Contents

[1. General 1](#_Toc471277019)

[2. Boot Process 2](#_Toc471277020)

[3. Files System 3](#_Toc471277021)

[4. Console 5](#_Toc471277022)

[5. Files 6](#_Toc471277023)

[5.1 General 6](#_Toc471277024)

[5.2 Find 6](#_Toc471277025)

[5.3 Grep 7](#_Toc471277026)

[6. Archives 9](#_Toc471277027)

[7. Processes 10](#_Toc471277028)

[7.1 Priorities 10](#_Toc471277029)

[8. SSH 11](#_Toc471277030)

[9. Samba 12](#_Toc471277031)

[10. VIM 13](#_Toc471277032)

[10.1 Commands Modes 13](#_Toc471277033)

[10.2 Commands 13](#_Toc471277034)

[11. Shell Scripts 14](#_Toc471277035)

[11.1 Execute 14](#_Toc471277036)

[11.2 Commands 14](#_Toc471277037)

[12. Scheduling 16](#_Toc471277038)

[12.1 Cron 16](#_Toc471277039)

[13. Coding 17](#_Toc471277040)

[13.1 Compile 17](#_Toc471277041)

[14. Embedded Linux 18](#_Toc471277042)

[14.1 Cross Compiling 18](#_Toc471277043)

[14.2 License 18](#_Toc471277044)

[1. Embedded Linux and Copyright Law 19](#_Toc471277045)

[1.1 The Bottom Line: Use but don’t Link 19](#_Toc471277046)

[1.2 The details 19](#_Toc471277047)

[1.3 Copyleft – GPL the Viral License 19](#_Toc471277048)

[1.4 Rules to code by 19](#_Toc471277049)

# General

## Distributions

* Redhat – not free
* Ubuntu – receive consistent updates.
* Debian – known for its stability and therefore, its update cycle is much slower than in other distributions.

## Linux Information

* Get installed Linux version:

$ cat /proc/version

* Get help on Linux (section) of specific command:

$ man [<section>|-a] <command>

* Run administrator-level commands (Super User DO):

$ **sudo** <command>

* History of commands:  
  $ history
  + After ‘history’, you can execute specific command by: $ !{the command num}
  + You can also use $ history | grep ‘string’ to search for specific commands

## Linux regular expressions:

* <http://gnosis.cx/publish/programming/regular_expressions.html>

## Installations

* Get information on installed (Debian) packages:

Sudo dpkg –l <package name>

* Update the information on all the installed packages on your system:

sudo apt-get update

* Download and install packages (should run apt-get update before):

sudo apt-get install <package name>

* Upgrade installed packages to their latest version (should run apt-get update before):

sudo apt-get upgrade

# Boot Process

The final phase of the boot process is when **init** forks away from the Kernel, making it the first process in the user-space, the mother of all other user-space processes.

Init seeks out its configuration file, **/etc/inittab**. This file is an array of records, in the following format:

<record id (ignored)>:<run level>:<directive (sysinit/initdefault/wait/ctr

## Reboot

To reboot your Linux system:

sudo reboot

# Admin

* Create a new user account:

$ sudo adduser <username>

* Giving an existing user admin permissions:

$ sudo adduser <username> sudo

# Files System

* Show disk free space:

$ df

* Current directory:

$ pwd

* The previous directory:

-

* refer to your root directory (per user):

~

* refer to a different user’s root directory:

~<user>

* list a directory:

ls [-l long format] [-a show hidden files] [-R recursive] <search expressions>  
for example:

$ ls -l

drwxr-xr-x 4 cliff user 1024 Jun 18 09:40 WAITRON\_EARNINGS

-rw-r--r-- 1 cliff user 767392 Jun 6 14:28 scanlib.tar.gz

^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

| | | | | | | | | | |

| | | | | owner group size date time name

| | | | number of links to file or directory contents

| | | permissions for world

| | permissions for members of group  
| |  
| permissions for owner of file: r = read, w = write, x = execute -=no | permission  
|  
type of file: - = normal file, d=directory, l = symbolic link, and others...

* list all hidden files:  
  ls -al
* Changing file permission:

chmod 755 file Changes the permissions of file to be rwx for the

owner, and rx for the group and the world.

(7 = rwx = 111 binary. 5 = r-x = 101 binary)

chgrp user file Makes file belong to the group user.

chown cliff file Makes cliff the owner of file.

chown -R cliff dir Makes cliff the owner of dir and everything in

its directory tree.

Note: You must be the owner of the file/directory or be root before you can do any of these things.

* Important directories on Linux:
  + /home – where all the users’ directories
  + /etc – configuration files
  + /var – variable file. Files the you expect to grow over time. Usually for system and application logs.
  + /bin – where binaries that can be used by all users live (e.g. ls)
  + /sbin – where binaries that can be used only be the root user lives
  + /lib – where the libraries that service all the binaries in the system (both bin and sbin) lives
  + /usr – where all user application (binaries) live. These binaries, unlike the binaries in bin are not required by the system for bootup.
* Find a file on the system:

$ find –name <file-name>

* Find which file is running the command:

$ which <cmd>

* Copy directory – will re-calculate all symbolic links and keep the attributes of all files:

$ cp –dpR <dir-to-copy> <name-of-new-dir>

* Move/rename file:

mv file1 newname move or rename a file

mv file1 ~/AAA/ move file1 into sub-directory AAA in

your home directory.

# Console

* Clear screen:

$ clear

* Search for command in history:

Ctrl+R +[enter string to search for]

* Dump file contents to console:

$ cat <file-name> [<more files..>]

* Print the 1st/last 10 lines of the file:

$ head/tail <file name>

Optional parameters:

-f : the command will loop forever, checking for new data again and again.

* View file and scroll in it:

less <file name>

The **less** command allows you to use these key-commands to move around within the log:

* + Press the up-arrow and down-arrow keys to scroll while viewing the log.
  + Type gg to scroll to the top of the log.
  + Press Shift+g to scroll to the bottom of the log
  + Type /ERROR to search for text with the word ‘ERROR’. Replace it with any word you want to search.
  + Use q to quit the **less** view.
* Pipes:

$ <command> **>** <will write output to file, erasing previous contents>

$ <command> >> <will append output to file (without erasing previous contents>

$ <command 1> | <will send the output of command 1 as the command line arguments for command 2>

* Redirecting errors to file:

$ 2 > <file name>

* Redirecting errors to null (not displaying them):

$ 2 > /dev/null

* Print only strings from binary files – will print all the strings in the file and ignore all binary junk:

$ strings <file name>

* Command substitution:

You can use the output of one command as an input to another command in another way called command substitution. Command substitution is invoked when by enclosing the substituted command in backwards single quotes. For example:

cat `find . -name aaa.txt`

which will cat ( dump to the screen ) all the files named aaa.txt that exist in the current

directory or in any subdirectory tree.

# Files

## General

Some general useful commands:

* whereis <file name> - all the locations where is file exist
* which <program name> - which <program name> will be executed
* <program> --version
* <program> --help
* man <program/command>

## Find

Return all the files in the tree that satisfy the parameters:

**$ find** <lacation(s)> <comparison-criteria> <search-term>

Notes:

* <location(s)> - must end with “/“ in order to search the path
* <search-term> - must have “” around the search term.

0Search Creteria:

-name “<file-name wildcards expression>”

-iname “<file name with wilcards + ignore case>”

-maxdepth <num. if 1 – will only search location>

-not <search criteria> - will return all the files that don’t match.  
 can also use ! instead of –not

<search criteria> <nothing == AND / -o == OR> <search criteria> :   
 example: find –name ‘\*.php’ –o –name ‘\*.txt’ –not –name  
 ‘\*info.txt’  
 will find all the files that ends with ‘.php’ or  
 with ‘.txt’ but not the files that ends with   
 ‘info.txt’

-type <f: files / d: dirs> : will find only files/dirs.

-perm <permissions> : find files with specific premissions. Can use   
 all the same permission flags like in ‘chmod’ command  
 example: find –perm /u=r :will return all read-only files  
 find –perm /a=x : all executable files

-user <user name> : search in user root directory

-group <group name> : search files belonging to a specific group

-mtime <days> : will find files modified x days ago.

-atime <days> : will find files that were accessed in the last x days

-mtime +<x days> -<y days> : will find files that were modified  
 more than x days ago and less than y days ago.   
 Can also use only x or y e.g. ‘-mtime +x’

-cmin -<x minutes> : find files that were modified within the last   
 x minutes

-amin -<x minutes> : files that were accessed in the last x minutes

-[a|c]newer <file> : files that were modified/accessed(a)/changed(c)   
 more recently than <file> was modified.

-size <exact size: files in this size. E.g.-size 50MB

-size +<larger than size e.g. 50MB> -<smaller than size e.g. 100MB>

-empty : find empty files/directories

-exec <command> {} \; : execute Linux command on all the files that   
 you found in this find command.  
 examples:  
 // remove all .txt files in /tmp (and sub-directories): find /tmp -type f -name "\*.txt" -exec rm -f {} \;

// run ls on all files in the current directory and then  
 pipe the result to sort them in reverse order (from large  
 to small) and display the first 5 (largest files)  
mtime$ find . -type f -exec ls -s {} \; | sort -n -r | head -5

## Grep

search text in files and return all the lines that contains them:

$ grep [opt] “<pattern to search for>” <filenames…>

<pattern to search for> - simple regular expression. For more complex  
 regular expression, use egrep instead.

<-n> - print line numbers

<-v> - show all the lines that DON’T contain the text

<-A|B|C> <n> - show <n> lines after (A)/before(B)/both before and  
 after (C) the match

<-c> - count only – return the number of matching lines

<-i> - ignore case

<-w> - whole word only. Won’t match partial words.

<-x> - eXact match only – will return only line that are exact match

<-f> <pattern\_file> - allow you to specify a pattern\_file that  
 contains the pattern to search for.

<-r> - recursive in complete path

* **Egrep** – extended grep with additional support for complex regular expression  
  [**http://ryanstutorials.net/linuxtutorial/grep.php**](http://ryanstutorials.net/linuxtutorial/grep.php)
* **Fgrep** – fast grep. Does not support any regular expressions and therefore faster.
* Grep with pipe: In order to search with grep on the output of a command:

$ <command> | <grep> <pattern to search for>

For example:

**$ grep “hello” \*** ; will search all the files in the current directory for the string “hello”

find . | grep "hello" ; will search the lines containing the   
; word ‘hello’ in all the files in the   
; current tree

Note: this can also be used to search on the results of a previous grep search. Example:

$ grep “result:” <on file> | grep “errors”

# Archives

* gzip:

$ gzip <file name>

* slower but more efficient zip:

$ bzip2 <file name>

* Create archive for backing up a complete directory:

$tar –cvf <filename.tar> <directory to archive>

* Restore directory from archive:

$tar –xv –f <filename.tar>

# Processes

* See all the running processes on the system:

$ ps –aux

* Kill a process:

$ kill <PID of process to kill>

$ kill [-9 : kill immediately] <PID>

## Priorities

All user-processes on Linux starts with the same priority.

Linux Kernel schedules the process and allocates CPU time accordingly for each of them. But, when one of your process requires higher priority to get more CPU time, you can use nice and renice commands  
  
The process scheduling priority range is from -20 to 19. We call this as nice value.

A nice value of **-20 represents highest priority**, and a nice value of **s** for a process.

By default when a process starts, it gets the default priority of 0.

# IP

* See your network interfaces:

$ ifconfig

* Change the IP address of a network card:

$ sudo ifconfig <network interface name e.g. eth0 or enp0s3 or lo> <new IP address> netmask 255.255.255.0

# SSH

Secure Socket Shell, is a UNIX-based command interface and protocol for securely getting access to a remote computer.

**SSH to remote computer:**

$ ssh <user name of remote>@<remote ip/address>

**Copy files to remote computer:**

scp /path/to/file <user on remote>@<remote IP>:/path/to/destination

**Copy files from remote computer:**

scp <user on remote>@<remote IP>:/path/to/file /path/to/destination

**Set up transfer directory between the two computers – on Linux:**

1. Install sshfs:

sudo apt-get install sshfs

1. create a empty dir

mkdir /home/user/testdir

1. "link" or "mount" the two directories

sshfs user@server.com:/remote/dir /home/user/test

1. When you’re done - "unlink" the dirs

fusermount -u /home/youruser/remotecomp

**Set up transfer directory between the two computers – on Windows:**

# Samba

Another convenient way to share file between Linux and windows machines is through Samba.

<http://www.howtogeek.com/176471/how-to-share-files-between-windows-and-linux/>

To set it up:

* + - 1. On the windows machine:
         1. Create a folder and make sure it’s shared and has read/write permissions to everyone.
         2. Under folder->properties, check the network path – this is the ‘windows-share-folder-path’ you’ll need to use in the mount.cifs command on Linux (see below).
      2. On the Linux machine:

$ sudo apt-get install cifs-utils

$ mkdir ~/<share-dir>

$ sudo mount.cifs <windows-share-folder-path> /home/<linux-user>/<share-dir> -o user=<windows-user-name>

* + - 1. You should now be able to copy files into/from the share folder.  
         NOTE: you must have administrator permissions to copy/access the folder. Therefore, use sudo for the linux commands.

# VIM

## Commands Modes

i – go into insert mode

: - go into last line mode (to enter commands)

esc – go into commands mode

## Commands

You may have noticed that several commands combine a text operation and movement key. gg takes you to the end of a file, and d is used to delete. Combining them gives you something more powerful. Vim's like that. If you're working in Vim and think "hey, I wonder if I can combine two things I know to make something easier," the answer is often (but not always) yes.

|  |  |  |  |
| --- | --- | --- | --- |
| u | Undo |  |  |
| Ctrl-r | Redo |  |  |
| d | Delete:   * d starts the delete operation. * dw will delete a word. * d0 will delete to the beginning of a line. * d$ will delete to the end of a line. * dgg will delete to the beginning of the file. * dG will delete to the end of the file. | p | Paste after current line:  Will paste the word/line/part that was previously either   * deleted or * selected (v/V) * P – paste on current line. |
| /  (enter to search) | Search for text.   * n – search again * N – search again in reverse direction * ? – search from bottom to top. | v | Select text   * V – select lines * Ctrl-V – select columns * y – copy selected text to clipboard |
| :%s/text/replacement text/g | search through the entire document for text and replace it with replacement text. |  |  |
| :%s/text/replacement text/gc | search through the entire document and confirm before replacing text. | :w <filename> | Write (save) file (optional: to different filename). |
| G | Goto end of file |  |  |
| gg | Goto start of file |  |  |
| <num>G | Goto line number <num> | :q | Quit vim |
|  |  | :sq | Save and quit vim |

# Shell Scripts

<https://devhints.io/bash>

## Alias Commands

Open ~/.bash\_profile

For every command to alias, add a line:

alias <new\_alias>=’<full command>’

for example:

alias l='ls -lah'

## Execute

In order to write and execute a shell-script:

* The first line of the script should be:  
  #!/bin/bash

This tells Ubuntu what program to use to run the file.

* Save the script as <name>.sh
* Mark the file as executable: chmod +x <file>
* To execute: $ sh <name>.sh

Some additional comments:

* In Ubuntu, the current directory is not the program search path, so you need to run ./<filename>, not <filename>
* Variable names are $<varname>, not %<varname>%
* Commands in a shell script are not printed by default, as in a batch file. I you need to debug a script, run it as:  
  bash –x script\_file.sh
* Comments start with #, not rem.

## Commands

NOTE: It’s very important to note the spaces, otherwise the commands won’t work!!

|  |  |
| --- | --- |
| [ -f <file> ] | True if file exists and is a regular file. |
| Export MY\_ENV\_VAR=<command or string> | Set MY\_ENV\_VAR environment variable to the value after the ‘=’.  To use:  $MY\_ENV\_VAR |
| $<number>: | Value of parameter <number> where $0 is the script’s name:  ./script.sh Hello World  Will make  $0 = script.sh  $1 = Hello  $2 = World |
| [ <condition> ] && <command>|| <another command>  Note:   * || only runs the second half if the first half is FALSE. * && only run the second half if the first half is TRUE. | Short form for:  If [ <condition> ] then  <command> else  <another command> fi |
| My\_func() {….}  ….  My\_func <args> | Define a function that can later be called from within the script. |
| $@ | All of the input positional parameters as a string (used to pass them into internal functions or other scripts). |
| *#!/bin/bash*  FILES=**/**path**/**to**/\***  **for f in $FILES**  **do**  **echo** "Processing $f file..."  *# take action on each file. $f store current file name*  **cat** $f  **done** | Loop over all the files and execute the commands to each of them. |
| -- | If you need to pass parameters in the format –x to a command in the script (e.g. set –x), you need to write ‘--‘ before the parameter (end of options): set -- -x  So that the shell will treat the ‘-‘ as a parameter and not as a script option. |
|  |  |
|  |  |

# Scheduling

## Cron

The software utility **Cron** is a time-based [job scheduler](https://en.wikipedia.org/wiki/Job_scheduler) in [Unix-like](https://en.wikipedia.org/wiki/Unix-like) computer [operating systems](https://en.wikipedia.org/wiki/Operating_system). People who set up and maintain software environments use cron to schedule jobs (commands or [shell scripts](https://en.wikipedia.org/wiki/Shell_script)) to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration—though its general-purpose nature makes it useful for things like downloading files from the [Internet](https://en.wikipedia.org/wiki/Internet) and downloading [email](https://en.wikipedia.org/wiki/Email) at regular intervals.

*cron* is most suitable for scheduling repetitive tasks. Scheduling one-time tasks is often more easily accomplished using the associated [*at*](https://en.wikipedia.org/wiki/At_%28Unix%29) utility.

# Tools

## Netcat (nc)

I very powerful tool for utility is used for just about anything under the sun involving TCP, UDP, or UNIX-domain sockets. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6. Unlike Telnet, nc scripts nicely, and separates error messages onto standard error instead of sending them to standard output, as telnet(1) does with some.

# Coding

C/C++ compiler: gcc

Binutils: ld

## Compile

gcc -o <output file> <input source files>

for example:

gcc -o hello hello.c

# Windows Subsystem for Linux

<https://github.com/michaeltreat/Windows-Subsystem-For-Linux-Setup-Guide>

<http://wsl-guide.org/en/latest/index.html>

<https://www.hanselman.com/blog/TheYearOfLinuxOnTheWindowsDesktopWSLTipsAndTricks.aspx>

<https://docs.microsoft.com/en-us/windows/wsl>

## Installation

1. Make sure you have 64 bit windows 10
2. Start->’turn windows features on/off’->turn on ‘windows subsystem for Linux’  
   After windows will download and install the necessary files, you will need to restart windows
3. Open windows store tab and download the required Linux distribution
4. Start the WSL like any other program from the windows start menu.

# Embedded Linux

## Cross Compiling

UBoot

Compiling Linux

Building File system

Cross compiling applications

## License

# Embedded Linux and Copyright Law

## The Bottom Line: Use but don’t Link

GPL license means that if you link with a GPL code either statically or dynamically – all your code becomes GPL as well and you must give it on request!!

However, using GPL code through it’s user interface (e.g. command line interface) does not count as linking and therefore, you may use a vanilla or modified Linux kernel without risking your proprietary source code, as long as you don’t link to it (via compiler)!

## The details

Thu, 2002-08-01 00:00 - Michael Barr

by [Michael Barr](http://www.barrgroup.com/Embedded-Systems/Experts/Barr)

The rising popularity of Linux has spurred many embedded developers to consider it as an RTOS alternative. Here's the straight scoop on the legal implications for the proprietary parts of your firmware.

One of the more confusing aspects of the open source phenomenon has been the proliferation of different [source code licensing schemes](http://www.barrgroup.com/Embedded-Systems/How-To/Embedded-Linux-Open-Source).

There are so many different licensing terms, in fact, that if you are considering using multiple pieces of software developed by others in your products, you'll probably want to have an intellectual property lawyer read the license agreement for each such component and advise you how best to proceed.

Fortunately, if you only want to use Linux, the situation is much more straightforward.

## Copyleft – GPL the Viral License

A common myth is that the use of any piece of open source code, including Linux, requires the user to give away the source code to their own proprietary application. In truth, most open source licenses protect only the borrowed code and do not place any restrictions on other software you might develop for use alongside it.

The specific license accompanying the Linux kernel is called the GNU [General Public License (link is external)](http://www.fsf.org/licensing/licenses/gpl.html) (GPL). The GPL defines rules that apply when you are leveraging software you would not have had access to if the code were proprietary. Under these rules, anyone is entitled to improve or modify the Linux kernel and its device drivers, applications, and services. But because these modifications create a derivative of the existing code, they must be made public under the same licensing terms.

If you don't modify the operating system, the GPL requires only that you give credit where credit is due, do not impose any further licensing or distribution conditions upon your customers, and provide the Linux source code you used to your customers, if they request it. Those are pretty reasonable terms, by any measure.

## Rules to code by

Of course, there are many situations in which an engineering organization might want to keep its own code proprietary even when that code is surrounded by Linux's open source code. This is easily accomplished provided three rules of thumb are followed during development:

#### 1. Start proprietary software from a clean code base

By ensuring that your proprietary code does not build directly upon any open source code, you remain clear of the "derivative work" clause found in the GPL. Derivative works are the source of most legal confusion; they must typically be made open source under the same terms as the original code from which they are derived. But proprietary code that merely interfaces to open-source code is not derivative.

#### 2. Use only LGPL libraries

The GPL requires **any code that links to a GPL library--statically or dynamically--to also be released under the GPL**. However, a less protective license called the GNU [Lesser General Public License (link is external)](http://www.fsf.org/licensing/licenses/lgpl.html) (LGPL) was created so that developers could link to these open source libraries **dynamically** without being bound to release their application's source code. Most key Linux libraries are licensed under the LGPL.   
\* In LGPL, if the application is linked statically, the developer will have to supply their code in an object format (not source code) so that users can modify the library and re-link it with the application’s objects.

#### 3. Don't modify the interface to the Linux kernel

Under the GPL terms, any modification made to the monolithic portion of the Linux kernel must be released as open source software. Note, however, that if your application requires that you make changes to the kernel, only those kernel changes must be made public. You can still keep your application code (and even loadable kernel modules) proprietary, provided that they simply interface with the kernel via Linux's standard system calls.

If you observe these simple rules, you should be able to distinguish between Linux and your proprietary code for all intents and legal purposes. Of course, it may still be prudent to talk with an intellectual property lawyer.