
# System Software
# Linux command line manual: lab 1
# 2016 - 2017

## *Bachelor Electronics/ICT*

*Course coördinator:* *Luc Vandeurzen*
*Lab coaches:* *Floris De Feyter*
*Stef Desmet*
*Tim Stas*
*Jeroen Van Aken*
*Luc Vandeurzen*

*Last update: January 26, 2017*

***Important remark before you get started***

*The Linux operating system (OS) is used for these labs,. It will not be possible to implement the exercises in Windows OS. Moreover, you will soon find out that you really need Linux outside the lab hours to finalize your exercises, to make assignments or to experiment with the lecture code. Therefore, it's recommended to get access to a Linux system not only at the university but also at your home. You may consider a dual boot (Windows and Linux) solution on your computer (and probably this is the best solution), or use a virtual PC solution (VMware, VirtualBox, ...), or run Linux from a bootable USB stick. You might find some helpful tutorials here: linux.about.com/od/howtos/ss/How-To-Create-A-Persistent-Bootable-Xubuntu-Linux-USB-Drive.htm.*

## Short introduction to Linux

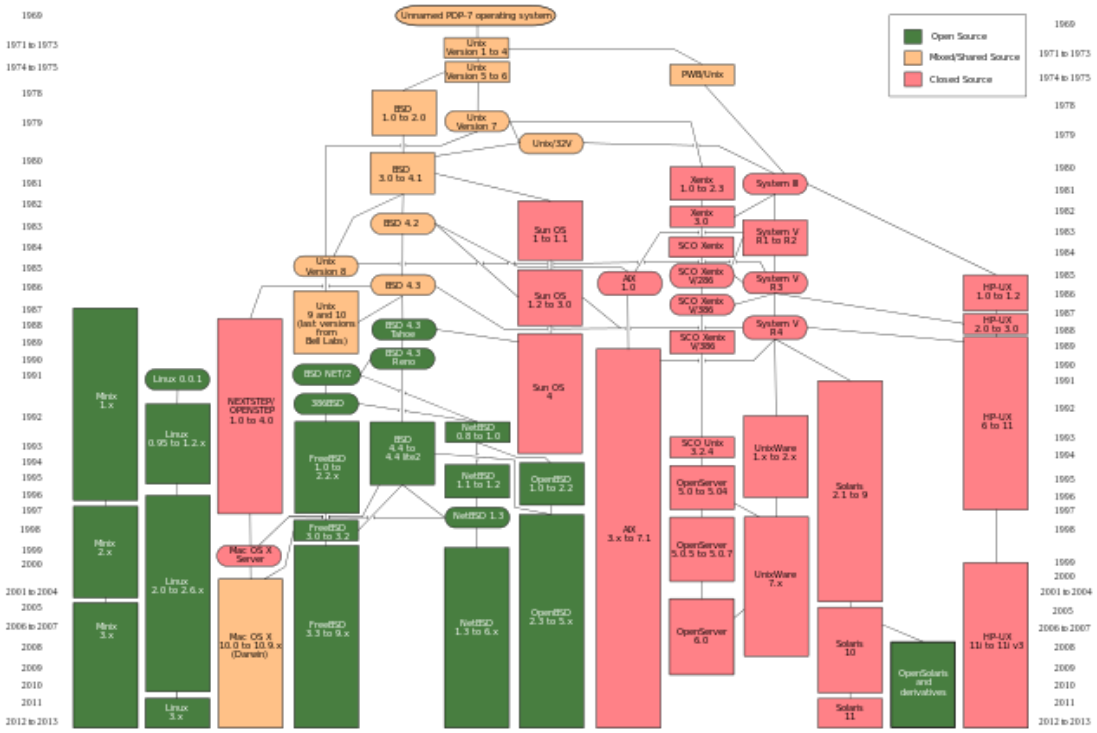
Linux has its roots in Unix. Actually, Linux is an implementation of Unix for desktop PCs. Unix is a multi-user multi-tasking operating system initially meant for mainframes, workstations, and servers. Although designed and implemented in the 1970s, it still is widely in use. Based on the statistics found at en.wikipedia.org/wiki/Usage_share_of_operating_systems, Windows OS rules the desktop/laptop market but most of the internet runs on Unix/Linux and almost every supercomputer runs Linux. Actually, its user-base is even bigger than ever and still growing as Unix/Linux is often the OS of embedded devices, smart-phones and tablets (Android runs on a Linux kernel!) and even your car. Go to the link https://en.wikipedia.org/wiki/List_of_Linux_adopters to get some impression on which countries and companies are running their business on Linux.
The history of Unix is sketched below (for a better view, visit https://upload.wikimedia.org/wikipedia/commons/7/77/Unix_history-simple.svg).
Also notice that MAC OS X is a Unix-based OS.

The Linux OS is comprised of several components, including:

- A boot loader: manages the startup (booting) of the OS;
- The Linux kernel
- Device drivers: software to control and interface the hardware devices;
- Daemons: services (printing, internet, …) running in background;
- The shell: a text-based interface that allows you to give instructions to the kernel, run programs, etc.;
- A window manager: a system that handles all graphics (windows, buttons, scrollbars, etc.) to be displayed on screen; Several popular window managers are in use: xfwm, fluxbox, motif, openbox, …;

- A desktop environment: a GUI-based interface to control and interact with the system, these days more popular than the shell on desktop/laptop PCs; A desktop environment defines the look-an-feel of the desktop, but needs a window manager for drawing all the graphical components;

- Applications.



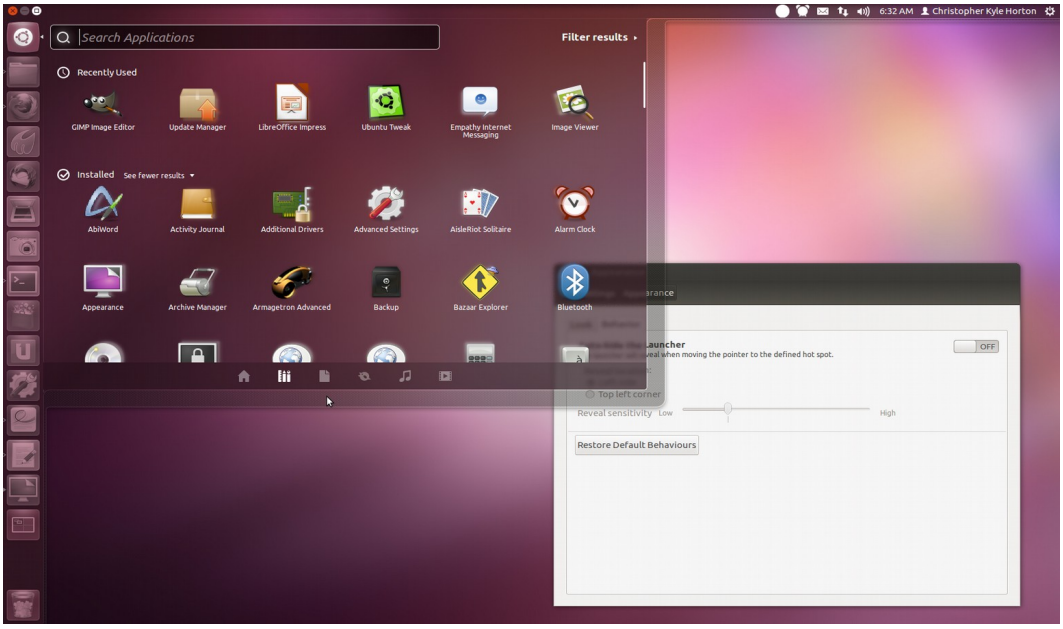

*By Eraserhead1, Infinity0, Sav_vas [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons*

One of the things that Linux-starters struggle with is that there is no such thing as one Linux system: there exist more than hundred of Linux distributions (distro for short). Distros basically differ in the desktop environment, the pre-installed applications, and the software repository (cf. app store) they offer. Several popular desktop environments exist, among which:

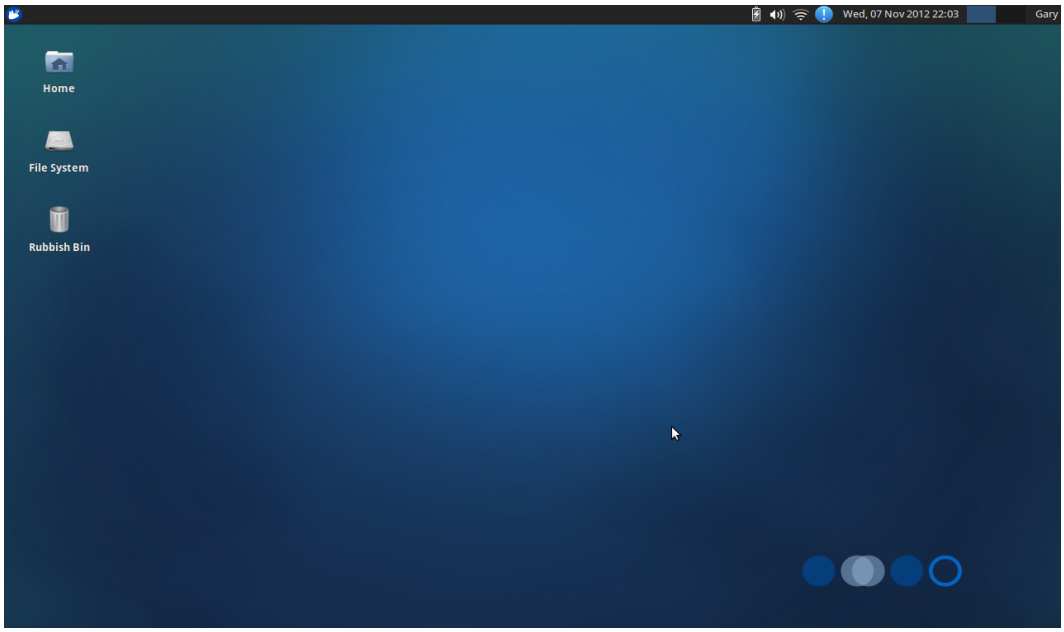
- KDE or Plasma desktop: desktop environment that focuses on an intuitive, easy-to-use and highly customizable workplace; more info can be found on www.kde.org.

- Gnome: focuses since Gnome 2 on usability, cognitive ergonomics and accessibility; written in C / C++ / Python / Vala / Javascript; more info can be found on www.gnome.org.

- Unity: graphical shell for the Gnome desktop environment developed by Canonical Ltd. For the Ubuntu system; written in C / C++ / Python / Vala / Javascript and based on Unity 2D; more info can be found on unity.ubuntu.com.




- XFCE: aims to be a fast and light-weight while still being visually appealing and easy to use; written in C and based on GTK+ toolkit; more info can be found on www.xfce.org.



- LXDE: a very light-weight X11 desktop with a low (lowest?) memory footprint and energy consumption, typically used on resource-constrained systems (e.g. Raspberry Pi) or older hardware; written in C/C++ and based on GTK+ toolkit; more info can be found on www.lxde.org.


Linux distros come in many flavors, some designed for general purpose use, others for dedicated tasks. Only a handful distros are discussed below to give you some examples and references. A simple search on the internet (e.g. http://distrowatch.com/) reveals many others and there even exist documents with guidelines to help you choose the right distro.

- Red Hat Linux : One of the original Linux distribution. The commercial,non-free version is Red Hat Enterprise Linux, which is aimed at big companies using Linux servers and desktops in a big way.

  Free version: Fedora Linux.

- Debian GNU/Linux : A free software distribution. Popular for use on servers. However, Debian is not what many would consider a distribution for beginners, as it's not designed with ease of use in mind.

- SuSE Linux : This distribution is primarily available for pay because it contains many commercial programs, although there's a stripped-down free version that you can download.

- Ubuntu Linux: based on Debian, designed to have regular releases, consistent user experience and commercial support on both desktop and server. There are a lot more official distributions of Ubuntu-based distros: Kubuntu (Ubuntu

with a plasma desktop), Lubuntu (Ubuntu with a LXDE desktop), Xubuntu (Ubuntu with a XFCE desktop), …

- Linux Mint: based on Debian and Ubuntu, using a Cinnamon or MATE desktop, with a mix of open source and proprietary software applications;
- Gentoo Linux: is designed as a highly portable, flexible and modular system that is distributed as open-source code which must be compiled and optimized on the local system; this should result in machine-specific optimizations and speed improvements;
- Arch Linux: follows the KISS principle (keep it simple, stupid) and focuses on simplicity, minimalism and code correctness; it follows a 'rolling release model' meaning that only regular system updates are required to always run the latest release; Arch expects that users have a certain understanding of the system's operation;
- XBMC : a Linux distro for media players and home theater systems;
- OpenWrt : Linux version for routers;
- Kali : a niche Linux targeting computer security, digital forensics and penetration testing;
- ...

*Lab targets: learn to work with the Linux command line, Bash shell, understand Linux file system basics, file permissions and manipulation.*

**Linux terminals**

As mentioned in the introduction, Unix/Linux is a multi-user system. In the old days (before the desktop PC revolution), computer power was scarce and expensive. Therefore, powerful workstations and mainframe computers were build that could be shared between many users. These users connected to the Unix computer with a terminal (keyboard, screen, serial interface). The terminal was only used for getting input, displaying output and errors (*stdin, stdout, stderr!*) and the real computational work was done on the Unix computer.

These days, Linux runs on even the most modest desktop computer and we don't use real terminals anymore to access it. Nevertheless, the idea of a terminal is still valid since you now open an emulated or virtual terminal to get access to the command line. And you can open many of them at the same time. For instance, you can open three terminal sessions and run in each of them the sensor node simulator from the second C programming lab.

Virtual terminals are often indicated by pts/0, pts/1, etc. where 'pts' stands for the slave part of a pseudo terminal device. Pseudo terminals are typically used by graphical terminal implementations such as 'xterm' and in cases of remote logins, like 'telnet' or 'ssh'.

Native terminals, serial device terminals, hardware or kernel emulated terminals on the other hand, are typically indicated by tty/0, tty/1, etc. where 'tty' stands for 'tele-typewriter'.

**Exercise**: *terminals*

Open several kinds of terminals and use the 'tty' command to find out which terminal you are using.

- Find terminal applications (xterm, terminal emulator, …) on your system and run these applications.
- After booting Linux, the root typically starts 6 terminals: tty/1 up to tty/6. You can switch to these terminals by pressing <ctrl>+<alt>+Fi with i=1,..,6. You can switch back again by pressing <ctrl>+<alt>+F7.

**Exercise:** *users and user info*

Multiple user account can (will) exist on your system. Open a few terminals. The following commands help you to obtain information related to the logged on user(s):

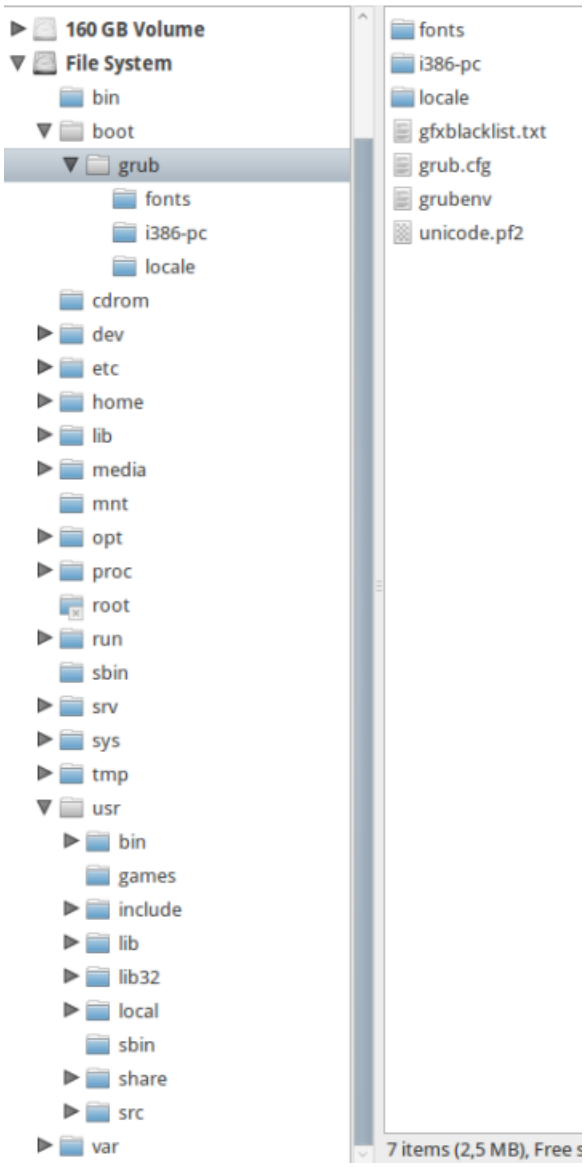
- whoami : shows the user name (id) of the owner of the current login session;

- users : shows who's logged on;
- who : shows who's logged on with some additional info;
- w : shows who's logged on and what they are doing;
- last : prints a history of the last logged on users;
- lastlog : prints the most recent login info of all existing users.

Notice the terminal indication in the info given by some of these commands. Many of these commands can take options to obtain less or more detailed output. Which options are available and info on each of them is available in the info/man pages. Very soon you will learn what info/man pages are and how to access them.

**Linux files and the file system**

There is a saying about Unix/Linux that goes as follows: “In Unix/Linux, everything is a file; if something is not a file, it is a process.” What exactly a process is, will be studied later on this course. We first focus on files, as they seem so important.
A file is a core part of the Linux system. Files may contain readable ASCII text or binary data representing documents, pictures,, ... but also the keyboard and mouse, printers, etc. are represented as a kind of file. Files can be stored on any medium such as the hard-disk, SSD, SD, CD-ROM, … Linux organizes all files - wherever they are stored - in a tree-based structure: the file system. The picture below illustrates this idea.
Everything starts at the root of the tree, denoted as '/'. Files can be organized in 'directories', which are actually 'special' files containing lists and info of other files and directories. It doesn't matter where files are physically stored – on one of the hard-disks, SD or even the network – they are all mapped in directories of the root file tree.
Several types of files exists. Regular files contain normal data, e.g. text, SQL data or executable binary code. Directories are another type of files already explained. Besides regular files and directories, there are a number of special files. For instance, special files in /dev used for input and output, and real time status info about the kernel, processes, I/O, etc. (Domain) sockets are also special files used for inter-process network-style communication with file system access control protection. Named pipes provide another way for inter-process communication without the network-style of communication. More details on these special files will follow later on.




By default, you start in your home directory when you open a new terminal session. Linux is a multi-user system. Linux creates for every user a dedicated home folder. In your home folder, you have all permissions needed to create, rename, and delete files and directories. Outside your home folder, you might not have these permissions.

The most basic commands to navigate through and edit/change the file system tree, are given below.

- pwd : prints the current working directory;
- ls : list the content of the current directory;
    - ls -l : like 'ls' but shows more detailed information, as can be seen in the picture shown below. The '-l' is a command line option or argument. Most

commands have many possible options which are typically explained in the 'info' or 'man' pages of the command. More on that soon.

```
$ ls -l
total 53264
-rw-rw-r--  1 u0066920 u0066920 25165306 Jan 28 16:31 63571.PDF
drwxrwxr-x  4 u0066920 u0066920     4096 Dez  2 18:24 basickernel
drwxr-xr-x  2 u0066920 u0066920     4096 Okt 27 09:47 Desktop
drwxr-xr-x  5 u0066920 u0066920     4096 Dez 29 15:03 Documents
drwxr-xr-x  4 u0066920 u0066920    12288 Jan 22 09:39 Downloads
drwxrwxr-x  4 u0066920 u0066920     4096 Okt  8 13:10 GNS3
```

Notice that in the first column information like 'drwxrwxr-x' is printed. The first character - in the example 'd' – indicates the type of file as discussed earlier. The following convention is used:

- - : a regular file
- d : a directory
- l : a link
- c : a special file
- s : a socket
- p : a pipe
- b : a block device

The other information like 'rwxrwxr-x' defines the permissions on this file, but that will be discussed in more detail later on.

- ls <dir> : list the content of <dir>;

- cd <dir> : change working directory to <dir>;
- cd .. : change working directory to the parent directory of the current directory (in the file system tree);

  Note: <dir> must indicate the location of the directory in the file system tree. This location can be specified as a 'relative' or an 'absolute' path. For example, '/home/u0066920/Documents' is an absolute path: it specifies exactly the location of the directory independent of the current location. The path '../Documents' on the other hand is a relative path: the destination directory is found by first going up one level in the file system tree (to the parent level) and then selecting the 'Documents' directory. Of course, the final destination depends on the current directory, i.e. is *relative* to the current directory. Remark that '..' is a shorthand used to indicate the parent directory and '.' is a shorthand indicating the current directory, e.g. './main.c' denotes the 'main.c' file in the current working directory.

- cd ~ : change working directory to the user's home directory, whatever the current location;

  Note: you can use '~' in the definition of a relative path.

- cd - : return to the previous working directory;
- mkdir <dir> : create a new directory <dir> in the current directory;
- cp <src-file> <dest-file> : copy a file from some source (can include the path to the file) to a destination file;

  Note 1: be aware of using a directory name with a trailling '/', e.g. the result of 'cp lab1 lab2' is different from 'cp lab1 lab2/'. Try it!

  Note 2: the command line option '-R' is used to 'recursively' copy a directory, meaning that all the content – files and sub-directories – are copied.
- mv <src-file> <dest-file> : move a file from some source to a destination;
- rm <file> : remove file;

  Note and warning: again the command line option '-R' can be used to 'recursively' remove files and directories, but be careful with this option because it might delete much more than you intended and there's no way to 'recycle' removed files!
- rmdir <dir> : remove directory;
- tree : list the contents of directories in a tree-like format. Be careful with this command because it can generate very long listings, e.g. if the current directory is the root dir.

**Exercise:** *Linux file system basics 1*

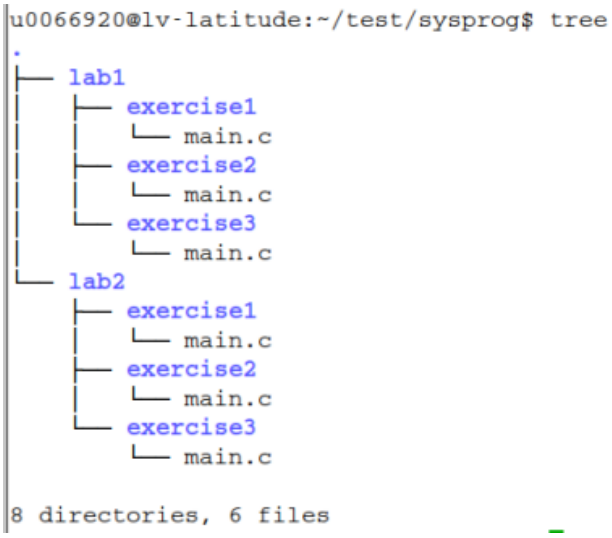
Open a terminal session. Make a directory 'sysprog' in your home folder. Next create a directory called 'lab1' and within this directory create sub-directories for each of the 3 C programming exercises of this lab.

**Exercise:** *command line editor*

A simple command line based text editor is 'pico' or 'nano'. Move to your home directory and start up this editor. Write a simple 'hello world' program in C and save the file as 'main.c' in your home directory.

**Exercise:** *Linux file system basics 2*

Move the 'hello world' main.c file to 'sysprog/lab1/exercise1' and make then copies of this file to the other exercise directories. The result should look like the picture shown below.




In the future, you can create for every lab and exercise sub-directories in 'sysprog' to store the files of your solution. Let's prepare not only this lab session but also the future labs (Assume every lab has three exercises.). Instead of creating all directories one by one, it 's more efficient to copy and rename the entire /lab1/exercise1 directory including 'main.c'. Finally, move to your home directory and check with the 'tree' command that everything is done correctly.

**Info and man pages**

Sometimes you don't remember the exact name of a command or you forgot the meaning of a command line option. A number of information tools will help you out.

- apropos <keyword> : returns a list of commands containing <keyword>;
- man <keyword> : shows the man(ual) pages of <keyword>;

  Note 1: you can get help on 'man' itself by typing 'man man'.

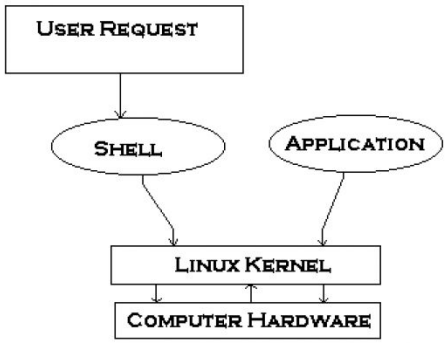
  Note 2: the man pages are divided in 9 sections such as system calls, library calls, shell commands, etc. See 'man man' for details. A keyword may appear in more than one section, for instance the keyword 'write'. In that case, you should mention the section number to select the correct man page, for instance, 'man 1 write' or 'man 2 write'.

  Note 3: you can also search forward and backward within a man page with /<pattern> and ?<pattern>, respectively. Use the 'h' help function in a man page for details.

- info <keyword> : shows the info pages of <keyword>;
- whatis : … well, you just learned about 'man' and 'info'; find out yourself what 'whatis' is doing.

**Bash shell features**

When you open a terminal session, a **shell** is started that allows you to interface with the Linux system by entering commands or executing scripts and programs. In contrast to GUI-driven (graphical user interface) systems like Windows, Gnome or KDE, a shell is a CLI (command line interface) based interaction solution. Several different flavors of shells exist in Linux: 'sh' (Bourne shell), 'ksh' (Korn shell), 'csh' (C shell), 'Bash' (Bourne again shell – combines Korn and C shell features and is often the default shell in Linux), ...




If you don't like the default shell assigned to you, use the 'chsh' (change shell) command to switch to another shell.

We assume that you use Bash in this course. A number of key combinations are worth remember to enhance your efficiency.

- <tab> : command/file name completion;
- <tab><tab> : show all command/file name completion possibilities;
- <ctrl>-d : log out of the current shell session (same as 'exit' or 'logout');
- <ctrl>-c : terminate a running program;
- <ctrl>-l : clear the terminal;
- <ctrl>-a : jump to the beginning of the current line;
- <ctrl>-e : jump to the end of the current line;
- <ctrl>-w : cut from start of line to current cursor position;
- <ctrl>-y : paste;
- history : command showing the command history;
  - !<number> : execute the <number>th command in the command history;
  - !! : execute the first command in the command history;

- <ctrl>-r : search command history (repeat <ctrl>-r to browse through the history);
- cursor up/down: alternative to <ctrl>-r and probably much easier;
- !<string> : redo the last command in the command history starting with <string>.

**Exercise:** *file manipulation tools*
Use info/man pages to find out what the meaning is of the following commands:

- cat
- more : also try 'less';
- tail : also try 'head'
- split
- file
- touch
  - Notice: 'touch' can also be used to create a new file, e.g. 'touch some_file';

Enter the following command in a terminal: 'ls -R > lsout' to list recursively ('ls -R') all content in your home directory (or use another directory if their isn't much content yet in your home directory) and redirect the output (>) to the file 'lsout'. In the next exercise you get more info on I/O redirection.

- Apply 'cat', 'more' ('less') and 'tail ('head') on this file and notice the difference.
- Split this file into 10KB file parts called 'lsout1', 'lsout2', 'lsout3', etc.
- Apply 'file' to a number of files in your home directory and monitor the result.

**I/O redirection**
Most programs require some input and produce some output. Linux commands act the same. By default, input is being given with the keyboard and output is displayed to screen. The source of input of a command line program, is called *standard input (stdin)*. The default output source of a command line program, is called *standard output (stdout)*. To be complete, we also mention the *standard error (stderr)* output where by default error messages of programs are written to. As being said before, by default *stdin* is the keyboard and *stdout* is the screen, but this can be changed if needed. With I/O redirection, you can easily redirect the input or output of a program to another source than 'stdin', 'stdout' or 'stderr'.

The most basic, but very powerful, I/O redirection operators are:

- > : redirect output (stdout) to file;
  - 1> : this is the same as using > ;
  - 2> : redirect stderr to file;
  - &> or >& : redirect stdout and stderr to file;
- < : redirect input (stdin) from file;
- >> : append to file;
- | : pipe the output (stdout) of the left program to the input (stdin) of the right program.

The following examples illustrate these operators. Enter these commands in a terminal session and check out the result yourself.

- pwd > list : 'pwd' prints the result to the file 'list' i.s.o. stdout;
- ls -l >> list : the output of 'ls -l' is appended to the file 'list';
- cat < list : 'cat' uses 'list' as input i.s.o. stdin;
- tree | more : the output of 'tree' is given as input to 'more' (check this out in a directory where the output of 'tree' consists of multiple terminal pages).

Note: '/dev/null' is often used to discard output. For example, 'ls > /dev/null' will discard all output of 'ls'.

**Exercise:** *Linux file system layout*

A picture of a Linux file system layout is already given in the introduction of this lab. This layout is not standardized and will change in time and depends on the Linux distribution. Without having the intention to be complete, we briefly discuss some of the typical directories in the root file system and reveal their purpose. More detailed information can be found in the Linux documentation pages (http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/c23.html) but be aware that the information might not exactly match with your Linux distribution (probably Ubuntu).
Use the commands learned before ('ls', 'pwd', 'cd', … and combine commands, e.g. 'ls -l | more' for viewing long listings) to visit and discover these directories while reading through the text below.

- /bin : contains the binaries of common programs, shared by the system, the system administrator and users; (On a Windows system, this directory corresponds more or less to the folder 'c:\windows'.)

- /boot : contains the startup files required for the boot process, the kernel (vmlinuz), …; (On a Windows system, this directory corresponds more or less to the folder 'c:\boot'.)

- /dev : contains references to all the CPU peripheral hardware, hard-disk partitions, etc., which are represented as files with special properties. Notice the links to stdin, stdout and stderr. Another curiosity is 'null' and 'zero'. Writing to '/dev/zero' is like writing to /dev/null: all data is lost. Reading from '/dev/zero' generates as many null-chars as read. Try 'cat -v /dev/zero' to check it out.

- /etc : contains the most important system configuration files (i.e. data similar to those in the Control Panel in Windows); for example, '/etc/passwd' contains all user info, '/etc/group' contains all groups info and '/etc/shells' contains the names of available shells; use 'cat' to check it out; (On a Windows system, this kind of information is stored more or less in the Windows registry.)

- /home : contains the home directories of the common users; each home directory is only accessible by the corresponding user and the system administrator; (On a Windows system, this directory corresponds more or less to the folder 'c:\users'.)

- /lib : contains kernel modules and shared library files for all kinds of programs needed by the system and the users (in /bin and /sbin); (On a Windows system, this directory corresponds more or less the folder 'c:\windows\system' containing 'dll' files.)

- /mnt : standard mount point for external file systems, e.g. a CD-ROM or a digital camera; (On a Windows system, this directory matches more or less to c:\, d:\, f:\, etc.)

- /opt : a directory reserved for add-on packages and third party software, not part of the default installation or package manager; (On a Windows system, this directory corresponds more or less the folder 'c:\program files'.)

- /proc : is a virtual file system used by the kernel to send information about system resources (kernel data structures) to processes containing information; for example, type 'cat /proc/loadavg' to see the current load average on your Linux system;

- /root : the home directory of the root (system administrator); mind the difference between /, the root directory and /root, the home directory of the root user; (On a Windows system, this directory corresponds more or less the folder 'c:\users\administrator'.)

- /run : a temporary file system used during the boot process;

- /sbin : contains programs for use by the system and the system administrator; (On a Windows system, this directory corresponds more or less to a folder like 'c:\windows\system32'.)
- /srv : used for data needed by network services (e.g. HTTP, FTP, …);
- /sys : is a virtual filesystem reflecting the kernel's view of the system;
- /tmp : a directory reserved for temporary space for use by the system and applications; often cleaned upon reboot, so don't use this for saving any work; (On a Windows system, this directory corresponds more or less to the folder 'c:\windows\temp'.)
- /usr : contains programs, libraries, documentation etc. for all user-related programs; (On a Windows system, this directory corresponds more or less to the folder 'c:\program files'.)
- /var : contains variable files such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, … that persist between two boot sequences; mind /var/log containing among others the Linux system logs; for example, '/var/cache/apt/archives/' contains installation packages installed by 'apt-get'.

**Super user (root)**

The superuser or root is a special user that has all rights and permissions in all modes. The superuser is typically used for system administration tasks. Two commands are worth exploring (see man pages for details):

- su : switch user / become superuser;
- sudo : execute a command as another user / superuser.

Especially 'sudo' is very interesting to use when you want to execute a single command without having the correct permissions.

**Exercise:** *apt-get*

One such command that needs 'root' permissions is 'apt-get'. Apt-get is running the command interface of the package (software) manager. With 'apt-get install', for instance, you can install additional packages.
Check if the 'valgrind' tool is installed on your system (just try to run 'valgrind'). If 'valgrind' is not installed, the system suggests which additional package(s) you need to install. Use 'apt-get install' to do this.
Do the same as above but now for the 'cppcheck' tool.

**File permissions**

Three access rights are given to every file: read (r), write (w) and execute (x) permission. Programs must have execute permission in order to run. These permissions can be set for three groups of users: the user(s), the group, and others. The user is typically the owner of the file; users in the same group of the file owner

get the group-permissions; everybody else gets the others-permissions. The concept of 'group' is explained below – just accept it for now. The permissions of a file can be easily checked using 'ls -l'. Assume the permissions for some file are 'rwxrw-r--'. The first three permissions indicate that the user has read, write and execute permissions on the file, the next three permissions tell that the group can only read and write the file, and the final three permissions show that everybody else can only read the file.
The permissions of files and directories can be changed with 'chmod'. Let's illustrate this command with an example:

```
chmod u+x, o-wx some_file
```

With this instruction, executable permission is added (+) to the 'user' and write and executable permissions are taken away (-) from 'others' of 'some_file'. Use the man pages to obtain more info on 'chmod'.

Note: the user, group and others permissions are sometimes defined in 'binary'. Every permission (r, w, x) for every type of user (user, group, others) can be set with 1 bit of information. In total, 3 sets of 3 bits are needed. Each group of 3 bits is given as a decimal number, ranging from 0 to 7. For example, 'chmod 764 some_file'. The encoding is defined as follows:

| | |
|---|---|
| 0 : --- | 4 : r-- |
| 1 : --x | 5 : r-x |
| 2 : -w- | 6 : rw- |
| 3 : -wx | 7 : rwx |

**Exercise:** *file permissions*
Go to '/sysprog/lab1/exercise1'. Use 'gcc main.c' to compile 'main.c'. If needed, open 'main.c' in 'nano (pico)' and fix the errors. Once 'main.c' is compiled, an executable file called 'a.out' should be created. Check if 'a.out' has executable permissions, and if so, run the program.
Use 'chmod' to set/delete executable permission of the user of 'a.out'. Run the program to notice the difference. Ls

Use 'chmod' to set/delete write permission of the user of 'a.out'. Edit the program to notice the difference.

**User and group**

Groups in Linux are used for better access control. To understand that, consider the following example. Some file 'X' should not be accessible for every user on the system. To better control access to 'X', create some group 'abc' and add all users that are allowed to access 'X' to the group 'abc'. Finally, set the file permissions of 'X' to something like 'rwxrwx---' such that the owner and the group have read/write/execute permissions but other users have no permissions at all.
The '/etc/group' file contains an overview of all existing groups on the system. Print this file to get all groups on your system.
The 'id' command can be used to obtain all groups that you belong to. Try it.

With the 'chown' command it is possible to change the owner and group of an existing file or directory. With the '-R' option you can even recursively apply the changes to an entire directory tree. If you only wish to change the group, you can also use the 'chgrp' command.
Although out of the scope of this manual, we mention the following commands to manage users and groups:

- useradd (groupadd) : create or update a user account (group);
- userdel (groupdel) : delete a user account (group) and related files;
- usermod (groupmod) : modify a user account (group);
- passwd : change the user password;
- chfn : change info on the user (real name and other info, typically used by 'finger').

Obviously, detailed information can be found in the info/man pages.

**Exercise:** *file permissions 2*
Go to your home directory. Open a new file with 'nano' (or 'pico'). Type the following instructions:

```
#!/bin/bash
cd <enter here the path to your main.c code here>
gcc -Wall -Werror main.c -o myprog
./myprog
```

The first instruction tells the shell that this file is a Bash script. The other instructions basically compile the indicated C files and, if compilation was successful, runs the created executable.
Save this to a file called 'run'. The only thing you have to do now to make 'run' an executable script, is to give it executable permissions. Do it and test if you can execute 'run' from whatever location with the instruction '~/run'.

Note: Although Bash scripts are out of the scope of this lab, this exercise illustrates that simple scripts using the commands discussed in these labs can be easily created.

**Miscellaneous**

*Typescripts*
When doing your exercises and experiments, you might have felt the need to save a log of a terminal session for future use. The 'script' tool might be exactly what you are looking for. An example illustrates how it works:

```
script  my_script_file

<type here the instructions you want to typescript>
```

<ctrl>-d

The first instruction starts a new typescript: the logged data will be sent to the file 'my_script_file'. If you omit the file, everything will be logged to a default file 'typescript'. After this instruction, you get a command prompt as if nothing happens. Type all instructions you want to typescript. When finished, type '<ctrl>-d to 'logout' the typescript session. You can view the typescript in any text editor, but then you will see lots of ugly 'control characters'. A simple way to get rid of these control characters is using 'cat my_script_file'.

*Command aliases*

It's possible to create shortcuts or synonyms for existing commands. These are called 'aliases'. To get an overview of which aliases are set in your shell, type 'alias'. You might get a similar output as the one depicted below. You can see that an alias 'll' is defined as a shortcut for the 'ls' command with the options 'alF'. This means that you can use 'll' as a shell built-in command. The shell will expand 'll' to 'ls -alF' and execute this instruction.

```
$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')
"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
$ ll
total 72
drwxrwxr-x  3 u0066920 u0066920  4096 Jan 19 13:57 ./
drwx------ 55 u0066920 u0066920 53248 Jan 19 11:32 ../
drwxrwxr-x  4 u0066920 u0066920  4096 Jan  7 18:22 sysprog/
-rw-rw-r--  1 u0066920 u0066920     0 Jan 19 13:57 typescript
```

An alias can be easily created. For example, the command:

alias cdlab='cd ~/sysprog/lab1'

creates an alias 'cdlab' to easily jump from wherever in the file tree to your exercise directory. You can use 'unalias cdlab' to destroy the alias.
There is one drawback: if you exit your login session, your aliases are lost. If you want to keep your aliases permanently, you should store them in a login script file, for instance, the hidden file '.bash_aliases' file in your home directory. This script file is executed each time you login to a new session and, hence, your aliases will be activated again.

## Summary: list of commands

- alias
- apropos
- apt-get
- cat
- cd
- chmod
- chgrp
- chown
- chsh
- cp
- file
- head
- id
- info
- last
- lastlog
- less
- ls
- man
- mkdir
- more
- mv
- nano
- pico
- pwd
- rm
- rmdir
- script
- split
- su
- sudo
- tail
- touch
- tree
- tty
- unalias
- users
- w
- whatis
- who
- whoami