completed in much the same way. For example, typing ‘em[tab]’ would complete ‘em’ to ‘emacs’ if *emacs* were the only command on your system beginning with ‘em’.

Completion can find a command in any directory in **PATH** or if given a full pathname. Typing ‘echo \$ar[tab]’ would complete ‘\$ar’ to ‘\$argv’ if no other variable began with ‘ar’.

You can list the possible completions of a word at any time by either **double pressing** the <tab> key or by typing ‘^D’ to run the *delete-char-or-list-or-eof* editor command. The shell lists the possible completions and reprints the prompt and unfinished command line, for example:

```
[user@localhost ~]# ls /usr/l[^D]
lbin/ lib/ local/ lost+found/
[user@localhost ~]# ls /usr/l
```

OR

```
[user@localhost ~]# ls /usr/l<tab><tab>
lbin/ lib/ local/ lost+found/
[user@localhost ~]# ls /usr/l
```

### 4.3. Commands and arguments

The general form of a shell command line looks as follows:

```
prompt command options arguments
```

The command determines what operation the shell will perform and the options and arguments customize or fine-tune the operation.

Linux generally stores program files which act as command in */bin*, */usr/bin*, or */usr/local/bin*.

Commands can be issued by themselves, such as **ls**. A command behaves differently when you specify an *option*, usually preceded with a dash (-), as in **ls -a**. The same option character may have a different meaning for another command. GNU programs take long options, preceded by two dashes (--), like **ls --all**. Some commands have no options.

The argument(s) to a command are specifications for the object(s) on which you want the command to take effect. An example is **ls /etc**, where the directory `/etc` is the argument to the **ls** command. This indicates to see the content of that directory, instead of the default, which would be the content of the current directory, obtained by just typing **ls** followed by **Enter**. Some commands require arguments, sometimes arguments are optional.

Options which described between <> are mandatory, while options described between square brackets [] are optional.

#### 4.3.1. Commands History

The Linux shell stores the last 1000 executed commands via the commands line to a buffer named “history”. In order to see recently executed commands, use the “history” command as follows:

```
[user@localhost ~]# history
```



**Tip:** The history buffer size is resizable and can be set to a different buffer size (please refer to chapter 5.1 for variables set) by updating the history buffer size settings as follows:

```
[user@localhost ~]# HISTSIZE=2000
```

#### 4.4. Wildcards

*Wildcards* is a shell feature that makes the command line much more powerful than any GUI file manager. Wildcards are special characters allowing select filenames that match certain patterns of characters. This helps to select even a big group of files by typing just few characters.

Here's a list of commonly used wildcards:

Wildcard	Meaning	Example
*	Match any string	<code>ls -l *</code>



Wildcard	Meaning	Example
?	Match any single character string	<code>ls -l ?</code>
[...]	Match any enclosed characters	<code>cat [abc]test</code>
[...-...]	Match exactly one character in the given range	<code>mv [a-g]test /tmp</code>
{word1,word2}	Match exactly one entire word in the given options	<code>rm {con,self}test</code>

## 4.5. Quotes

The Linux command-line interpreter supports enclosing string in quotes as part of the command usage, as follows.

Quote	Name	Meaning
"	Double quote	Anything enclosed in double quotes manipulated as <b>string</b> <b>Note:</b> except \ and \$ and `
'	Single quote	Anything enclosed in single quotes remains unchanged
`	Back quote	Anything enclosed is executed as <b>command</b>

In order to understand the quotes usage we will introduce the `echo` command first.

One of the most basic and commonly used commands is the “echo” command:

```
echo - Display a line of text
[user@localhost ~]# echo [options] [strings]
```

Common options	Description
-n	Do not output trailing newline (\n)
-e	Enable interpretation of the backslash-escaped characters

Examples for common uses of “echo” command:

```
[user@localhost ~]# echo "hello world" # Print string
hello world
```

```
[user@localhost ~]# echo $USER          # Print system variable
root

[user@localhost ~]# echo -n "test"      # Print w/o new line
test[user@localhost ~]#

# Print while controlling the output display
[user@localhost ~]# echo -e "First line\n\tSecond line with tab"
First line
        Second line with tab
[user@localhost ~]#
```

#### Quotes usage examples:

```
[user@localhost ~]# A=123                # Define variable A as "123"
[user@localhost ~]# echo "A is: $A"      # Output: A is: 123
[user@localhost ~]# echo 'A is: $A'      # Output: A is: $A
[user@localhost ~]# echo `A is: $A`      # Output: command not found
```

## 4.6. Getting help

Unlike other operating system's help, Linux provides a brief user manual for every utility installed (unless the utility developer did not provide one, which rarely happens). The user manual is a command named "man", and used as follows:

```
man -          format and display the on-line manual pages
[user@localhost ~]# man [options] <command name>
```

#### Example:

```
[user@localhost ~]# man ls
```

The output is the user manual for "ls" command, usually containing command description, command syntax and supported options.

Some commands have multiple man pages. For instance, the **passwd** command has a man page in section 1 and another in section 5 (since section 1 describes the command line utility and section 5 describes the **/etc/passwd** file content). By default, the man page with the lowest number is shown. If you want to see another section than the default, specify it after the **man** command:

```
[user@localhost ~]# man 5 passwd
```

If you want to see all man pages about a command, one after the other, use the `-a` option of `man`:

```
[user@localhost ~]# man -a passwd
```



**Note:** Sometimes running “man” on a command returns the help of the currently used shell. This occurs in cases the command is part of the shell and not a standalone utility, for example: “*man pushd*”.

In addition to “man” command, a detailed version for user manual can be found using “info” command.

```
info - read info documents
```

```
[user@localhost ~]# info [options] <command name>
```

Example:

```
[user@localhost ~]# info ls
```

A short index of explanations for commands is available using the **whatis** command, like in the examples below:

```
whatis - search the whatis database for complete words
```

```
[user@localhost ~]# whatis <keyword>
```

Example:

```
[user@localhost ~]# whatis ls
```

```
ls      (1)  - list directory contents
```

```
ls      (lp) - list directory contents
```

If you don't know how to get started and which man page to read, **apropos** gives more information:

```
apropos - search the whatis database for strings
```

```
[user@localhost ~]# whatis <keyword>
```

Example:

```
[user@localhost ~]# apropos ftp
```

```
.netrc [netrc]      (5)  - user configuration for ftp
```

```
Net::Cmd            (3pm - Network Command class (as used by FTP, SMTP)
```

```
Net::FTP            (3pm)- FTP Client class
```

```
ftp                 (1)  - Internet file transfer program
```

```
ftp [pftp]          (1)  - Internet file transfer program
```

```
ftputils             (5)  - list of users that may not log in via the FTP
```

```
lftp                 (1)  - Sophisticated file transfer program
```

```
netrc          (5)  - user configuration for ftp
sftp           (1)  - secure file transfer program
sftp-server    (8)  - SFTP server subsystem
vsftpd         (8)  - Very Secure FTP Daemon
vsftpd.conf [vsftpd] (5) - config file for vsftpd
```

A list of all ftp related strings from *whatis* database will appear.

## 5. The Shell Environment

### 5.1. Shell variables

In order to process the information resides in the hard drive, the information must be kept in the machine's RAM memory. RAM memory is divided into small locations, and each location has unique number called memory location or memory address. Programmer can give a unique name to this memory location or address called **memory variable** or **variable**.

The defined variables are then automatically inherited by executed commands or scripts, and can be used, changed and unset.

In Linux (Shell), there are two types of variable:

**System variables** - Created and maintained by Linux itself. This type of variable usually defined in CAPITAL LETTERS.

**User defined variables** - Created and maintained by the user. This type of variable usually defined in lower-case letters.

In order to list the environment variables use the command **printenv**, for example:

```
[user@localhost ~]# printenv
HOSTNAME=emanuel.com
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=::ffff:192.168.23.1 52676 22
SSH_TTY=/dev/pts/0
USER=root
HOME=/root
...
```



**Caution:** Do not modify System variable unless you are sure of the impact, otherwise this may create normal system operation errors.

In order to define additional environment variable, user-defined environment variable, use the following syntax:

<variable name>=<variable value>

```
[user@localhost ~]# abc=bin
```



**Tip:** Different shells might have different syntax for variable setting.

In order to unset environment variable, use the following syntax:

**unset <variable name>**

```
[user@localhost ~]# unset abc
```

In order to use variable at the command line integrated with any command, use the name of the variable adding "\$" as its prefix, for example:

```
[user@localhost /bin]# cd $HOME          # change current location to /root
[user@localhost /root]#

[user@localhost ~]# abc=bin
[user@localhost ~]# ls -l /$abc          # print the content of /bin dir
```

Variable naming is limited to the following rule:

Must begin with Alphanumeric character or underscore character (\_), followed by one or more Alphanumeric character or underscore.

For example:

Variable assignment	Status
[user@localhost ~]# abc=123	Valid
[user@localhost ~]# _abc=123	Valid
[user@localhost ~]# ab1c=123	Valid
[user@localhost ~]# ab_c=123	Valid
[user@localhost ~]# 5abc=123	Not valid
[user@localhost ~]# @abc=123	Not valid
[user@localhost ~]# ab-c=123	Not valid

In order to list all available shell variables (both user defined and system environments) use the **set** command (w/o any arguments), for example:

```
[user@localhost ~]# set
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
```

```
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="00" [2]="15" [3]="1" [4]="release" [5]="i386-
redhat-linux-gnu")
BASH_VERSION='3.00.15(1)-release'
COLORS=/etc/DIR_COLORS.xterm
COLUMNS=202
DIRSTACK=()
DISPLAY=localhost:10.0
EUID=500
GROUPS=()
...
```



**Tip:** A faster way to see ALL variables defined in the system (both system and user-defined), use the following sequence:

```
[user@localhost ~]# echo ${IFS}
```

### 5.1.1. Exporting variable

The `export` and `'declare -x'` commands allow parameters to be added-to and deleted-from the environment. If the value of a parameter in the environment is modified, the new value becomes part of the environment, replacing the old.

The environment inherited by any executed command consists of the shell's initial environment, whose values may be modified in the shell, less any pairs removed by the `unset` and `'export -n'` commands, plus any additions via the `export` and `'declare -x'` commands.

Examples:

```
[user@localhost ~]# export a=123
# By exporting "a", any command or script executed via the same
environment will inherit variable "a".

# On the other hand, setting user variable by using:
[user@localhost ~]# a=123
# The variable will not be familiar to any command or script executed via
the same environment.
```

## 5.2. The search PATH

Linux determines the executable search path and accordingly their naming completion using the `$PATH` environment variable. This variable contains a list of directories, in which the shell is looking for executables within them:

```
[user@localhost ~]# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
```

To add new directory to the `$PATH` environment variable, use the following method:

```
[user@localhost ~]# PATH=$PATH:/home/emanuel/myscripts
```

At the example above, the directory `/home/emanuel/myscripts` was added to the shell's search path.

## 5.3. Shell aliases

Shell aliases make it easier to use commands by facilitating abbreviated command names and by enabling pre-specification of common arguments. To establish a command alias, issue a command of the form:

```
alias name='command'
```

Examples:

```
[user@localhost ~]# alias ll="ls -ltr"      # defining new alias named "ll"
[user@localhost ~]# ll                    # using the new alias "ll" will
                                         list the current directory
                                         content
```

Linux usually has a pre-defined aliases set. In order to list them run "alias" command:

```
[user@localhost ~]# alias                # list current aliases set
alias cp='cp -i'
alias df='df -lh'
alias ll='ls -l --color=tty'
alias mv='mv -i'
alias rm='rm -i'
```

In order to unset existing aliases use the "unalias" command:

```
unalias <alias name>
```

Example:

```
[user@localhost ~]# unalias ll           # del "ll" from the alias list
```





**Note:** Aliases are not expanded when the shell is not interactive (e.g. in scripts).

## 5.4. Environment Configuration Files

Shell variables, aliases, PATH update and other configuration made during the work with the shell is lost once the terminal session has terminated (logout), since any configuration done is available during the current session only.

In order to save all configuration changes for the next login, there are two hidden files located under the user's home directory controlling the environment configuration: **.bash\_profile** and **.bashrc**. These files are loaded once user logs to the system, or starts new session.

Another file holding configuration is the **.bash\_logout** which is executed once the user logs-out of the shell session.



**Note:** The environment configuration files are hidden files (starting with dot).

Each of the environment configuration file is executed on a different event as follows:

- ✓ **.bash\_profile** - executed once user logs-in to the system
- ✓ **.bashrc** - executed once user starts new shell session (logged-in already)
- ✓ **.bash\_logout** - executed once user logs-out of the shell session

Usually, **.bashrc** holds user specific aliases and functions while **.bash\_profile** holds user specific environment and startup programs. The **.bash\_logout** file is executed during shell logout process; hence any logout procedures are to be placed in it.

## 5.5. Configuring Shell Prompt

Most Linux systems have a default prompt in one color that indicates the logged in user name, the name of the machine working on and some indication of the current working directory.

The BASH supports customization of the prompt, supporting all sorts of information to be displayed (tty number, time, date, load, number of users, uptime etc.), supporting ANSI colors, either to make it look interesting, or to make certain information stand

out. In addition, the title bar of the terminal can be updated to reflect some of this information.

In order to change the prompt look, the “PS1” reserved variable is to be changed, and for command continuation prompt, the “PS2” is to be changed.

The prompt can indicate large number of properties using the following escape sequences during the “PS1” variable set:

Escape Sequence	Output
\a	ASCII bell character (07)
\d	Date in "Weekday Month Date" format (e.g., "Tue May 26")
\e	ASCII escape character (033)
\h	Hostname up to the first `.'
\H	Hostname
\n	Newline
\r	Carriage return
\s	Name of the shell, the basename of \$0 (the portion following the final slash)
\t	Current time in 24-hour HH:MM:SS format
\T	Current time in 12-hour HH:MM:SS format
\@	Current time in 12-hour am/pm format
\u	Username of the current user
\v	Version of bash (e.g., 2.00)
\V	Release of bash, version + patchlevel (e.g., 2.00.0)
\w	Current working directory
\W	Basename of the current working directory
\!	History number of this command
\#	Command number of this command
\\$	If the effective UID is 0, a #, otherwise a \$
\nnn	The character corresponding to the octal number nnn
\\	Backslash

Escape Sequence	Output
\[	Begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt
\]	End a sequence of non-printing characters

For example:

```
[user@localhost ~]# PS1="\u#\h \W>> "      # Set PS1
# Will change the prompt line as follows
user#localhost ~>>
```

In order to make a permanent change of the prompt, update the `.bashrc` file located at the user's home directory with the "PS1" variable set.



## 6. Command Input and Output

Most Linux commands read input and write output. By default, input is being given using the keyboard, and the output is displayed to the terminal.

The keyboard is the standard input (stdin) device, and the terminal is the standard output (stdout) device.

However, these default settings don't necessarily have to be applied. The standard output may be a file or a printing device and the standard input may be a text file etc.

Input and output redirection is done by the following operators:

Operator	Syntax	Description	Comments
>	command > filename	Command's output redirected to the BEGINNING of file	Creates file if does not exist
>>	command >> filename	Command's output redirected to the END of file (appended)	Creates file if does not exist
<	command < filename	Input redirection to command from file	Not all commands support this redirection mode
	command1   command2	Pipe - redirect output of command1 as an input of command2	

### Examples:

```
# Redirect output of "cat" to the BEGINNING of file 2.log
[user@localhost ~]# cat 1.log > 2.log
```

```
# Append output of "cat" to the END of file 2.log
[user@localhost ~]# cat 1.log >> 2.log
```

```
# Input redirection - sending mail using mail body from file instead from
user (stdin)
[user@localhost ~]# mail emanuel@gmail.com < mail_body.txt
```

```
# Redirect the output of "cat" to "sort" command
```

```
[user@localhost ~]# cat /test.txt | sort
# Combine input and output redirection. The file text.txt is first
checked for spelling mistakes, and the output is redirected to an error
log file:
[user@localhost ~]# spell < book.txt > spelling_mistakes.log
```

Output of one command can be piped into another command virtually many times, as long as these commands would normally read input from standard input and write output to the standard output.

Similar to any other open file in the system, the standard input and output also have unique file descriptor as if they were files. Naturally, their file descriptors ID are the initial ones:

I/O	File descriptor ID	Device name
Standard <b>input</b>	0	/dev/stdin
Standard <b>output</b>	1	/dev/stdout
Standard <b>error</b>	2	/dev/stderr



**Tip:** A very commonly used system device is the /dev/null. This is a special device discarding all data written to it.

If the file descriptor number is omitted, and the first character of the redirection operator is <, the redirection refers to the standard input (file descriptor 0).

If the first character of the redirection operator is >, the redirection refers to the standard output (file descriptor 1).

It is possible to explicitly provide the file descriptor ID to the command. The syntax usage is as follows:

```
# List all files under / and redirect the standard output into
"file_list.txt"
[user@localhost ~]# ls -l / > file_list.txt

# List all files under / and redirect the standard output into
"file_list.txt" and error messages (standard error) to "list_err.txt"
[user@localhost ~]# ls -l / > file_list.txt 2> list_err.txt

# List all files under / and redirect the standard output into
"file_list.txt" and append error messages (standard error) to
"list_err.txt"
```

```
[user@localhost ~]# ls -l / > file_list.txt 2>> list_err.txt
# List all files under / and redirect the standard output into
"file_list.txt" and error messages (standard error) to /dev/null
[user@localhost ~]# ls -l / > file_list.txt 2> /dev/null

# List all files under / and redirect both the standard output and
standard error into "file_list.txt".
[user@localhost ~]# ls -l / > file_list.txt 2> file_list.txt

# Search for "string" at file "1.log". Ignore all errors by redirecting
the standard error output of "grep" execution to /dev/null device
[user@localhost ~]# grep "string" 1.log 2> /dev/null
```

### Making things complicated...

Can you guess what the following command does?

```
[user@localhost ~]# kill -9 8198 > /dev/null 2>&1
```

**Answer:** kills process with PID 8198 (by sending signal SIGKILL [9] to the process), and redirects any output of the kill command including error outputs to /dev/null;

How come?

Lets explain it step by step:

```
[user@localhost ~]# ls /tmp > dirlist.txt
```

Redirect direct standard output to the file *dirlist.txt*.

```
[user@localhost ~]# ls /tmp > dirlist.txt 2>/dev/null
```

Redirect standard output to the file *dirlist.txt* and the standard error to /dev/null.

```
[user@localhost ~]# ls /tmp > /dev/null
```

Redirect standard output to /dev/null.

```
[user@localhost ~]# ls /tmp > /dev/null 2>&1
```

Redirect standard output to /dev/null and standard error to standard output - which is already redirected to /dev/null.

Pay attention: the location of **2>&1** is important:

```
[user@localhost ~]# ls 2>&1 > dirlist.txt
```

Will only direct standard output to *dirlist.txt*. This happens because the standard error is duplicated as standard output **before** the standard output was redirected to *dirlist.txt*.

The use of the ampersand (&) indicates that the following number is not a file name, but rather a location that the data stream is pointed to.

**Important:** the “&” sign should not be separated from the “2>” by spaces. If it would be separated, it would be pointing the output to a file again. The example below demonstrates this:

```
[user@localhost ~]# ls 2> tmp           # write ls errors to tmp file
[user@localhost ~]# ls 2 > tmp          # List file named 2 to "tmp"
ls: 2: No such file or directory

# Following examples are equal (space after the "&" sign is legal)
[user@localhost ~]# ls /tmp > dirlist.txt 2>&1
[user@localhost ~]# ls /tmp > dirlist.txt 2>& 1
```



## 7. Processes

### 7.1. Programs and Processes

Linux operating system is a multi-user multi-process operating system, enabling multiple users running multiple commands at the same time.

While some programs initiate a single process like **rm**, others initiate a series of processes, such as **firefox** web browser.

Every process is allocated with a unique ID during its execution, and a dedicated directory containing full process details resides at the **/proc** filesystem.

The **/proc** filesystem is the kernel's memory map. Its content is cleared on system shutdown, therefore while listing the content of the **/proc** filesystem, most of the files and directories are zero sized, and no physical allocation consumed from the hard drive.

In order to manage the processes running within the system, Linux operating system provides a set of process management utilities.

Commonly used utilities for process management are:

```
ps          -          Report a snapshot of the current processes
[user@localhost ~]# ps [options]
```

Common options	Description
-e or -A	List all processes on the system
-C <cmd name>	List all processes corresponding to the <cmd name>
-U <uid>	List all processes corresponding to the <user ID>
-o <format>	Format the output according to a predefined format
-f	List processes using full-format listing
-F	List processes using extended full-format listing
-u	List processes using user-oriented format
-H	List process hierarchy



**Note:** some options can't be used with others. Refer to the user manual of "ps" command for more information.

Examples for common uses of the “ps” command:

```
[user@localhost ~]# ps -eF
[user@localhost ~]# ps -AH
[user@localhost ~]# ps -u
[user@localhost ~]# ps -U root
[user@localhost ~]# ps H
[user@localhost ~]# ps -C sleep
[user@localhost ~]# ps -o "%u%g%p%c"
```

**top** - Linux task manager

```
[user@localhost ~]# top
```

Common options	Description
W	Save configuration changes
h	Open help menu
q	Quit
d	Set update interval
1	Toggle single CPU vs. separate CPU information
f	Select sort field

**pstree** - Display system's processes using tree view

```
[user@localhost ~]# pstree [options]
```

Common options	Description
-a	Show command line arguments
-p	Show process ID

**pidof** - Find the process ID of a running program

```
[user@localhost ~]# pidof [options] <program name>
```

Common options	Description
-s	Show the PID of the first process found
-x	Show process ID of scripts in addition to programs

Examples for common uses of “pidof” command:

```
[user@localhost ~]# pidof bash
[user@localhost ~]# pidof -x test.sh
```

```
[user@localhost ~]# pidof -s httpd
kill -      Notify process with signal
[user@localhost ~]# kill [options] <process ID>
```

Common options	Description
-s <signal id>	Send specific signal to process
-l	Print available signals

The kill command enables to send a wide range of signals to a process in order to control the process's execution, i.e. stop, suspend, abort, alarm etc.

More than 60 different signals are supported by the Linux operating systems. For more information regarding signals and their variation please refer to the manual of signal (7) as follows:

```
[user@localhost ~]# man 7 signal
```

In case no signal is provided by the command line, by default, the TERM signal is sent to the process.

Examples for common uses of the “kill” command:

```
[user@localhost ~]# kill 4450      # Send the TERM signal to process
[user@localhost ~]# kill -s 9 4454 # Send the KILL signal to process
[user@localhost ~]# kill -SIGHUP 458 # Send the SIGHUP signal to process
```

```
time -      Monitor running time of a command
[user@localhost ~]# time <command line>
```

Examples for common uses of the “time” command:

```
[user@localhost ~]# time ps -f      # monitor running time of "ps -f"
UID      PID  PPID  C  STIME TTY      TIME CMD      # Output of the
root      6212 6210  0 16:43 pts/2    00:00:00 -bash      "ps -f" command
root      9234 6212  0 19:57 pts/2    00:00:00 ps -f

real      0m0.064s
user      0m0.006s
sys       0m0.024s      # Executing time output by the
                        "time" command
```

Linux command line supports execution of several commands in queue written in one sentence by separating the commands with “;” symbol, for example:

```
# The following example shows execution of 3 commands, one after the
other using the ";" symbol separating each. The commands are executed
according to the written order.
```

```
[user@localhost ~]# ls -l ; rm -f /tmp/1.log ; cp ./file /root
```

## 7.2. Job control and Signals

Processes, in some cases, are running interactively while in others processes will have to continue to run even when the user who started them logs out.

Processes have two running modes:

- ✓ Run in *foreground*
- ✓ Run in *background*

Run in foreground - every regular command/program which is executed via the shell is by default running in foreground, means, the shell session is occupied during its execution.

Run in background - process is executed, but the shell session can accept new commands during its execution.

Job control refers to the ability to selectively stop (suspend) the execution of a process and continue (resume) its execution at a later time.

When the shell starts a job asynchronously, it prints a line that looks like: [1] 25433

Meaning:

- [1] - Indicating the index number of the job at the shell's job control list
- 25433 - Indicating the process unique ID

A process can be switched from running in foreground to be running in background and vice versa. There are several ways to change process's running mode:

Mode change	Description
Run in fg	Execute the command regularly
Run in bg	Add "&" to the end of the command line
fg → bg	When the program is running do: <ol style="list-style-type: none"> <li>1. Press <b>Ctrl+z</b> (suspends the program execution)</li> <li>2. Type <b>bg</b></li> </ol>
bg → fg	Run <b>jobs</b> command and verify the command's index

Mode change	Description
	Run <b>fg n</b> (where “n” is the process’s index ID from the jobs output)

Pressing the *suspend* sequence (Ctrl+z) while a process is running causes the process to be stopped and returns control to the shell. A Ctrl+z takes effect immediately.

### Examples:

```
# Run the program in foreground
[user@localhost ~]# sleep 1000

# Run the program in background
[user@localhost ~]# sleep 1000 &
[3] 8507

# Switch running mode from foreground to background
[user@localhost ~]# sleep 1500          # run program regularly
Ctrl+z                                # suspend it using Ctrl+z
[1]+  Stopped                  sleep 1500    # process suspended
[user@localhost ~]# bg              # change mode to background
[1]+ sleep 1500 &

# Switch running mode from background to foreground
[user@localhost ~]# sleep 1000 &        # run program in background
[1] 8584
[user@localhost ~]# sleep 1500 &        # run program in background
[2] 8585
[user@localhost ~]# sleep 2000 &        # run program in background
[3] 8586
[user@localhost ~]# jobs                # list background processes
[1]  Running                  sleep 1000 &
[2]-  Running                  sleep 1500 &
[3]+  Running                  sleep 2000 &
[user@localhost ~]# fg 2                # run program 2 in foreground
sleep 1500
```

The “+” symbol indicates the last job which was stopped or executed in the background, and the “-” symbol indicates the previous job executed in background.

```
wait - Wait for the specified process to end
[user@localhost ~]# wait <process ID>
```

Examples for common uses of “wait” command:

```
[user@localhost ~]# sleep 200 &      # Run sleep in background
[1] 19741
[user@localhost ~]# wait 19741      # Waiting for the “sleep” command to
                                     end using its PID. Afterwards return
                                     control to the shell

[user@localhost ~]# sleep 200 &      # Run sleep in background
[1] 19741
[user@localhost ~]# wait             # Waiting for the last command to
                                     end, afterwards return control to
                                     the shell
```



**Note:** wait command cannot be executed in background or suspended using the Ctrl+Z signal.

In case the user attempts to exit the shell while jobs are stopped, the shell will print a message warning that there are stopped jobs. The `jobs` command may then be used to inspect their status. If a second attempt to exit is made without an intervening command, the shell will not print another warning, and result in:

1. Stopped jobs will be terminated
2. The background running jobs will continue running having init process as their parent process.

```
# Logging out while there are stopped processes
[user@localhost ~]# logout
There are stopped jobs          # Shell message
```

In addition to the process run modes described above, some processes are called “daemon”. Daemon is a process (usually a system service), which runs in background and its parent process is usually the “init” process, since its executing shell was terminated.

These processes are typically represented inside square parentheses [], for example:

```
[user@localhost ~]# ps -eF
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	15:12	?	00:00:01	init [3]
root	2	1	0	15:12	?	00:00:00	[migration/0] #daemon

```
root          3          1    0 15:12 ?                00:00:00 [ksoftirqd/0]  #daemon
```

### 7.3. Exit status

On process termination, either normally or abnormally, an EXIT status integer value is saved to the environment variable “\$?”. This variable is constantly updated and tracks the exit status of the last command. In order to verify the EXIT status code of the last command executed, you could use the following command: “**echo \$?**” right after its termination, for example:

```
[user@localhost ~]# sleep 5      # Will delay the shell for 5 seconds
[user@localhost ~]# echo $?      # Print the "sleep" exit code status
0                                # The exit code was "0"
[user@localhost ~]# sleep 5      # Will delay the shell for 5 seconds
Ctrl+c                          # Stop the process using Ctrl+c signal
[user@localhost ~]# echo $?      # Print the "sleep" exit code status
130                              # The exit code was "130" due to abnormal
                                termination
```





## 8. Organizing Files

### 8.1. File ownership & permission

On a Linux system, every file is owned by a user and a group of users. There is also a third category of users, those who are not the user owner and don't belong to the group owning the file called others.

For each category of users, *read*, *write* and *execute* permissions can be granted or denied.

File permissions are indicated by nine characters as follows:

```
[user@localhost ~]# ls -l
-rw-r--r-- 1 root vuser 56155 Nov 22 2008 install.log
```

File's permissions indicators

- The first three characters (of the nine) display access rights for the **actual user** that owns the file.
- The next three are for the **group owner** of the file.
- The last three for **other** users.

The permissions are always in the same order: **(r)ead**, **(w)rite**, **e(x)ecute** for the user, the group and the others.

Note that hyphen (-) indicates no permission set.

In order to change file's ownership or permission use the following commands:

```
chmod - Change file access permissions
[user@localhost ~]# chmod [options] <permission update> <file name>
```

Common options	Description
-R	Change files and directories recursively

Permission addition is done using the + (plus) character, on the contrary, removal of permission is done by the - (minus) character.

Examples:

```
# Add execution permissions to user
[user@localhost ~]# chmod u+x install.log
```

```
# Remove execution permissions to user
[user@localhost ~]# chmod u-x install.log

# Add write permissions to group
[user@localhost ~]# chmod g+w install.log

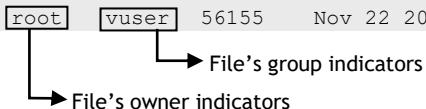
# Add read permissions to other
[user@localhost ~]# chmod o+r install.log

# Short way of changing permissions:
# User  rwx   (7)
# Group rw-   (6)
# Other r--   (4)
[user@localhost ~]# chmod 764 install.log
[user@localhost ~]# ls -l install.log
-rwxrw-r--  1 emanuel vuser 56155 Nov 22  2008 install.log
```

User can change permissions only to files which he (or she) is their owner.

Ownership of the file is indicated by two columns as follows:

```
[user@localhost ~]# ls -l
-rw-r--r--  1 root vuser 56155 Nov 22 2008 install.log
```



**chown** - Change file's ownership and group

```
[user@localhost ~]# chown [options] <owner:[group]> <file name>
```

Common options	Description
-R	Change files and directories' ownership recursively

**chgrp** - Change file's group

```
[user@localhost ~]# chgrp <group name> <file name>
```

Common options	Description
-R	Change files and directories' group recursively

Examples:

```
# Change file's ownership to user "emanuel"
[user@localhost ~]# chown emanuel install.log

# Change file's ownership (user and group) to "emanuel" and "users"
[user@localhost ~]# chown emanuel:users install.log

# Change file's group ownership to "users" by adding ":" before the group
name
[user@localhost ~]# chown :users install.log

# Change file's group ownership to "users"
[user@localhost ~]# chgrp users install.log
```

The use of the **chown** and **chgrp** commands is restricted for non-privileged users. If you are not the super user of the system, you cannot change user nor group ownerships for security reasons.

Linux shell permits fast changing current user without re-login by using the "su" (switch user) command. This is especially useful when super user permissions are required for executing specific command, e.g. change the ownership of file and more.

```
su      -      Switch current user
[user@localhost ~]# su [-] [username]
```

The hyphen indicates to the shell to switch the whole shell environment according to the user's environment configuration, including home directory, shell used etc.

Examples:

```
# Switch current user to be "root" user (including shell environment)
[emanuel@localhost ~]# su - root
Password:
[user@localhost ~]#

# Switch current user to be "emanuel" user (using current shell
environment)
[jacob@localhost ~]# su emanuel
Password:
[emanuel@localhost ~]#
```

In order to verify your current real and effective UIDs and GIDs use the “id” command:

```
id          -          Display current real and effective UIDs and GIDs
[user@localhost ~]# id [user name]
```

#### Examples:

```
# Display current logged in user's ID
[user@localhost ~]# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm)

# Display user's info for other user (emanuel)
[user@localhost ~]# id emanuel
uid=2003(emanuel) gid=101(vuser) groups=101(vuser)
```

## 8.2. Hard and soft links

Similar to other operating-systems' filesystem, Linux filesystem supports linking between files for various reasons.

A link is nothing more than a way of matching two or more file names to the same set of file data. There are two ways to achieve this:

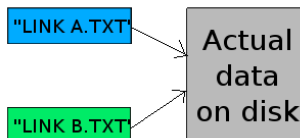
- ✓ **Hard link:** Associate two or more file names with the same inode. Hard links share the same data blocks on the hard disk, while they continue to behave as independent files.

There is an immediate disadvantage: hard links can't span partitions, because inode numbers are unique within a given partition.

- ✓ **Soft link or symbolic link:** a small file which is a pointer to another file. A symbolic link contains the path to the target file instead of a physical location on the hard disk. Soft links can span across partitions.

The process of *unlinking* hardlinks dissociates a name from the data on the filesystem without destroying the associated data. The data is still accessible as long as at least one link points to its inode. When the last link is removed, the space is considered free.

In the figure below, two hard links, named "LINK A.TXT" and "LINK B.TXT", point to the same physical data:



If the filename "LINK A.TXT" is opened in an editor, modified and saved, those changes will be visible even if the filename "LINK B.TXT" is opened for viewing since both filenames point to the same data.

In order to make either symbolic or hard link use the “ln” command:

**ln** - Links between files

```
[user@localhost ~]# ln [options] <target> [link name]
```

Common options	Description
-s	Create symbolic link

# Create hard link

```
[user@localhost ~]# ln /etc/my.cnf /etc/mysql.cnf
```

# Create soft link

```
[user@localhost ~]# ln -s /etc/my.cnf /home/my.cnf
```

```
[user@localhost ~]# ls -l
```

```
total 3
```

```
lrwxrwxrwx 1 root root 11 Sep  6 00:15 my.cnf -> /etc/my.cnf
```



## 9. Power Tools

### 9.1. Using diff, find and which

Some more commonly used commands with the Linux shell are: `diff`, `find` and `which`.

The “`diff`” command compares two files and returns the difference between the two.

```
diff -          Compare two files
[user@localhost ~]# diff [options] <file1> <file2>
```

Common options	Description
-b	Ignore changes in amount of white space
-q	Report only whether the files differ, not the details of the differences.
-i	Ignore changes in case
-r	Recursively compare any subdirectories found.
-w	Ignore white space when comparing lines
-y	Use the side by side output format

The output of the comparison looks as follows:

```
[user@localhost ~]# diff file1 file2
4c7          # "4" indicates line number on left file (file1)
              vs. "7" line number compared on right side (file2)
< joe        # line on left side file (file1)
---
> jim        # different line on right side file (file2)
```

Linux shell provides several utilities to help us locate the location of file and directories. Two of these utilities are: ***find*** and ***which***.

```
find -          Search for files in a directory hierarchy
[user@localhost ~]# find <path> [options]
```

Common options	Description
-name <name>	The name of the file
-type <type>	Define the type of file to find (e.g. `f` for file, `d` for directory etc)
-maxdepth <levels>	Set the directory max depth to search in
-mount	Look for files located only in the current filesystem
-user <username>	Look for files owner by specific user
-printf <format>	Print results in specific format

### Examples:

```
# Look for files or directories named "file1" starting from "/" path
[user@localhost ~]# find / -name "file1"

# Look for files named starting with "pass" starting from "/etc" path
[user@localhost ~]# find /etc -name "pass*" -type f

# Look for files or directories with name containing "pass", start
searching from "/" path for max directory depth of 2 levels
[user@localhost ~]# find / -name "*pass*" -maxdepth 2

# Look for files or directories with name containing "pass", start
searching from "/" path and limit to the partition of the "/" location
[user@localhost ~]# find / -name "*pass*" -mount

# Look for files or directories starting with "pass" starting from "/"
path and owner by user "joe"
[user@localhost ~]# find / -name "pass*" -user "joe"

# Look for files or directories starting with "pass" starting from "/"
path and display the output according to the format provided
[user@localhost ~]# find / -name "pass*" -printf "%u:%h%p\n"
```

**which** - Display the full path of (shell) command

```
[user@localhost ~]# which <command name>
```

### Example:

```
[user@localhost ~]# which ls
alias ls='ls --color=tty'      # first match which is the alias set
```



```
/bin/ls # Second match is the /bin/ls command
```

## 9.2. Using **grep**

One command which is commonly used with redirection and piping is the “grep” command:

```
grep - Print line matching defined pattern
[user@localhost ~]# grep [options] <pattern> <file name>
```

Common options	Description
-c	Count number of appearances of pattern in file
-E	Interpret PATTERN as an extended regular expression
-i	Ignore case while searching pattern
-l	Print only file names of those files which pattern was found
-R	Search recursively in directory
-s	Suppress error messages about nonexistent or unreadable files
-v	Invert the sense of matching
--color	Surround the matching string with color

Examples:

```
# will print all lines containing the string "test" from file /tmp/file1
[user@localhost ~]# grep "test" /tmp/file1
```

```
# Search process named "ssh" from current running processes
[user@localhost ~]# ps -ef | grep ssh
```

```
# Combine output redirection and piping.
# In the example below the output of the history command is piped to the
# grep command as input, in which the pattern "rm" is searched and all
# output is saved under /tmp/ directory:
[user@localhost ~]# history | grep --color "rm" > /tmp/rm_commands.log
```

## 9.3. Regular expressions

A regular expression is a pattern describing a certain amount of text. The shell supports a level of regular expressions within the command line, but not its entire standard.

The power of regular expressions is when it is needed to manipulate some strings from text files, i.e. email address, and to automate the operation.

The tables below provide the syntax rules with corresponding examples at the table afterwards.

Example for utilities support regular expressions are the “grep” and “egrep”. These utilities print lines matching a pattern according to the regular expression provided.

At the following example, the “egrep” will list all lines which the sub-string “root” is located **ONLY** at the beginning of the line:

```
[user@localhost ~]# egrep "^root" /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

### Brackets & Positions

Syntax	Description	Example #
[]	Match anything inside the square brackets for one character position once and only once	12,13
-	The - (dash) <b>inside square brackets</b> is the 'range separator' and allows us to define a range	12,13
^	Match the beginning of the string	8,10
\$	Match the end of the string	8
\b	Match the beginning or end of a word	1-5,7,9,12, 13,15,16,19

### Repetitions & Metacharacters

Syntax	Description	Example #
*	Repeat any number of times	2,5,10,15, 19
+	Repeat one or more times	6,11,15,18
?	Repeat zero or one time	16-18,20
{ <i>n</i> }	Repeat <i>n</i> times	4,7-8,14
{ <i>n,m</i> }	Repeat at least <i>n</i> , but no more than <i>m</i> times	9,14
{ <i>n</i> ,}	Repeat at least <i>n</i> times	20
	Find the left OR right values	12,13

Syntax	Description	Example #
*?	Repeat any number of times, but as few as possible	
+?	Repeat one or more times, but as few as possible	
??	Repeat zero or one time, but as few as possible	
{ <i>n,m</i> }?	Repeat at least <i>n</i> , but no more than <i>m</i> times, but as few as possible	
{ <i>n</i> ,}??	Repeat at least <i>n</i> times, but as few as possible	

### Special Characters

Syntax	Description	Example #
.	Match any character except newline	2
\w	Match any alphanumeric character	5,7,9,10, 15-19
\s	Match any whitespace character	15,18
\d	Match any digit	3-4,6,8,14
\<num>	Back reference - match the text that was captured by group number <num>	

### Negatives of Special Characters

Syntax	Description	Example #
\W	Match any character that is NOT alphanumeric	
\S	Match any character that is NOT whitespace	11
\D	Match any character that is NOT a digit	
\B	Match a position that is NOT the beginning or end of a word	
[^abc]	Match any character that is NOT one of the characters <i>abc</i>	19

### Grouping

Syntax	Description	Example #
(exp)	Match exp and capture it in an automatically numbered group	13-18
(?<name>exp)	Match exp and capture it in a group named name	

Syntax	Description	Example #
(?:exp)	Match exp, but do not capture it	
<b>Lookarounds</b>		
(?=exp)	Match any position preceding a suffix exp	16,18
(?<=exp)	Match any position following a prefix exp	17,18
(?!exp)	Match any position after which the suffix exp is not found	20
(?<!exp)	Match any position before which the prefix exp is not found	
(?#comment)	Comment	

**Examples:**

#	Expressions	Description
1	\belvis\b	Find <i>elvis</i> as a whole word
2	\belvis\b.*\balive\b	Find text with "elvis" followed by "alive"
3	\b\d\d\d-\d\d\d\d	Find seven-digit phone number
4	\b\d{3}-\d{4}	Find seven-digit phone number a better way
5	\ba\w*\b	Find words that start with the letter a
6	\d+	Find repeated strings of digits
7	\b\w{6}\b	Find six letter words
8	^\d{3}-\d{4}\$	Validate a seven-digit phone number from beginning of text
9	\b\w{5,6}\b	Find all five and six letter words
10	^\w*	The first word in the line or in the text
11	\S+	All strings that do not contain whitespace characters
12	\b05[0 2 5]-[0-9]{3}	Lines containing numbers of type: 050-XXX, 052-XXX, 054-XXX
13	\b((050) (052) (054))-0-	Same as above

#	Expressions	Description
	9]{3}	
14	(\d{1,3}\.){3}\d{1,3}	A simple IP address finder
15	\b(\w+)\b\s*\1\b	Find repeated words
16	\b\w+(?=ing\b)	The beginning of words ending with "ing"
17	(?<=\bre)\w+\b	The end of words starting with "re"
18	(?<=\s)\w+(?=\s)	Alphanumeric strings bounded by whitespace
19	\b\w*q[^u]\w*\b	Words with "q" followed by NOT "u"
20	\d{3,}(?! \d)	At least three digits not followed by another digit



## 10. The VI text editor

### 10.1. Concepts of the VI editor

The **vi** editor is a very powerful text editor including an extensive built-in manual, which can be activated using the **:help** command when the editor is running.

What makes **vi** confusing to the beginner is that it can operate in two modes:

- ✓ Command mode
- ✓ Edit mode

The editor always starts in command mode. Commands move you through the text, enable search, replace strings, mark blocks and perform other editing tasks, and some of them switch the editor to edit mode. Hence, each key has not one, but likely two meanings: it can either represent a command for the editor while in command mode, or a character while in a edit mode.



**Note:** In order to use the VI editor, it must be installed on the system first.

Running the VI is done as follows:

```
[user@localhost ~]# vi # Open VI
```



**Note:** It is recommended to provide at the command line the name of new file to be created rather than doing so while saving the file via the VI interface:

```
[user@localhost ~]# vi new.txt // Open new document named "new.txt"
```

### 10.2. Using VI

As previously mentioned, VI works in two modes: command & edit modes.

As VI starts, the default mode is *command*. Switching mode is done as follows:

- ✓ Switch from command mode to edit mode press the **"i"** key.
- ✓ Switch from edit mode to command mode press the **"Esc"** key.

There are several concepts for using the commands (in command mode):

1. VI includes internal command-line for complicated operations, i.e.: replace text, file operations etc. Using VI's command line is achieved by pressing the **":"** key.

2. Searching text is done by pressing the “/” key and providing the string to be searched.
3. The table below summarizes most useful VI commands:

Command	Description
:q	Exit vi
:q!	Force exiting Vi
:w	Save changes
:wq	Save changes and exit
u	Undo last action
Ctrl + r	Redo
.	Repeat last operation
<b>Moving around</b>	
b	Move to the beginning of a word
e	Move to the end of a word
:n	Jump to line number <i>n</i>
<b>Editing</b>	
i	Insert before cursor
a	Append after cursor
A	Append to the end of the current line
o	Open a new line below and insert
O	Open a new line above and insert
C	Change the rest of the current line
r	Overwrite one character
ESC	Exit insert/edit mode
x	Delete characters under the cursor
dd	Delete the current line
~	Change the case of characters



Command	Description
yy	Copy one line
p	Paste the copied line
<b>Visual mode</b>	
v	Start highlighting characters
V	Start highlighting lines
<b>Search &amp; Replace</b>	
/string	Search the file for <i>pattern</i>
n	Scan for next search match in the same direction
N	Scan for next search match but opposite direction
:<R>s/test/text/<A>	<p>Substitute <b>test</b> with <b>text</b>. <i>R</i> determines the range and <i>A</i> determines the arguments</p> <p><u>&lt;R&gt; can be:</u></p> <p>nothing – work on same line</p> <p>number – work on line number</p> <p>% - work on all file</p> <p><u>&lt;A&gt; can be:</u></p> <p>g – replace all occurrences</p> <p>i – ignore case</p> <p>c – confirm each substitution</p>



## 11. Writing Shell Scripts

Shell script is simply a file that contains list of commands to be executed. By storing commands as a shell script it is easy to keep executing them.

### 11.1. Simple scripting

For example, in case we need to delete the content of the temporary directory once in a while, it might be better to put all commands into a single script named “cleaner” as follows:

```
#!/bin/bash
echo "Cleaning temporary directory content"
rm -rf /tmp/*
echo "Done."
```

- ✓ The `#!/bin/bash` line indicates which shell is executing the script.
- ✓ The `echo` commands simply print text on the terminal.
- ✓ The `rm` command removes from the `/tmp` directory all files and directories.

In order to make the script executable, add “execution” permission to the script as follows:

```
[user@localhost ~]# chmod ugo+x cleaner
```

This gives the owner, members of the group, and everyone else the ability to execute the file.

Finally, to execute the script simply call it:

```
[user@localhost ~]# ./cleaner
Cleaning temporary directory content
Done.
```

For detailed information on **shell scripting**, please refer to the “**Linux Shell Scripting**” book.



## 12. Basic Communications

### 12.1. Linux in a networked environment

Linux supports many different networking protocols, over each a wide set of utilities to communicate and control the nowadays networked environments. Some of the most common protocols are supported by the operating system: TCP/IP, HTTP, SMTP, POP, SNMP, PPP and many more.

Most common user friendly Linux distributions come with various graphical tools, allowing easy setup of the computer in a local network, for connecting it to an Internet Service Provider or for wireless access etc. These tools can be started up from the command line or from a menu:

- ✓ Ubuntu network configuration is done selecting: System → Administration → Networking.
- ✓ Red Hat Linux comes with **redhat-config-network**, which has both a graphical and a text mode interface.
- ✓ On Gnome systems: **gnome-network-preferences**
- ✓ On KDE systems: **knetworkconf**

Usually, the network configuration files are text files which can be configured manually. Some of these files and directories are:

- ✓ **/etc/hosts** - A name resolving file contains addresses of additional hosts, which can be contacted without using an external naming service.
- ✓ **/etc/resolv.conf** - This file contains your domain name and the name server(s) to contact.
- ✓ **/etc/sysconfig/network-scripts/** - Location of the file configurations for each network device on the system.
- ✓ **/etc/hostname** - This file stores the system's host name and the system's fully qualified domain name.
- ✓ **/etc/sysconfig/network** - This file is used to specify information about the desired network configuration on the server.

One of the most common utilities regarding setting up and configuring the Linux network environment is the “*ifconfig*”:

<b>ifconfig</b>	-	Configure the network interfaces
-----------------	---	----------------------------------

```
[user@localhost ~]# ifconfig [options]
```

Common options	Description
-a	Show information on all interfaces
up	Add IP address to the interface
down	Remove IP address from the interface

### Examples:

```
# Display all system's IP addresses
```

```
[user@localhost ~]# ifconfig -a
```

```
eth0      Link encap:Ethernet  HWaddr 00:0C:29:20:94:3F
          inet addr:192.168.23.128  Bcast:192.168.23.255
          Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe20:943f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15746 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10688 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1615348 (1.5 MiB)  TX bytes:1191701 (1.1 MiB)
          Interrupt:185 Base address:0x2000
```

```
eth0:1    Link encap:Ethernet  HWaddr 00:0C:29:20:94:3F
          inet addr:192.168.23.130  Bcast:192.168.255.255
          Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:185 Base address:0x2000
```

```
# Add new IP address to the NIC
```

```
[user@localhost ~]# ifconfig eth0:2 192.168.23.200 up
```

## 12.2. Networking utilities

The following table contains a list of commonly used network utilities. For more details please refer to their manual using “man <command>”:

Utility	Description
ping	Administration utility used to test whether a particular host is reachable across an Internet Protocol (IP) network and to measure the round-trip time for packets sent from the local host to a destination computer, including the local host's own interfaces.

Utility	Description
telnet	Utility that provides a bidirectional interactive text-oriented communications facility via a virtual terminal connection.
ftp	A file transfer utility to data transfer from one computer to another
rlogin	A utility allowing users to log in on another host via a network
rsh	A utility enabling execution of specified commands over a remote host
rcp	A non interactive utility enabling copying files from remote host





## 13. Looking After Your System

### 13.1. Starting and stopping the system

Linux is famous with its long uptime capabilities with no shutdown demands (due to administration reasons), yet sometimes you might need to run some administrative actions, which will demand system shutdown, start-up service control and more.

The next list of commands describes some common used administrative management commands:

```
uptime      -      Display how long the system has been running
[user@localhost ~]# uptime
03:22:38 up 12:10,  1 user,  load average: 0.11, 0.04, 0.01
```

```
shutdown    -      Bring the system down
[user@localhost ~]# shutdown [options]
```

Common options	Description
-h	Halt after shutdown
-t sec	Wait for <sec> before shutting down
-c	Cancel running shutdown

```
reboot      -      Reboot the system
[user@localhost ~]# reboot
```

Common options	Description
-h	Halt after shutdown
-t sec	Wait for <sec> before shutting down
-c	Cancel running shutdown

Linux operating system supports several system load levels, each provides different uses as follows:

- Runlevel 0 - halt (Do NOT set init default to this)
- Runlevel 1 - Single user mode
- Runlevel 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
- Runlevel 3 - Full multiuser mode

- Runlevel 4 - unused
- Runlevel 5 - X11 (GUI)
- Runlevel 6 - reboot (Do NOT set initdefault to this)

Linux operating system enables switching between runlevels without rebooting the system using the “init” command:

```
init - Process control initialization
[user@localhost ~]# init [options]
```

Common options	Description
1-5	Select runlevel
-s	Change to single user mode
-b	Boot directly into a single user shell without running any other startup scripts

Example:

```
# Switch to GUI mode
[user@localhost ~]# init 5
```

The boot definitions, including default level and more can be found at `/etc/inittab` file.

## 13.2. Managing Services

During each of the runlevels described above, different set of services is loaded, according to the runlevel needs. Configuring the services to load according to the runlevel can be controlled (by the super user only) using the “*chkconfig*” utility:

```
chkconfig - Updates runlevel information for system services
[user@localhost ~]# chkconfig <options>
```

Common options	Description
--list	Lists all of the services which chkconfig knows about
--add <name>	Add new service for management by chkconfig
--del <name>	Delete service from chkconfig management
<name> <on off reset>	Change load status of service

**Examples:**

```
# Print list of all managed services
[user@localhost ~]# chkconfig --list

sshd          0:off  1:off  2:on   3:on   4:on   5:on   6:off
winbind       0:off  1:off  2:off  3:off  4:off  5:off  6:off
psacct        0:off  1:off  2:off  3:off  4:off  5:off  6:off
mysqld        0:off  1:off  2:off  3:off  4:off  5:off  6:off
...

# Add httpd service to start during boot time for all runlevels
[user@localhost ~]# chkconfig httpd on

# Add httpd service to start during boot time at runlevel 3
[user@localhost ~]# chkconfig --level 3 httpd on
```



**Tip:** For more service configuration abilities, use the services configuration files located under the `/etc/rc.d` directory.

During system runtime, services can be either started or stopped using the **service** command:

```
service -          Control system services
[user@localhost ~]# service <options> [service name [command] ]
```

Common options	Description
<code>--status-all</code>	Provides status of all system services

Each service might support a different command list. The basic supported commands are: **start** | **stop** | **status** | **reload** | **restart**



**Note:** The service command can be executed by *root* user only.

**Examples:**

```
# Control "httpd" service
[user@localhost ~]# service httpd stop

Stopping httpd:          [ OK ]
```

In order to get the list of all available services to be controlled, run **service** using **--status-all** flag or use the **chkconfig** utility.

#### Examples:

```
# List all system services
[user@localhost ~]# service --status-all

# List all system services using chkconfig utility
[user@localhost ~]# chkconfig --list
```

### 13.3. User manipulation

Since Linux operating system is a multi-user operating system, user's addition and deletion is a quite standard operation taken by system administrators.

The following files contain the list of existing users and their hashed passwords:

- ✓ **/etc/passwd** - contains the list of all users registered in the system
- ✓ **/etc/shadow** - contains the user's passwords (saved encrypted)

The following commands are common commands used for basic user management:

```
passwd - Update a user's password
[user@localhost ~]# passwd [options] [username]
```

The "**passwd**" updates the current user's password. Only "**root**" user is allowed to reset password for other users.

Common options	Description
-d <username>	Remove user's password (disable user login)

#### Examples:

```
# As root, Change password to user "emanuel"
[user@localhost ~]# passwd emanuel
Changing password for user emanuel.
New UNIX password:
Retype UNIX password:
```

```
# As "emanuel", change self password
[emanuel@localhost ~]# passwd
```

```
Changing password for user emanuel.
```

```
(current) UNIX password:
```

```
New UNIX password:
```

```
Retype UNIX password:
```

```
whoami -          Print effective userid
```

```
[user@localhost ~]# whoami
```

Example:

```
[user@localhost ~]# whoami
```

```
emanuel
```

```
useradd -          Add user to the host
```

```
[user@localhost ~]# useradd [options] <name>
```

Only “root” user is allowed to add new users.

Common options	Description
-d <home dir>	Define explicitly path to the user’s home directory
-G <group>	Define user’s group name
-p passwd	Define user’s password

Examples:

```
# As root, add user “jacob”
```

```
[user@localhost ~]# useradd -d /home/users/jacob -G users jacob
```

```
userdel -          Remove user from the host
```

```
[user@localhost ~]# userdel [options] <name>
```

Only “root” user allowed to delete existing users.

Common options	Description
-r	Remove user’s home directory

Examples:

```
# As root, remove user “jacob” (leave home directory)
```

```
[user@localhost ~]# userdel jacob
```

## 13.4. Other operations

Yet more useful utilities used during administrative system management are:

```
who      -      Shows who is logged in to the host
[user@localhost ~]# who [options]
```

Common options	Description
-b	Time of last system boot
-q	Count logged in users
-r	Provide current runlevel

```
date     -      Shows/update system's date and time
[user@localhost ~]# date [options]
```

The date command provides a wide set of options; for more details please refer to “date” manual.

Using “date” Examples:

```
# Get current time and date
[user@localhost ~]# date
Sat Sep  6 04:17:58 IDT 2008
```

```
# Change current time
[user@localhost ~]# date 111210342009
Thu Nov  12 10:34:00 IDT 2009
```

## 14. Appendix I - Shell Commands Examples

```
# Pause for some time
sleep 4                # sleep in seconds
usleep 1000            # sleep in microseconds
```

```
# Machine's date & date manipulation
date                   # Output: Thu Jan  7 17:23:37 GMT 2010

# Convert some date to seconds since 1970 (Output: 1292834230)
date --date="Sun Dec 20 08:37:10 UTC" +%s
date --date='2009-12-20 08:37:10 UTC' +%s
date --date='2009/12/20 08:37:10 UTC' +%s

# Calculate future date
date -d '2010-01-21 + 2 weeks 3 days'
date -d +2months17days
date -d yesterday-10days
```

```
# Using "sed" to Replace characters
sed "s/search pattern/replacement pattern/g" <filename>

# To extract a range of lines, say lines 2 to 4
sed -n 2,4p somefile.txt

# To extract lines 1,2 and 4
sed -n -e 1,2p -e 4p somefile.txt

# Replace character "1" with "2"
echo 1234 | tr "1" "2"
```

**Note:** sed/awk/grep etc. support extended regular expressions.

```
# Basic awk parsing command
# Print the first string until space (Output: emanuel)
echo "emanuel shmuelov" | awk '{print $1}'

# Using delimiter "," and print the lines without the first two segments
(Output: shmuelov)
echo "emanuel=shmuelov" | awk -F"=" '{print $2}'

# In case number of result values is unknown
```

```
echo "my,name,is,emanuel,shmuelov" | awk -F"," '{OFS=":" ; $1=$2="" ;
print $0}' # Output is "::is:emanuel:shmuelov"
```

```
echo "my,name,is,emanuel,shmuelov" | awk -F"," '{OFS=":" ; $1=$2="" ;
print $0}' | sed "s/:://g" # Output is "is:emanuel:shmuelov"
```

```
dirname "/tmp/dir1/file" # Output: /tmp/dir1
```

```
basename "/tmp/dir1/file1" # Output: file1
```

```
# Convert string/file case from lower to upper and vice versa
```

```
dd if=input_file of=output_file conv=ucase
```

```
dd if=input_file of=output_file conv=lc case
```

```
tr '[:lower:]' '[:upper:]' < input.txt > output.txt
```

```
awk '{ print toupper($0) }' input.txt > output.txt
```

```
# Using grep command
```

```
grep 'apples|oranges' fruits.txt # Look for apples OR oranges
```

```
grep -e "apples" -e "oranges" fruits.txt # Same as above
```

```
# Same but using extended regular expressions
```

```
grep -E 'green apples|red oranges' fruits.txt
```

```
# Using regular expressions: search for all lines with
```

```
# following pattern:
```

```
# STARTING with 5, then any num of characters, then "," and
```

```
# then "-1", then and set of characters and ENDING with 0
```

```
grep -E "^5.*,.*,\-1*0$" file
```

```
# Colorize output
```

```
grep --color=auto abc a_file.txt
```

```
# Preventing from grep to appears when using "ps"
```

```
ps -ef | grep emacs
```

```
# Provide the pids of all "less" processes
```

```
pgrep less
```

```
# Provide the pids of all "less" processes owned by "root"
```

```
pgrep -u root less
```



```
# Teach your shell new commands
alias grep="grep --color=auto"
alias ll='ls -lth --color=tty'
```



## 15. Appendix II - Common Linux Commands

Command	Description
alias	Create your own name for a command
arch	Print machine architecture
awk/gawk	Pattern scanning and processing language
basename	Remove directory and suffix from a file name
bash	GNU Bourne-Again Shell
bc	Command line calculator
bunzip2	Unzip .bz2 files
cat	Concatenate a file print it to the screen
chmod	Change file permissions
chown	Change the owner of a file
clear	Clear terminal screen (command line)
cp	Copy command
cut	Print selected parts of lines to standard output
date	Display date and time
dc	Command line calculator
df	Show amount of disk space free
diff	Determine difference between two files
diff3	Determine difference between 3 files
dos2unix	Converts plain text files in DOS/MAC format to UNIX format
du	Show disk usage
echo	Display a line of text
egrep	Print lines matching a pattern
env	Display the path
false	Exit with a status code indicating failure

Command	Description
find	Find a file
free	Display free memory
grep	Search for a pattern using regular expression
gunzip	Unzip .gz files
gzip	Compress using Lempel-Ziv coding (LZ77)
halt	Stop the system
head	Print the first 10 lines of a file to standard output
history	Display entire command history
hostname	Show or set the system's host name
id	Print information for username, or the current user
ifconfig	Display network and hardware addresses
kill	Terminate a process
ln	Create a link to the specified TARGET with optional LINK_NAME
locate	Locate a file
ls	List directory contents
lynx	Command to start the Lynx browser
mac2unix	Converts plain text files in DOS/MAC format to UNIX format
man	Display a particular manual entry
mkdir	Create a directory
mktemp	Make temporary filename (unique)
more	Page through text one screen at a time.
mv	Move and / or rename files
pidof	Get process id using process name
ping	Send ICMP ECHO_REQUEST to network hosts
pwd	Print Working Directory
reboot	Stop the system, power off, reboot

Command	Description
rename	Rename files
rm	Remove files or directories
rmdir	Remove a directory
rpm	rpm command options
sed	Stream editor
sh	Shell (BASH)
sleep	Delay for a specified amount of time
sort	Sort lines of a text file
ssh	Secure shell connection command
su	Become super user ( root )
tail	Print the last 10 Lines of a file standard output
tar	Create an Archive
tcsh	Enhanced completely compatible version of the Berkeley UNIX C shell, csh
tee	Copy standard input to each file, and also to standard output
telnet	User interface to the telnet protocol
touch	Change file timestamps
tree	Display file tree
true	Exit with a status code indicating success
uname	Print system information (kernel version)
uniq	Remove duplicate lines from sorted file
unlink	Call the unlink function to remove the specified file
untar	Un-archive a file
unzip	Unzip .zip files
usleep	Sleep a given number of microseconds. default is 1
vi	Start the vi editor

Command	Description
w	Show who is logged on and what they are doing
wc	Word count of a file
wget	Non-interactive download of files from the Web
which	Find command path
who	Show who is logged on
whoami	Print effective userid
whois	Client for the whois service
zip	Compression and file packaging utility
zipin fo	List detailed information about a ZIP archive