Data Storage Infrastructure - Introduction to the Laboratory

27 May 2021

Contents

Executive summary

The Data Storage Infrastructure lab sessions are designed to complement the lectures. For this reason, each lab session consists of a **previous work** and a **work to be done during the session in the laboratory**. As each lab session lasts 3 hours, the previous work is meant to require a maximum of 4.5 hours. After the lab session there is no more assessable work, although the student can complete and/or review the work done to improve the learning process.

It is recommended to do the previous work and the work in the lab session **in pairs**. Although it will not be compulsory *unless the groups are full*, doing the work in pairs enables faster progress and also improves the learning process. In fact, it is less common to get *stuck* when working in pairs. Moreover, it must be considered that both, receiving an explanation from our partner and/or explaining a concept to our partner could give us a better understanding of the matter.

During the whole semester students are going to have **the support of laboratory lecturers** to complete the assigned tasks, both during face-to-face and/or virtual tutoring (especially for the previous work) and in the laboratory during the practical sessions.

The NETinVM virtual machine is going to be used in the lab sessions.

Previous work

Why do we have previous work?

> The previous work prepares the student for the lab work. It is worthwhile reviewing both theoretical and practical concepts before attending the laboratory session because the lab sessions complement the lectures. Moreover, lab sessions also complement the lectures on topics such as the shell, scripts, or system administration. Therefore, the previous work must be fully completed to make the most of the lab sessions.

How is the previous work done?

> We use the specified virtual machine. By working in this way, it is possible to have the same work environment as in the laboratory. VMware Workstation Pro works on Windows, Mac OS, and Linux, so it is not necessary to change any setting in the personal computer.
>
> If neither of the two students in a lab pair has a computer that can run the virtual machines,

laboratory lecturers can suggest alternatives for preparing the lab sessions.

What if previous work is not done?

You will experience difficulties. Mainly because you are going to spend most of the lab session recalling theoretical concepts and solving simple doubts that should not require help from the lecturer. Therefore, experience tell us that it is highly likely that you are not going to have time to finish the lab work.

What must be uploaded?

In each lab session script, it will be clearly stated what must be uploaded. Sometimes it will just be the answers to several questions related to the lectures or to the preparation of the lab session. In other cases, you must complete small exercises that prove that previous work has been reviewed. All work must be in a single file. If multiple files must be included, they can all be packed into a ZIP or a compressed TAR file.

How is the previous work uploaded?

There will be a task in the Virtual Classroom tagged in Spanish 'entrega del trabajo previo', which means 'upload of previous work'. The deadline for this task is the day before the laboratory session.

What happens if I do not upload the task?

A one-point penalty will be subtracted from the lab session mark. However, the main problem will be that it is going to be difficult to make the most of the lab session. Therefore, it is quite common to obtain a much lower mark than the mark that would have been obtained if the previous work had been done.

Session in the lab

How does a lab session work?

Lab exercise tasks are divided into several exercises. Even when the session’s main goal is to carry out a single programme, there are several intermediate exercises.

When an exercise is finished or a checkpoint is reached, the laboratory lecturers must be asked to check it. The lecturer will ask some questions to be sure that both students understand what they have done, and if there are mistakes, the lecturer will suggest how to correct them. Students, after correcting them, should call the lecturer again to make a quick check.

Failing while checking the work does not result in points being subtracted, since making a mistake helps learning. Thus, a group that at the end finishes the lab session correctly, even if they have had to correct some mistakes during the session, will receive the maximum mark (10). It must be considered that groups that continuously make mistakes will advance more slowly and will not be able to reach the end of the lab session - and so marks will be lower.

The previous work has been designed to help students to complete the tasks in the lab. If a group has not done the previous work (or they have done it but do not understand it very well), they will naturally have more doubts, will make more mistakes, and are likely to advance more slowly.

Why work in pairs? Is it mandatory?

As already explained, doing the work in pairs enables faster progress and improves the learning process. This happens because, when working in pairs, a student who does not understand something may find that their colleague does understand, and therefore it is less common to get stuck. Moreover, having to explain something forces us to try to understand it better. In the same way, a colleague’s explanation may be more effective than that of the lecturer. This is because a colleague has just faced the same problem and has a similar background, therefore, they may have a better understanding of what is difficult about the question or exercise.

It will not be compulsory to do practices in pairs *unless the groups are very full*.

How is work assessed in the lab session?

The laboratory lecturer mainly considers the following points:

- How far have the students reached in the lab tasks?
- Are the various exercises solved adequately?
- Have the students been able to properly answer questions made by the lecturer. (It must be noted that the lab lecturer will ask both students, and consequently, the marks of the two members of the group may be different).

To do this, the lecturer will consider both the notes taken during the session and the material uploaded at the end of the session by the students.

As at the end of the session, requests for checking the tasks tend to accumulate, and the final parts may remain unchecked by the lecturer. In this case, the lecturer would take into consideration the materials of these final unchecked parts that have been uploaded by students. This is only for the final parts of the sessions if it has not been possible to check them before leaving the lab.

Notice: *Uploaded material will not be considered unless it has been reviewed during the session by the laboratory lecturer.*

Do I have to write a lab report?

No. It is only needed to compile all the materials requested in each lab session: screenshots; answers to questions; commands used; programme lists; etc. Nevertheless, it is a good idea to complete these with notes that can help the lab lecturer mark the exercise. Moreover, these notes can be helpful later when studying for the exam or using the contents in another subject, final degree project, or in professional activity.

What must be uploaded?

A ZIP file that contains the information requested in the lab script: screenshots; answers to questions; commands used; program listings; and so on. (In some cases all this information can be included in a single text or PDF file. In that case, uploading this file will be sufficient).

How is the work submitted?

There is a task in the Virtual Classroom (Aula Virtual) labelled 'entrega del trabajo de la sesión de laboratorio' (lab work submission) which will include a deadline for uploading the file, the lab session (with some few additional minutes).

Why cannot I improve my grades later?

Because the lab sessions are designed so that the previous work is done **before** the lab session and the lab session is designed to be completed on schedule.

Could my marks be kept for later courses?

In this subject, the lab marks are not saved from one course to another year. The reason is that lab sessions are designed to complement the lectures and it makes sense to do them at the same time and while lectures and problems are being studied. Moreover, the laboratory work is not evaluation by examination, which would be considered necessary to be able to save the marks for later courses.

Further work

What is it for?

> It is useful to complement the learning process. Students can review and complete the work done in the laboratory. In addition, by having the virtual machines, it is possible to delve into all those additional contents that they are interested in.

Is it mandatory?

> No, but it is advisable.

Is this considered for the marks?

> No. However, reviewing and completing laboratory work helps in the learning of theoretical concepts.

Virtual machines

Why do we use virtual machines?

> Because students can prepare lab session in advance, review, and complete them even after the lab session. In addition, they enable students to delve into those topics that are most interesting to them. It is also possible to test many things that will help students in the learning process.

Do I have to use the virtual machines of the subject?

> Yes. Although it is true that many lab sessions could be done on any Linux machine, using our own virtual machines avoids compatibility issues caused by using other versions of the tools or using a different system configuration.

Do I need a USB external hard drive? Why?

> It is highly recommended, but not essential. The idea is to store a copy of the subject's virtual machines on the USB hard drive and run them from there. This allows you to have the previous work when you arrive at the lab (since you use the same virtual machine hosted on the USB disc) and you can take with you the work done at the end.
>
> If you do not have a USB hard disc, you can use the copy of the virtual machines that are in the 'D:' drive of the laboratory computers. However, the work done will be lost when the virtual machine is shut down, so it should be copied first. In some lab sessions this is simple (for example, when making a program); but in others it is not possible (for example, when changes are made to the system in the systems administration lab session).
>
> **Notice:** *It is not possible to use a USB memory stick, it must be a hard drive because writes to USB sticks are usually too slow to run virtual machines fluently.*

Can I use my laptop?

> Yes. However, you need to run the machine on your laptop before attending the lab and be sure that it runs smoothly enough. Generally, any system with a processor that supports virtualisation and 4 GB of RAM should be powerful enough.

If I use my laptop, do I need an external hard drive?

> No, except as a backup or in case the laptop does not work during one of the sessions.

DSI - Laboratory Session 0: Virtual Machine Installation and First Commands

27 May 2021

Contents

Goals

In this first session, the Linux virtual machine that will be used in the laboratory sessions of the subject will be introduced. The goal is that students become familiar with the Linux environment and learn the basic commands for manipulating files and directories from a terminal or console.

Introduction

A virtual machine is software that emulates a computer and, therefore, can run programs as if it were a real computer. System virtual machines (such as the one that will be used in this lab) simulate hardware that can be configured from the program that runs the machine, enabling a new operating system to be installed, which may be different from the host system. The virtual machine that will be downloaded will enable testing on any computer without having to change its configuration, since the virtual machine will not affect the operating system installed on the host.

The GNU/Linux operating system is a free operating system. This means, among other things, that it is free and that the source code of the system can be provided. In this way, it is possible to study and modify the system to fix possible bugs or introduce improvements. Linux is understood to be the kernel (the program that starts taking control of the machine) of the operating system. But this kernel by itself does not form an operating system. It must be supported by more free programs, developed by the Free Software Foundation, and known collectively as the GNU project. It is more appropriate to refer to the system as GNU/Linux (although sometimes only Linux is used).

Since there are many free software programs (such as word processors, graphics, programming environments, and spreadsheets), there are different Linux systems, and this is when the concept of distribution appears. A distribution is nothing more than a Linux kernel plus a selection of free programs together with a particular management of its installation. Some of the most used Linux distributions include RedHat, Debian, Ubuntu, OpenSUSE, LliureX, and Fedora

Required downloads

To attend lectures or the laboratory with the laptop, it is necessary to have downloaded the virtual machine with which the sessions will be carried out (3.4 GB):

> https://informatica.uv.es/~carlos/ns/netinvm/netinvm-kvm_2020-07-15_vmware.zip

NETinVM is a virtual machine image from VMware that provides the user with a complete computer network. NETinVM can be used to learn about operating systems, computer networks, and system and network security.

An in-depth description of the machine NETinVM can be found at:

> https://informatica.uv.es/~carlos/docencia/netinvm/netinvm.html

In addition, an introductory lab session ('Introducción a NETinVM ', in Spanish) is available. It was designed for students of 'computer security' in the computer engineering degree course and can be used to deepen your understanding of the management of NETinVM and as a reference manual.

The virtual machine works with software that executes it. In this case it is VMware, which is free in its VMware Workstation Player version (available for Windows and Linux) and sufficient for these lab sessions. Nevertheless, it is recommended to install the WMware Workstation Pro version, available for UV students at https://software.uv.es with a campus license (license paid by the university and free for students). Mac OS X users can use VMware Fusion and we have a campus license (there is no free version of this software).

To run this virtual machine, you need a version higher than 15 (Workstation), but it is always recommended to install the most recent version. It is recommended to follow the following procedure:

- Go to https://software.uv.es and *login* (green box on the right).
- In 'Search' (top right), write 'Workstation' (or 'Fusion' in the case of Mac OS X).
- Follow the 'VMware Workstation' (or 'Fusion') link for the host operating system.
- Click on the icon in the form of a spider web, available both in 'Access' and in 'Code', within 'License data'. This is how to access the university section of the VMware store.
- Make a purchase (must show a cost of € 0) of the chosen software (eg: VMware Workstation 15).
- The download of the program is made from the store itself.
- Once downloaded, the software must be installed as an administrator.

> **Note**
>
> It is possible that after installation, the program may notify that there is a new version (e.g., 15.1). This is normal and it is recommended to install these updates.

Booting the virtual machine

To boot the virtual machine, VMware Workstation must run first. To launch the virtual machine, the option 'Open a Virtual Machine' must be selected, and the file with the extension '`.vmx`' must be chosen. This file is found inside the folder where the '`.zip`' file corresponding to the Linux virtual machine has been unzipped. When this document was written, it was named 'NETinVM KVM 2020-07-15.vmx' but you may find a newer version. Once the machine is loaded, it must be started by selecting the option *Start up this guest operating system*.

At the first start it will ask if the machine has been moved or copied. '`I copied it`' must be selected.

In the virtual machine

- The non-privileged user is named **user1**.
- The password for 'user1' and the administrator (user 'root') is included in the clipboard (klipper).

Once the machine has started, links are available through icons in the lower left. The 'K' symbol, bottom left, is the start menu. From there, it is possible to launch applications, configure the system, review the history, close the session, etc.

How to be more efficient with the shell

One of the characteristics of the shell is that commands and their parameters must be typed. This can become time consuming with commands with complex syntax and many parameters. Fortunately, this task can be made easier using some key combinations that can be custom configured, but which, by default, are those described below.

Editing the current line

Tabulator
: By pressing the tab once, the word in progress (name of the program, of a file...) is autocompleted. Pressing twice displays a list of candidates for autocompletion.

Home (or control-a)
: Go to the beginning of the line.

End (or control-e)
: Go to the end of the line.

Left arrow (or control-b) and right arrow (or control-f)
: Move one character left or right.

Control-k
: Kill to the end of the line and save the content.

Control-y
: Yank the last thing deleted.

Using command execution history

Up arrow (control-p) and down arrow (control-n)
: Go to the previous (up) or later (down) command.

Control-r
: Search for a previous command based on a string.

First commands

Exercise 1: `man`

The Unix/Linux `man` command is a system manual, which is useful to avoid memorising all commands and/or all their options. For example, to find what a command does and what its options are, just type:

```
man <command>
```

to invoke the man page for the specified command. If the goal is to discover how to operate with the man command, the system must be asked with a 'man' of the 'man' command:

```
user1@base:~$ man man
```

To exit and return to the shell, just press **q**. Generally, there are different sections within the manual, and there may be information about the same command in different sections.

Tasks to be completed

Execute:

```
man passwd
```

The first page of the manual should be accessed and read.

To be uploaded

- Explanation, in one sentence, about the usefulness of this command.

Exercise 2: Some basic commands

The Linux command list is very long. This laboratory session it is going to deal with some basic commands to operate in the system and handle files. In the first two laboratory sessions, more advanced commands will be used with the intention of building command lines to search and manage data.

Tasks to be completed

Execute the following commands individually, seeing what their utility is. In the last two cases an order must be passed as an argument:

```
clear
date
ls
pwd
history
clear
whatis <order>
whereis <order>
```

To be uploaded

- A single sentence explanation of what each command is used for.

Exercise 3: Users

The Linux system is a multi-user system, so that different users can be working on the same machine at the same time. The `who` command shows which users are currently connected to the machine. The command `whoami` shows which user is using the terminal. Additionally, the `id` command shows the identifier of the user and the group to which it belongs.

Tasks to be completed

Execute the following commands individually to discover their purpose:

```
who
whoami
id
```

To be uploaded

- Number of users connected to the system.
- User identity.
- User identifier number, and the group to which it belongs.

Generated on: 2021-05-27 15:02 UTC. Generated by Docutils from reStructuredText source.

DSI - Laboratory Session 1: Command Line

28 May 2021

Contents

Goals

The main goal in this first session is that students become familiar with basic commands for manipulating files and directories from a terminal. To do this, the tasks in the previous work section must be completed before the laboratory session, and the remaining activities must be completed in the laboratory.

Previous work

Basic commands from the terminal

A terminal on *base* at NETinVM must be executed. This can be done in several ways:

- Click on the icon with the shape of a black screen that appears at the bottom left.
- Use the 'Application Launcher' to launch the 'Terminal' or 'Konsole'.

Once opened, you will see the prompt with the name of the user, the machine name and the working directory: '`user1@base:~$`'. The orders that are written on the keyboard will appear on the right of this prompt.

Unix/Linux commands have the following structure:

```
command [options] <file list>
```

Options are not strictly necessary. In some cases, the file list is not needed either. Options are expressed with a hyphen before the option, for example: '`ls -l`'. If you want to use more than one option, they can be joined together or use separate hyphens. For example: '`ls -l -a`' is equivalent to '`ls -la`' or '`ls -al`'.

Simple commands return a value that represents their exit status. If there has been no error, the return value is 0, otherwise it will return a value other than 0. It is possible to see at any time what the returned value is by running the command '`echo $?`'

```
user1@base:~$ ls -l
total 12
drwxr-xr-x 2 user1 user1 4096 Jul 15 16:04 Desktop
drwxr-xr-x 2 user1 user1 4096 May 19 15:51 Documents
drwxr-xr-x 2 user1 user1 4096 May 22 13:31 Downloads
lrwxrwxrwx 1 user1 user1   12 May 19 17:20 netinvm -> /srv/netinvm
lrwxrwxrwx 1 user1 user1   14 May 19 17:20 shared -> netinvm/shared
user1@base:~$ echo $?
0
user1@base:~$ ls -l itdoesntexist
ls: cannot access 'itdoesntexist': No such file or directory
user1@base:~$ echo $?
2
user1@base:~$
```

Warning

If you want to interrupt the execution of a command, you must use the key combination 'Ctrl-c'. For example, using '`sleep`' as an example of a command to interrupt:

```
user1@base:~$ sleep 30
^C
user1@base:~$
```

(The '^C' symbol is displayed by the shell when 'Ctrl-c' is pressed).

A common mistake is to use 'Ctrl-z' (as it is used as 'undo' in many environments). The problem is that 'Ctrl-z' does not end the execution; instead, it sends the command to background. If this happens, the way to end the command execution is to use the '`kill`' command like this:

```
user1@base:~$ kill %
```

If it is not clear, the '`jobs`' command lists the jobs in the background. You can use '`kill %`' repeatedly until all jobs have been ended.

Previous exercise 1: directories

To create a new directory from the command line, you must execute:

```
mkdir <directory name>
```

To delete a directory, you must execute:

```
rmdir <directory name>
```

This command deletes a directory providing the permissions are correct and the directory is empty.

To change directory:

```
cd <directory name>
```

In Unix, the directory '.' is the current directory, and '..' is the directory above the current directory. In this way, the command '`cd .`' does not make any change, since we go to the current directory. However, the command '`cd ..`' goes up one level in the directory tree. The command '`cd`', run without arguments, makes the current directory the *home* directory, which is the directory where the session starts. It is always possible to establish in which directory you are by using the command '`pwd`'. The command '`ls`' shows the contents of the current directory.

Tasks to be completed

- Create a new directory with the name 'test_P1'.
- Change to this new directory. Check that it has been done with the command '`pwd`'.
- Create a new directory with the name 'tmp1'.
- Create a new directory with the name 'tmp2'.
- Change to 'tmp1'; to do this execute '`cd tmp1`'.
- Now try to change to 'tmp2' with the command '`cd tmp2`'.
- Explain why it is not possible.
- Write a command to go to 'tmp2' from 'tmp1' with just one execution of the command '`cd`'.
- Go back to the *home* directory, from where 'test_P1' was created, and try to delete it with the command '`rm test_P1`'. Why is it not possible to delete it?
- Use '`man rm`' to find the correct option and so be able to delete the directory 'test_P1' directly. The command '`rm`' is like '`rmdir`' but, by default, it works with files. However, there is an option that allows recursive deletion. (Warning! special care must be taken with this option.)

To be uploaded

- Command that changes to 'tmp2' from 'tmp1' with just one execution of the command '`cd`'.
- Command to delete the directory 'test_P1' and its subdirectories.

Previous exercise 2: files

Although you can use text editors or input/output redirection, the easiest way to create a file called 'file' is to use the command '`touch file`'. Although the function of '`touch`' is to enable changing the time attributes associated with the file (e.g. the last time the file was accessed); if the file does not exist, it creates an empty one.

The command '`ls`' with the appropriate options shows information about the files that the current directory contains:

- `-l`: detailed information about files
- `-a`: shows all files, even hidden ones
- `-X`: shows files sorted by name
- `-r`: shows files sorted by name, but in reverse order
- `-t`: shows the files sorted by date and time
- `-d`: if an argument is a directory, only its name is shown
- `-F`: distinguish between files and directories
- `-1`: displays the information in one column

The command '`cp <src_file> <dst_file>`' is used to copy the file '`<src_file>`' into a new file with the name of the destination file '`<dst_file>`'. If the full path is not specified in the file names, it will be assumed that the files are in the current working directory.

The command '`cat [<file_list>]`' shows the content of the files on the screen, without pause. It is better to use the command '`more`', for viewing text files. This command pauses at the end of each screen until the space bar is pressed. Even better is the command '`less`', that enables the use of the arrows and advance and rewind page keys to navigate through the file.

Tasks to be completed

- Execute '`cd`'.
- Execute '`cp /etc/passwd Documents`'.
- Execute '`less Documents/passwd`'.
- Show all files and directories in the *home* directory without details.
- Show the content of 'Documents' (without changing to that directory).
- Show the content of the directory *home* ordered by name in reverse order.
- Show the content of the directory *home* ordered by date and time.

To be uploaded

- Command used in the tasks where the commands are not included (the last 4).

Previous exercise 3: permissions

As an example, if '`ls -al`' is executed in the *home* directory, information like this will be displayed:

```
total 1460
drwxr-xr-x 12 user1 user1    4096 Sep  1 10:23 .
drwxr-xr-x  3 root  root     4096 May 19 15:45 ..
-rw-------  1 user1 user1      15 Aug 31 17:33 .bash_history
-rw-r--r--  1 user1 user1     220 May 19 15:45 .bash_logout
-rw-r--r--  1 user1 user1    3526 May 19 15:45 .bashrc
drwxr-xr-x  6 user1 user1    4096 Sep  1 10:25 .cache
drwxr-xr-x 17 user1 user1    4096 Sep  1 08:56 .config
drwxr-xr-x  2 user1 user1    4096 Jul 15 16:04 Desktop
-rw-------  1 user1 user1      60 May 19 16:22 .directory
drwxr-xr-x  2 user1 user1    4096 May 19 15:51 Documents
drwxr-xr-x  2 user1 user1    4096 May 22 13:31 Downloads
```

```
drwx------  5 user1 user1    4096 May 25 10:57 .gnupg
-rw-r--r--  1 user1 user1     375 Sep  1 08:56 .gtkrc-2.0
drwxr-xr-x  3 user1 user1    4096 May 19 15:51 .kde
-rw-------  1 user1 user1      54 Sep  1 10:23 .lesshst
drwxr-xr-x  3 user1 user1    4096 May 19 15:51 .local
drwx------  5 user1 user1    4096 May 19 16:25 .mozilla
lrwxrwxrwx  1 user1 user1      12 May 19 17:20 netinvm -> /srv/netinvm
drwx------  3 user1 user1    4096 May 20 19:23 .pki
-rw-r--r--  1 user1 user1    1010 May 19 17:21 .profile
lrwxrwxrwx  1 user1 user1      14 May 19 17:20 shared -> netinvm/shared
-rw-------  1 user1 user1      49 Sep  1 08:56 .Xauthority
-rw-------  1 user1 user1 1405039 Sep  1 10:23 .xsession-errors
```

The information displayed is grouped in columns (separated by spaces). The first column contains the permissions. The first character of the first column gives information about the type of object, '**d**' for directory, '**-**' for file, '**l**' for symbolic links (an object which is a link to another - like a Windows direct access) and so on. The encoding of the rest of the characters corresponds to the different permissions and its interpretation is as follows:

```
drwxrwxrwx
d********* object type (d, l, or -)
*r******** read for the object owner (r or -)
**w******* write for the owner (w or -)
***x****** execution (search if directory) for the owner (x, s, S, or -)
****r***** read for the group (r or -)
*****w**** write for the group (w or -)
******x*** execution/search for group (x, s, S, or -)
*******r** read for other users (r or -)
********w* write for others (w or -)
*********x execution/search for others (x, t, T, o -)
```

If one of these positions contains '-', the object does not have the permission indicated by the corresponding position. For example, a file with the permissions '**-rwxr-xr--**' can be read, written and executed by the user, read and executed by the group, and only read by the rest of the users. It is important to note that files in Linux are not executable files just because they have a specific name or extension (**exe** or **com** extensions in Windows), but because they have permission to be executed. The rest of the columns show information such as the name of the user, the group to which it belongs, the size of the object in bytes, the date and time of the last modification, and, finally, the name of the file.

Changing permissions

It is possible to change the permissions of files and directories if you are the owner or the administrator (*root*):

```
chmod <who><action><permissions> <file_list>
who:    u -> user
        g -> group
        o -> other
        a -> all
action: + -> add permissions
        - -> remove permissions
        = -> assign permissions
permissions:r -> read
        w -> write
        x -> execution
```

The difference between '=' and '+/-' is that '=' establishes a set of permissions, while '+/-' preserves the existing permissions (other than the set being added or removed).

Tasks to be completed

- Execute '`cd Documents`'.
- Check the files, along with their permissions, that the current directory '`ls -l`'

contains.
- Now run '`chmod g=w passwd`'. What are the permissions now?
- Modify the permissions so that the other users of the group can execute the 'passwd' file in addition to the permissions they already had.
- Modify the permissions so that the owner of the 'passwd' file cannot read it. Check that it cannot be read with '`less passwd`'.

To be uploaded

- Command used in the tasks where the commands were not included (the last 2).

Previous exercise 4: move, rename, and delete files

In the command '`mv <src_file> <dst_file>`' both source and destination can be files or directories. It is also possible to use '`mv <file_list> <directory>`' to move the list of files to that directory.

The command '`rm [options] <file_list>`' deletes files. But, depending on the options, it can also delete a complete directory tree.

Options:

`-i`
: Interactive mode. Permission is sought before performing an operation.

`-r`
: Recursive mode. If applied to a directory, it deletes the directory and all the elements contained in it.

Warning

We must be extremely cautious when using this command while being 'root' - because any file can be deleted from the system.

Tasks to be completed

- Copy the directory 'Documents' and all its contents to the directory 'Documents2' with a single '`cp`' command.
- Rename the file 'passwd' that is in 'Documents2' as 'keywords'.
- Rename the directory 'Documents2' to 'Doc2'. Check the contents of 'Doc2' and that 'Documents2' does not exist.
- Delete the 'Doc2' directory and all its contents with a single command. The command must ask for confirmation before each deletion.

To be uploaded

- Commands used in the tasks.

Previous exercise 5: using wildcards

Linux supports the use of wildcards to search for files and strings. The asterisk '*' stands for zero or more characters. In this way, a command like '`ls /etc/o*`' will generate a list of files in '/etc/' that start with the letter 'o' and the directories, including their content, that start with the same letter. (An object named '/etc/o' would also match the wildcard).

The wildcard '?' stands for a single character. You can also specify a single character, or a range of characters, using a pair of square brackets ('[]'). Thus '`ls [a-d]*`' will generate a list of files and directories starting with 'a', 'b', 'c' or 'd'. Similarly, '`ls *[a,z]`' will generate a list of files and directories ending in 'a' or 'z'. It is also possible to use '!' as negation. So, the command '`ls [!a]*`' will show all files/directories that do not start with 'a'.

To test the different possibilities, it is recommended to go to a directory with many files, such as '/usr/bin', and then practise wildcard substitution to find different sets of files.

Tasks to be completed

- Execute '`cd /usr/bin`'.
- Type a command that displays all files/directories whose penultimate character is '.'. That is, files like 'ri2.3', 'ruby2.3', etc.
- Type a command that only shows all the files/directories that do not start with a letter, for example: '411toppm', '7z', etc.
- Type a command that only shows all the files/directories that have three initial characters, then the character '2', and then any number of characters, for example: 'pdf2ps', 'sha256sum', etc.
- Type a command that shows the files that start with 'pod', but are not followed by a '2'. For example, 'podchecker' should be displayed, but not 'pod2html'.

To be uploaded

- Commands used in the tasks.

Work in the laboratory

Input/output redirection

Most Linux commands read data from the standard input, write the results of their operation to the standard output, and display error messages in the standard error. By default, the standard input ('stdin') is the keyboard and the standard output and error ('stdout' and 'stderr') are the terminal.

Before a command is executed, its input, output, and error can be redirected using '<', '>', and '>>' as follows:

```
command < file
```

Before executing 'command', the shell changes the standard input so that 'command' reads from 'file'.

`command > file`
: Before executing 'command', the shell opens 'file' for writing, creating it if it does not exist and truncating it if it exists; and redirects the standard output of 'command' to 'file'. For example:

```
user1@base:~$ date > date.txt
```

'date.txt' contains the output of the command 'date'.

`command >> file`
: This is similar to the previous one. The difference is that in this case if 'file' exists it is not truncated, but the output of 'command' is appended to the end of 'file'. For example:

```
user1@base:~$ date > date.txt
user1@base:~$ ls >> date.txt
```

'date.txt' contains the date in the first line and then the current directory listing. Instead, if you now run:

```
user1@base:~$ ls / > date.txt
```

the content of 'date.txt' becomes exclusively the listing of the '/' directory.

`command 2> file`
: Before executing 'command', the command interpreter opens 'file' for writing, creating it if it does not exist and truncating it if it exists; and redirects the standard error of 'command' to go to 'file'. For example:

```
user1@base:~$ gcc 2> errors.txt
```

'errors.txt' contains the 'gcc' error messages.

`command &> file`
: Redirects both the output and the standard error from 'command' to 'file'.

Exercise 1: redirecting input/output

Using the previous information, different redirection tests must be carried out.

Tasks to be completed

From the shell, perform the following actions:

1. Execute '`ls -l /`' so that the result is stored in the file 'ls.txt'.
2. Execute '`ls -a`' so that the result is added to the previous content of 'ls.txt'.
3. Execute '`gcc`' so that the error messages are stored in the file 'gcc-err.txt'.
4. Check that the content of the files 'ls.txt' and 'gcc-err.txt' is as expected.

To be uploaded

- Copy of the execution of the commands, their messages, and results. (Just copy and paste text from the terminal).
- Copy of the generated files.

Commands that can be used as filters

In Linux, there are many commands that are designed so that they can be used together to search, filter, or select data. In this section, some of these are presented and their true potential will be seen in the section 'Commands linked by pipes '.

The characteristics that allow these commands to be combined in many different ways are:

- Each command performs a specific task, such as copying, sorting, and looking for patterns.
- All are intended to take data from their standard input and provide results through their standard output. (Hence, they can be seen as **filters**).

The *cat* program

```
cat [options] [file]...
```

This program concatenates files (or standard input) and sends them to standard output. For example, supposing that there is a text file called 'hello', with the content 'Hello!', and another text file called 'goodbye', with the content 'Goodbye!':

```
user1@base:~$ cat hello
Hello!
user1@base:~$ cat goodbye
Goodbye!
user1@base:~$ cat hello goodbye
Hello!
Goodbye!
user1@base:~$
```

Thus, it is possible to process several files as if they were one. Indeed:

```
user1@base:~$ cat hello goodbye > hello_goodbye
user1@base:~$ cat hello_goodbye
Hello!
Goodbye!
user1@base:~$
```

The *head* program

```
head [options] [file]...
```

This program prints the beginning (10 lines by default) of each file in the standard output. For example:

```
user1@base:~$ head -n 2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
user1@base:~$
```

shows the first two lines of the file '/etc/passwd'. If the number of lines is negative, '`head`' shows all but the last N lines. For example:

```
user1@base:~$ wc -l /etc/passwd
38 /etc/passwd
user1@base:~$ head -n -35 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
user1@base:~$
```

The '`wc`' program with the '`-l`' option counts the lines it reads from the input files, or from the standard input, if no file is specified. In this case, it counts the lines in the '/etc/passwd' file.

With the '`head`' command, as the number of lines is negative, the last 35 lines of the file are removed from the output. This option is useful when a file contains additional information at the end that should not be processed.

Exercise 2: using *head*

You want to show the first eight lines of the file '/etc/bash.bashrc'.

Tasks to be completed

1. Type the command that displays the requested information.
2. Check that it works correctly.

To be uploaded

- Copy of the command execution, including the information displayed. (Just copy and paste text from the terminal).

The *tail* program

```
tail [options] [file]...
```

This program prints the end (10 lines by default) of each file to standard output. For example:

```
user1@base:~$ tail -n 2 /etc/passwd
libvirt-qemu:x:64055:106:Libvirt Qemu,,,:/var/lib/libvirt:/usr/sbin/nologin
uuidd:x:116:124::/run/uuidd:/usr/sbin/nologin
user1@base:~$
```

shows the last two lines of the file '/etc/passwd'. If the number of lines has the form '+N', '`tail`' prints from line N to the end. For example:

```
user1@base:~$ cat f1
First name:Last name
Carlos:Pérez
Carolina:Sáez
Juan:Puerto
user1@base:~$ tail -n +2 f1
Carlos:Pérez
Carolina:Sáez
Juan:Puerto
user1@base:~$
```

since the number of lines is '+2', '`tail`' discards the first line (prints from the second one to the end). This option is useful for removing header lines, as in the example.

Exercise 3: using *tail*

Given the file 'f1' with the following content:

```
-----------------------------
First name:Last name
-----------------------------
Carlos:Pérez
Carolina:Sáez
Juan:Puerto
```

Write a command line that displays the first and last names without any of the header lines. The line should work correctly for all files that have three header lines, regardless of the number of lines they contain.

Although there are several ways to create a file with that content, an easy way to create the file is to use the 'cat' program:

```
user1@base:~$ cat > f1
```

Since the 'cat' program reads from standard input, it will be stuck waiting for us to type in information. It is easier to paste the file content in the terminal, instead of typing it. The end of the file is generated with 'Ctrl-d'.

In fact, it can be checked using the 'cat' program itself:

```
user1@base:~$ cat f1
-----------------------------
First name:Last name
-----------------------------
Carlos:Pérez
Carolina:Sáez
Juan:Puerto
user1@base:~$
```

Tasks to be completed

1. Prepare the file 'f1' with the appropriate content (six lines in total).
2. Type the command line that displays the requested information.
3. Check that it works correctly.

To be uploaded

- Copy of the command line execution.

The *cut* program

```
cut [options] <file_list>
```

This program displays part of each line of the files on the screen. For example, using as input a copy of the first five lines of the users file:

```
user1@base:~$ head -n 5 /etc/passwd > pw
user1@base:~$ cat pw
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
user1@base:~
```

The following command only shows the first ten characters of the previous file:

```
user1@base:~$ cut -c 1-10 pw
root:x:0:0
daemon:x:1
bin:x:2:2:
sys:x:3:3:
sync:x:4:6
user1@base:~$
```

The 'cut' program can also work with fields, for example:

```
user1@base:~$ cut -d ':' -f 3 pw
0
1
2
3
4
user1@base:~$
```

generates a list of the first five user identifiers (3rd field of the file '/etc/passwd'). The '-d' option specifies the delimiter character. Quoting the colon is to prevent the shell from giving it a special meaning (with a colon it is not important, but it would be, for example, with '|').

Exercise 4: using *cut*

Given the file 'f1' with the following content:

```
Title/Year/Director/Cast
Casablanca/1942/Michael Curtiz/Humphrey Bogart, Ingrid Bergman
Citizen Kane/1941/Orson Welles/Orson Welles, Joseph Cotten
From Here to Eternity/1953/Fred Zinnemann/Burt Lancaster, Montgomery Clift
Vertigo/1958/Alfred Hitchcock/James Stewart, Kim Novak
```

List the title and director of the films in the file.

Tasks to be completed

1. Prepare the file 'f1' with the appropriate content (in all five lines).
2. Type the command that displays the requested information.
3. Check that it works correctly.

To be uploaded

- Copy of command execution.

The *grep* program

```
grep [options] <pattern> <file_list>
```

This command displays the lines of files that match a pattern. For example:

```
user1@base:~$ cat f1
Title/Year/Director/Cast
Casablanca/1942/Michael Curtiz/Humphrey Bogart, Ingrid Bergman
Citizen Kane/1941/Orson Welles/Orson Welles, Joseph Cotten
From Here to Eternity/1953/Fred Zinnemann/Burt Lancaster, Montgomery Clift
Vertigo/1958/Alfred Hitchcock/James Stewart, Kim Novak
user1@base:~$ grep 'Orson Welles' f1
Citizen Kane/1941/Orson Welles/Orson Welles, Joseph Cotten
user1@base:~$
```

Although the easiest way to use '`grep`' is with a string, '`grep`' supports more complex filtering using regular expressions. For example:

```
user1@base:~$ grep -E '^[FV]' f1
From Here to Eternity/1953/Fred Zinnemann/Burt Lancaster, Montgomery Clift
Vertigo/1958/Alfred Hitchcock/James Stewart, Kim Novak
user1@base:~$
```

In this command, '`grep`' selects the lines that start with 'F' or 'V'.

The *sort* program

```
sort [options] <file_list>
```

This command is used to sort the lines of the text files and print the result to standard output. For example:

```
user1@base:~$ cat f1
one
two
three
four
five
user1@base:~$ sort f1
five
four
one
three
two
user1@base:~$
```

'`sort`' offers several interesting options, such as sorting numerically (30 goes before 100) or sorting based on a certain field. For example:

```
user1@base:~$ cat f1
First name,Last name,Age
Carlos,Pérez Conde,18
Carolina,Sáez Iniesta,52
Pedro,López Sáez,52
Juan,Puerto Sáez,8
user1@base:~$ sort -t ',' -k 3n -k 2 f1
First name,Last name,Age
Juan,Puerto Sáez,8
Carlos,Pérez Conde,18
Pedro,López Sáez,52
Carolina,Sáez Iniesta,52
user1@base:~$
```

sorts the content of file 'f1' by age (numerically) and undoes the ties using surnames.

Exercise 5: sorting with *sort*

You want to sort the content of the file '/etc/passwd' by user identifier (it is the third field), considering that the identifier is a number.

Tasks to be completed

1. Type the requested command.
2. Run it and check that it works as expected.

To be uploaded

- Command used.
- Results obtained.

The *uniq* program

```
uniq [options] [src_file [dst_file]]
```

This command discards from the input successive identical lines (leaving only one) and writes the result to the output. For example:

```
user1@base:~$ cat f1-ordered
one
three
three
three
two
two

user1@base:~$ uniq f1-ordered
one
three
two
user1@base:~$
```

Note that '`uniq`' only discards consecutive identical lines and hence the need to work on an ordered input file. If applied to a not ordered file, the result is different:

```
user1@base:~$ cat f1
one
two
three
two
three
three
user1@base:~$ uniq f1
one
two
three
two
three
user1@base:~$
```

Exercise 6: using *cut*, *sort* and *uniq*

Extract the list of the different shells used by system users, erasing duplicates.

As an aid to solve the problem, it is known that:

- The requested information is in the file '/etc/passwd'.
- In each line of the file, the last field (the seventh) is the shell of the corresponding user.

Tasks to be completed

1. Extract the list of shells from the user file and save it in the file 'shells'.
2. Create an ordered version of the above file (for example, 'ordered-shells'.
3. Eliminate duplicates to generate the final list.

To be uploaded

- Command lines used.
- List of shells without duplicates.

The *find* program

```
find [path(s)] [search condition(s)] [actions]
```

This command enables searching for files through the directory trees specified by the paths, evaluating the search conditions from left to right. Furthermore, actions to be performed can be specified on the results obtained.

For example, if we want to search for files that start with 'pass' in the '/etc' directory we would write:

```
user1@base:~$ find /etc -name "pass*" 2>/dev/null
```

If we want to find files that are larger than 10 Mbyte in '/usr/bin' and do an '`ls -lh`' of each one, we would write:

```
user1@base:~$ find /usr/bin -size +10M -exec ls -lh {} \;
```

where '{}' is the name of each file that matches the required conditions and ';' indicates the end of the command (it is the ';' character, but it must be preceded by '', since ';' has a special meaning for the shell).

The '`find`' command is very functional. Using logical operations, several conditions can be combined that allow for more complex and exact searches. For more information, it is possible to check the command manual page.

Exercise 7: using *find* to find objects

Use '`find`' to find in the '/usr/bin' directory those files that, at the same time, begin with the letter 'b', are less than 10 KB in size, and are files (and not other types of objects, such as directories or symbolic links). For each one of them, a long list with an easily readable size (options '`-lh`') must be printed.

To solve the exercise, it is recommended to check the 'type' and 'size' options of '`find`' in the manual.

Tasks to be completed

1. Type the requested command.
2. Run it and check that it works as expected.

To be uploaded

- Command used.
- Results obtained.

Commands linked by pipes

A pipe is a sequence of one or more commands separated by the '|' character. In this case, the standard output of one command is connected to the standard input of the next one. The shell waits for all commands in the pipeline to finish before requesting the next command line. (The exit status of a pipe corresponds to the exit status of the last command.)

An example:

```
user1@base:~$ ps -A | less
```

'`ps -A`' generates a list with all the processes of the system. This listing, instead of going to the terminal, goes to the pipeline, from where it is read by the 'less' program, which sequentially displays it in the terminal page by page. Finally, when 'less' finishes its execution, the shell displays the prompt again.

A set of commands are used with pipes (as filters) to accomplish quite complex tasks in a simple way, as can be seen in the following examples.

Example: applying a filter to a set of files

```
user1@base:~$ cat f1 f2 | wc -l
22
user1@base:~$
```

In this example, the 'cat' program concatenates the files 'f1' and 'f2', allowing 'wc' to work on them as if they were a single file.

Example: applying several successive filters

```
user1@base:~$ ls -l /usr/bin | cut -d ' ' -f 3 | sort | uniq

daemon
root
user1@base:~$
```

In this example, a list without duplicates of the users who own files in the directory '/usr/bin' is obtained. As expected, only system users (*root*, the administrator, and *daemon*, another system user) have files in this folder.

However, it is possible to see an additional blank line in the output. Why does this happen? How can you remove it? To shed light on this unexpected result, apply filters one by one. However, as the result of 'ls' would be very long, it is a good idea to limit it by using 'head':

```
user1@base:~$ ls -l /usr/bin | head -n 4
total 303548
-rwxr-xr-x 1 root   root         60064 Feb 28  2019 [
-rwxr-xr-x 1 root   root            96 Oct 11  2019 2to3-2.7
-rwxr-xr-x 1 root   root         10384 Jan 30  2016 411toppm
user1@base:~$
```

As it is possible to see, the first line shows a summary and, thus, it should be deleted. How to do this is left as an additional exercise.

Example: detecting users with duplicate identifier in '/etc/passwd'

```
user1@base:~$ cat /etc/passwd | cut -d ':' -f 3 | sort | uniq --count --repeated
user1@base:~$
```

This pipeline works as follows:

- 'cat' feeds the pipe with '/etc/passwd'
- 'cut' extracts the numeric identifier (the delimiter is ':' and it is the 3rd field)
- 'sort' sorts the list of identifiers
- 'uniq' shows the repeated elements together with the number of occurrences ('--count')

In this case, as you would expect in a well-configured system, there are no repeating identifiers. However, what would happen if a hacker added an alternative name for the system administrator (identifier 0)?

```
user1@base:~$ cp /etc/passwd passwd-probe
user1@base:~$ echo 'hacker:x:0:0:I am the hacker:/root:/bin/bash' >> passwd-probe
user1@base:~$ cat passwd-probe | cut -d ':' -f 3 | sort | uniq --count --repeated
      2 0
user1@base:~$
```

As can be seen from the test file, duplicates are detected. To see which entries have user ID '0', use the *grep* program.

```
user1@base:~$ grep "^.*:.*:0:" passwd-probe
root:x:0:0:root:/root:/bin/bash
hacker:x:0:0:soy el hacker:/root:/bin/bash
user1@base:~$
```

Exercise 8: finding big files

Using pipes generates a list sorted by size (from largest to smallest) of the three largest files in the '/usr/bin' directory. The output should be something like:

```
-rwxr-xr-x 1 root   root        19M Feb  2  2020 mysql_embedded
-rwxr-xr-x 1 root   root        13M Apr 21 14:26 qemu-system-x86_64
-rwxr-xr-x 1 root   root        13M Apr 21 14:26 qemu-system-i386
```

Suggestions:

- Use '`ls`' to generate the list ordered by size (check the manual for the option '--sort').
- Do not forget to eliminate the first line of the list.

Tasks to be completed

1. Type the requested command line.
2. Check that it works as expected.

To be uploaded

- Copy of command line execution.

Exercise 9: removing headers and footers

Given the file 'f1' with the following content:

```
=========================================================
Registered list
Course 2020-21
Last updated: September 1, 2020
=========================================================
--------------------------------
First name:Last name
--------------------------------
Carlos:Pérez
Elena:Rubio
Carolina:Sáez
Rafael:Martínez
Alberto:Marqués
Sonia:Puerto
--------------------------------
Total=4
--------------------------------
```

Write a command line that shows the first and last names without headers or final summary. The line should work correctly with all files that have eight header lines and three footer lines, regardless of the number of registered students.

Tasks to be completed

1. Type the requested command line.
2. Check that it works as expected.
3. Add one or two more registered students to the 'f1' file and check that it still works correctly.

To be uploaded

- Command line used.

Exercise 10: count the shells used

It is known that the users' shell is the last field in the '/etc/passwd' file. We want to find out how many different shells are used in the system.

Tasks to be completed

1. Type a command line (using pipes) that provides the requested count.
2. Execute it and verify that it works as expected.

To be uploaded

- Copy of command line execution.

Generated on: 2021-05-28 10:47 UTC. Generated by Docutils from reStructuredText source.

DSI - Laboratory Session 2: Scripts

27 May 2021

Contents

Goals

The main goal in this second session is that students become familiar with the use of files with a sequence of commands that the command interpreter must execute. Additionally, to construct more complex scripts, some control structures and the use of variables are going to be introduced. All the experiments are going to be performed in the base machine of NETinVM.

Previous work

Commands lists

A command list is a sequence of one or more pipes separated by one of the operators ';', '&', '&&' or '||', and optionally ended by ';', '&' or 'new line'. Operators '&&' and '||' have equal precedence, followed by ';' and '&', which have the same precedence between them.

Commands separated by ';' are executed sequentially. The result is the same as if the commands were written one after the other, each on one line. The difference is that, in this way, all the commands to be executed can be written on a single line. The command interpreter waits for each command to be finished before executing the next one. For example:

```
$ date; sleep 5; date
```

In this example, the command 'date' is executed first, and shows the current date and time to the standard output. Once the execution of this command has finished, the next command is executed ('sleep 5'). This command pauses for five seconds. Once this pause has finished, the last command is executed, which again shows the current date and time, and so revealing that those five seconds have actually passed.

Commands that end with the control operator '&' are executed in the background (the shell does not wait for the command to finish before executing the next one). For example:

```
$ kcalc & kwrite &
```

The control operators '&&' and '||' stand for AND and OR lists respectively. An AND list has the form: 'command1 && command2'. In this case, command2 is executed if and only if command1 returns an exit status of 0 (that is, if it succeeds). For example:

```
$ cd test && echo "I have been able to change to the directory"
bash: cd: test: File or directory does not exist
$ mkdir test
$ cd test && echo "I have been able to change to the directory"
I have been able to change to the directory
$
```

If the directory change is successful, the message is printed. Otherwise, when the 'cd' command fails, the 'echo' command cannot be executed and the message does not appear (although the error message of the 'cd' command does appear).

Similarly, an OR list has the form: 'command1 || command2'. In this case, the command2 is executed if and only if the command1 returns an exit status distinct from 0 (that is, if it is unsuccessful). For example:

```
$ cd temp || echo "The directory does not exist"
```

If the directory 'temp' does not exist, the command 'cd' cannot be executed. In this case, the message will appear as the directory does not exist. (Note that 'cd' will also print its own error message).

Finally, several commands can be grouped between braces, so that several commands are syntactically treated as one. For example:

```
$ cd temp || { echo "The directory does not exist, I am going to create it ...";
mkdir temp && echo "Directory created"; }
```

Previous exercise 1: conditional command list

It must be written a conditional command list which:

- creates the directory '/tmp/backup-copy'.
- if it is successful, the file '/etc/passwd' must be copied to that directory and then the contents of the directory must be listed.

Hint

To check that it works, just repeat the command twice, as the second time it will fail. Why?

Tasks to be completed

1. The requested command list must be built.
2. The command list must be executed. It must also be checked that **the file has been copied and the requested list is shown**.
3. The command list must be executed again to verify that the message is **not** shown.

To be uploaded

- Command used.
- Result after the different executions.

Command substitution

This allows a command to be substituted by its output. For example:

```
ps >processes_$(date +"%Y-%m-%d")
```

saves a list of processes in a file whose name will be 'processes_2019-02-21' (if it is executed on 21 February 2019), since the command interpreter, before doing the redirection, evaluates the commands that are inside '$()'. In this case, 'date +"%Y-%m-% d"'. Some examples follow below.

- Using 'echo' in a simple way:

  ```
  echo "There are $(ps -A | tail -n +2 | wc -l) processes on the system."
  ```

  This command shows the number of processes running on the system. ('ps' is used to generate the list of processes, 'tail' to remove the header, and 'wc' to count the number of lines).

- A more extensive example:

  ```
  echo -e "System: $(hostname) \n- processes: $(ps -A | tail -n +2 | wc -l) \n-
  Disk:\n$(df -h /)"
  ```

  This example is quite similar to the previous one, although with several lines of information. The '-e' option enables 'echo' to process escape sequences like '\n', which indicate a new line. In addition, it also displays information about the usage of the root file system employing 'df'.

Previous exercise 2: taking advantage of command substitution

A command line must be prepared that when executed adds the following information to the 'registry.txt' file:

- The name of the host, obtained with the 'hostname' command.
- The date and time, obtained with the 'date' command.
- The list of users connected to the system, generated by the 'who' command.

The format of the record should be as close as possible to the following lines:

```
==== START
System review: base
Date: Tue Sep 26 10:19:00 CEST 2017
Users:
user1 pts / 2 2017-09-26 10:08 (:0)
==== END
```

Tasks to be completed

1. The command line must be prepared.
2. It must be checked that it works correctly. (Hint: launch multiple terminals so that the user list changes).

To be uploaded

- Command line used.
- Copy of the file 'registry.txt'.

Evaluation of conditional expressions

It is useful to be able to evaluate conditional expressions, so as to establish if a number is greater than another, if there is a file, or if two text strings are equal. This evaluation is useful both for the flow control structures that are going to be reviewed below and for generating command lists using '&&' and '||'.

The recommended way to use these expressions is by using the following conditional construction:

```
[[ expression ]]
```

Although it is also possible to use these expressions with the commands 'test' and '[', it is necessary to enclose the characters that have a special meaning for the *shell*, such as '<' or '>'.

There are different types of expressions that can be evaluated. The most important follow.

File evaluation

Many file evaluation options can be found on the man page for the 'test' command. Some of them are:

-a file
    Returns true if file exists.
-f file
    Returns true if file exists and it is a normal file.
-d file
    Returns true if file exists and it is a directory.
arch1 -nt arch2
    Returns true if arch1 is newer than arch2.
arch1 -ot arch2
    Returns true if arch1 is older than arch2.

Example:

```
f="/etc"
[[ -a $f ]] && echo "'$f' exists"
[[ -f $f ]] || echo "'$f' is not a normal file"
[[ -d $f ]] && echo "'$f' is a directory"
```

String evaluation

-z string
: Returns true if the string length is zero.

-n string
: Returns true if the string length is not zero.

string
: Equivalent to the previous one.

cd1 == cd2
: Returns true if the strings are equal.

cd1 = cd2
: Equivalent to the previous one.

cd1! = cd2
: Returns true if the the strings are not equal.

cd1 < cd2
: Returns true if cd1 is alphabetically sorted before cd2.

cd1 > cd2
: Returns true if cd2 is alphabetically sorted before cd1.

Examples:

```
[[ "$(uname)" = "Linux" ]] && echo "This is Linux"
```

```
[[ $HOME ]] && echo "HOME has a length greater than 0"
```

```
[[ "abc" < "bcd" ]] && echo "The first string goes before the second"
```

Numerical evaluation

Syntax:

```
number1 op number2
```

Numeric values must be integers (positive or negative). The numerical operators that can be used are:

-lt
: Less than

-le
: Less than or equal to

-gt
: Greater than

-ge
: Greater than or equal to

-eq
: Equal to

-ne
: Not equal to

Example:

```
a=1; [[ $a -lt 10 ]] && echo "a (a=$a) is less than 10"
```

Combining conditional expressions with '[[...]]'

(expression)
: Returns the value of *expression*. It can be used to avoid the default precedence of the operators.

! expression
    True if *expression* is false.
expression1 && expression2
    True if both expressions are true.
expression1 || expression2
    True if any expression is true.

The *expression2* of '&&' and '||' is evaluated only if it is necessary.

Examples:

```
[[ ! -f /etc ]] && echo "/etc is not a normal file"
```

```
[[ -d /etc || 3 -gt 2 ]] && echo "It is accomplished"
```

```
[[ -f /etc || ("abc" <"bcd" && 3 -lt 10) ]] && echo "Also true"
```

Previous exercise 3: working with conditional expressions

When the different examples in the previous sections have been reviewed and fully understood, build a command that writes 'Everything is OK :-)' when all the following conditions are met:

- The content of variable 'n' is numerically less than or equal to 80.
- The object whose name is in the variable 'f' is a normal file.
- The name of this object is '/usr/bin/date'.

Tasks to be completed

1. Build the command line described in the exercise.
2. Different values for 'n' and 'f' must be defined and check that the command works as expected.
3. Several cases must be tested to ensure that the command works properly.

To be uploaded

- Copy of the command.
- Copy of the pairs (n, f) used for testing the command.

The read command

The 'read' command waits for a line of text to be entered through the standard input.

The easiest way of using it is to read a line and assign it to a variable:

```
echo -n "Enter a text: "; read x; echo "You have entered: $x"
```

It is also possible to assign multiple variables at the same time separated by spaces. For instance:

```
echo "one two" | { read a b; echo "a is '$a' and b is '$b'"; }
```

In the case that 'read' is used to read several variables from the keyboard, the separation is made by spaces. If *ENTER* is pressed before entering as many strings separated by spaces as there are variables, the last variables are left unassigned.

Previous exercise 4: using read

Construct a command that asks for a name and an age, writing the phrase: 'X is Y years old', where X is the name and Y is the age entered by the user.

Tasks to be completed

1. The requested command line must be constructed.
2. Check that the command works as expected.

To be uploaded

- Copy of the command.
- A proof that it is properly working.

Work in the laboratory

Scripts

The task that must be performed with the shell may require a large number of commands. In this case, it is a good idea to keep all these commands in a file and thus be able to execute them together by invoking the file name. These files are called 'scripts'.

The simplest way to run a script is to run the shell, passing the file name as a parameter: '`sh file`'. Moreover, to execute it directly, it is possible to write '`./file`'. Prior to that, the file must have the appropriate access permissions for its execution (this is a quick reminder to change the permissions: '`chmod +x file`'). The first line of the script should also be:

```
#! /bin/bash
```

assuming that the script is going to be executed by the '/bin/bash' interpreter. This mechanism is wider, applicable both to shell scripts and to any other shell (Python or Perl, for example). Replace '/bin/bash' with the name of the program with which the operating system must process the file.

Example:

```
#! /bin/bash
# Show the N largest files in /usr/bin
echo "Enter the number of files to display:"
read num
ls -lh --sort size /usr/bin | tail -n +2 | head -n $num
```

When executed, the script asks for a number and that number of files in '/usr/bin' appears on the screen, ordered by size from largest to smallest.

Note

Notice that command sequences in the scripts are generated by including commands on separate lines, as below:

```
echo -n "Enter a number:"
read n
```

However, both commands could also be put on the same line, as has been previously done:

```
echo -n "Enter a number:"; read n
```

In the *scripts* it is better to use the most readable option which usually is to separate the commands on different lines.

Exercise 1: a simple script

Most but not all Linux programs are in '/usr/bin'. A useful command in Linux is the command 'which <command>', which locates the place (absolute path) where 'command' is located. The proposed simple script must do something quite similar; namely, searching for user files in the home directory tree.

Tasks to be completed

Design a script, which will be called *search*, that will be used to find the location (absolute path) of a file in the directory tree starting from the *home* directory. The script will request the file name on the screen. An example of how it should work is below:

```
Enter the name of the file to search:
f1
The f1 file is in /home/user1/Documents/tmp/f1
```

It must be noted that the file must have execution permission ('chmod + x search'), and it must be invoked with its name from the current directory: './search'.

To be uploaded

- Copy of the script.

Arguments in scripts

The previous script is simple and invoked by its name, it requests information, and it executes a series of commands. However, in this case, it would be interesting to provide the name of the file to search as an argument. In general, a script can have arguments.

To do this, the command interpreter has a series of predefined variables (as has been previously shown, the reference of the variables is made with the symbol '$'):

$#
: Stands for the number of arguments passed to the script

$*
: Refers to all arguments except the name of the script

$?
: Stores the return code of the last command

$0
: Represents argument 0 (the name of the script)

$1 to $9

Represent the first 9 arguments

Example:

```
#! /bin/bash
# script that displays the variables $#, $0, $* and $?
echo "$0 was called with $# arguments"
echo "The arguments are: \"$*\""
echo -n "System date and time: "; date
echo "The code returned by the last command was $?"
```

Once the script is saved and the necessary permissions have been given, this could be one of the results (it is assumed that the file has been saved with the name 'show'):

```
$ ./show one two three four
./show has been called with 4 arguments
The arguments are: "one two three four"
System Date & Time: Tue Oct 11 08:52:57 CEST 2016
The code returned by the last command was 0
$
```

Arithmetic expansion

Although it is possible to use the 'expr' program to perform arithmetic calculations, the simplest way to substitute an arithmetic expression for its result is as follows:

```
$((arithmetic expression))
```

If the arithmetic expression is invalid, the shell prints an error message and no substitution occurs.

The arithmetic evaluation rules are as follows:

++, --	increment and decrement (can be post or pre)
!, ~	logical and binary negation, respectively
**	power
*, /, %	multiplication, division, remainder
+, -	addition and subtraction
<<, >>	bits shift to the left and to the right
<=, >=, <, >	comparisons
==, !=	equality and inequality comparisons
&, ^, |	Binary AND, XOR and OR
&&, ||	Logical AND and OR
expr?expr:expr	conditional evaluation
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=	assignments

Example:

```
#! /bin/bash
# Powers of 2
```

```
echo "Please enter a number"
read x
echo "2 raised to $x = $((2 ** x))"
```

Repetitive structure *while*

Syntax:

```
while list; do list; done
```

The 'while' command continually executes the 'list' 'do' as long as the last command in the 'list' 'while' returns a zero exit state. The exit status of the command 'while' 'is that of the last command executed of the 'do' 'list', or zero if no command has been executed.

Examples:

- Print `n` numbers in a decreasing way, until reaching zero, passing `n` as an argument:

  ```
  #! /bin/bash
  # Print n numbers

  n=$1
  while [[ $n -gt 0 ]]
  do
      echo "$n"
      n=$((n - 1))
  done
  ```

- Wait until the file '/tmp/end' exists, checking it every `n` seconds, and using `n` as an argument:

  ```
  #! /bin/bash
  # Wait until the file "/tmp/fin" exists

  while [[ ! -f "/tmp/end" ]]
  do
      echo -n "."
      sleep $1;
  done
  echo -e '\n It exists!'
  ```

Tip: to check that it finishes properly, another terminal can be opened and then create the file (ex: '`touch /tmp/end`').

Exercise 2: testing loops with *while*

A script must be written with the name `create` that creates `n` files, whose name will have a common part that will be used as a parameter - and the rest of the file name will simply be a number to distinguish them. In this way, executing the command:

```
create name 100
```

It will create 100 files (from 0 to 99), following the pattern: 'name0', 'name1', ..., 'name99'. The files will have a common part and a different part. Example of file 50:

```
This file is called: name50
Line_1 ****************************
```

Tasks to be completed

1. The script described in the exercise must be built.
2. It must be checked with different numbers of files (10, 100, 1000) to verify that the command works as expected.

To be uploaded

- Copy of the script
- Functioning test '`create myfile 13`' and for '`create yourfile 100`'.

The *read* command with *while*

The most common way of using the *read* command is to read a variable by the keyboard. Moreover, the *read* command can also be used as a filter to read multiple data from a pipe and then display the chosen information. An example is shown with the script:

```
#! /bin/bash
# Example of using read with while

ls -l | while read perm links usr grp size month day time name
do
    [[ -f $name ]] && echo "The file $name: has a size of $size bytes, and the 2nd
line is: $(cat $name | head -n 2 | tail -n 1)"
done
```

The script lists the contents of the current directory with '`ls -l`'. The 'read' command assigns sequentially the different fields to the variables. In this case, the first field to 'perm', the second to 'links', etc. On the assumption that there were more fields than variables, the last variable would collect the final fields that could not be assigned to separate variables. If there were fewer fields than variables, the final variables would remain unassigned. The rest of the script explains itself. In the case of a normal file, its name and size are shown in the terminal and the second line is also displayed.

Exercise 3: testing *read* with *while*

Following the model of the previous script, build a script that shows the number of links of each subdirectory contained in any directory. To do this, the directory name will be used as a parameter. In the script, the number of links must be the second field in the instruction '`ls -l`'. In the case of a directory, this number is at least two, since it contains the current directory ('.') and a link to the upper directory ('..'). Therefore, 2 would have to be subtracted to show the number of 'real' or different subdirectories from the two described. Additionally, the script must be invoked from our working directory, and indicate the directory to analyse. For example, if the script had the name 'show-dirs', executing '`show-dirs /etc`' will display something like the following:

```
aliases.d: 0
alternatives: 0
ant.d: 0
apparmor: 0
apparmor.d: 5
audisp: 1
audit: 1
auto.master.d: 0
avahi: 1
bash_completion.d: 0
binfmt.d: 0
ca-certificates: 1
chrony.d: 0
cifs-utils: 0
ConsoleKit: 1
...
```

It must be noted that to test the existence of the directory, the absolute address has must used, since this is done on any directory. This absolute address is constructed from the script argument and the 'name' variable.

Tasks to be completed

1. The script described in the exercise must be built.
2. The script must be checked with different directories to verify that it works properly.

To be uploaded

- Copy of the script
- Functioning test for '`show-dirs /usr`'.

The *shift* command to shift the arguments of a script

In a script, 'shift' is used typically to handle any number of arguments. The command 'shift' drops argument 1 and shifts the other arguments so that 2 becomes 1, and so on and so forth:

```
#!/bin/bash
# Script that displays the variables $#, $0, $* and $?
echo "$0 was called with $# arguments"
echo "The arguments are as follows:"
i=1
while [[ $# -gt 0 ]]
do
    echo "- argument $((i ++)): $1"
    shift
done
```

Repetitive structure *for*

```
#! /bin/bash
for name [ in words ]
do
    list
done
```

The list of 'words' after 'in' is expanded to generate a list of items. The variable 'name' is defined as each of the elements in each iteration, and then 'list' is executed each time. The return status is the exit status of the last command that was executed. If the element expansion after 'in' is empty, no command is executed and the exit status is zero.

Example:

```
#! /bin/bash
# Create tmp folder in each subdirectory

for x in *
do
    (cd $x && mkdir temp) 2>> /dev/null
done
```

This script creates a folder with the name 'temp' in each of the subdirectories of the current working directory. The redirection of the standard error output to '/dev/null' is used to prevent the error messages from appearing on the terminal. Those messages are created when trying to change to a name that is not a directory.

It is possible to obtain information through the shell man page ('man bash') about other flow control structures. These are the 'until', 'case', and 'select' structures. Information about the 'break', 'continue', and 'exit' commands can also be found on that man page, which enable stopping the execution of repetitive structures, in a similar way to C/C++.

Exercise 4: testing loops with *for*

The policy for naming files that has been followed in Exercise 2 is not very smart. For example, when the files are listed in the directory that contains them, 'name10' comes before 'name2'. This would not happen if the names were 'name02' and 'name10'. Following this idea, make a new script that fulfils the task proposed in Exercise 2, but with the new naming policy. In this way, create a script called 'create2 <name> <n>' that creates 'n' files with the same content as in Exercise 2, this time with the reference to the file name, included in the first line, changed to the new policy. In this way, after executing 'create2 name 240' the following files will be obtained:

```
name000
name001
...
name238
name239
```

In this case, the elements of the list must be numbers of equal size (by adding leading zeros if necessary). One way to do this is with the command '`seq [options] <first> <last>`'. This command creates a list of numbers from first to last. The command is quite flexible (options can be seen in the 'seq' man page). For example, it is possible to set only the last number, the increment, etc. There are also several options, including the '-w' option, which generates numbers with the same width. That is, adding leading zeros if necessary. Finally, it must be ensured that the 'n' files are numbered from 0 to 'n-'.

Tasks to be completed

1. The script described in the exercise must be constructed.
2. The script must be tested with different names and different numbers of files. It should be verified that both the name of the file and its content follow the new naming policy.

To be uploaded

- Copy of the script
- Functioning test for an example with 100 files, with a screenshot of the directory that contains the files. Copy of the first and last file.

Conditional structure *if-then-else*

Syntax:

```
#!/bin/bash
if list
then
    list
[ elif
    list
 then
    list ]
...
[ else
    list ]
fi
```

The 'list' of the 'if' is executed. If its exit status is zero, then the 'list' of the 'then' is executed. Otherwise, each 'elif' list is executed in turn (executing its corresponding 'list' 'then' in the case of obtaining an output result equal to zero). If its exit state is non-zero, the 'else' list is executed. The exit status is that of the last command executed or zero if no condition was true.

Example:

```
#! /bin/bash
if who | grep -s paco> /dev/null
then
    echo "paco is connected"
else
    echo "paco is not connected"
fi
```

In this example, a check is made as to whether the user paco is connected or not. The '/dev/null' redirection of the 'grep' command output is made so that this information is not displayed on the screen.

Exercise 5: combining 'for' loops with 'if-then-else'

Unfortunately, despite our warnings, a user has generated 1000 files using the script 'create'. In this way, the first file will be 'name0' and the last one will be 'name999'. These files have the structure described in 'Exercise 2: testing loops with while'.

This file naming must be corrected so that they follow the naming policy of the exercise 'Exercise 4: testing loops with for'. It should be considered that the files may have been edited by adding information on any line except the first one (which contains the name). The added information must be kept, so the file must be renamed following the new naming policy (but keeping the content).

Design a script, which will be invoked as '`correct <name> <number>`', and which will correct both the file name and the reference to this name in the first line, if necessary, but keeping the rest of the information in the file.

Tasks to be completed

1. 1000 files must be generated with the wrong naming policy: 'name0', 'name1', ..., 'name999' using the 'create' script.
2. Some of the files must be edited by adding information from the second line, for example, the names of the students who are doing the lab session. Edit, for instance, a pair of files whose names are going to change. For example, files 2 and 20.
3. The script '`correct <name> <number>`' described in the exercise must be built. It is a good idea to start from the architecture of the 'for' loop in the 'create2' script to create the corrected names. Inside the loop, it is possible to add a variable that is increased with each iteration to generate the original name of the files.
4. Finally, the script must be tested. Now all the files will follow the correct naming policy. To do so, some files will have changed their name format, also changing the name in the first line if necessary. The two edited files (2 and 20) will become 002 and 020 but keeping the edited information.

To be uploaded

- Copy of script.
- Functional test with a screenshot of the directory containing the original files, and a screenshot of the modified content.
- Copy of the files 'nombre020', 'nombre200'.

DSI - Laboratory Session 3: Execution of Python Applications, Management of Processes and Threads

28 May 2021

Contents

Goals

The main goal of this session is to provide a global understanding of how processes and threads are managed in Unix. In addition, this session deals with the parallelisation of Python applications using multi-threading and multi-processing techniques. The contents of this session are focused on:

- Commands for managing the execution of processes in Unix: ps, top, kill, killall.
- Programming of Python applications that use multiple threads and processes, understanding the different way in which Linux manages their execution.

Previous work

Before attending the lab session, students should:

- Read the lab session document, paying special attention to the explanations of each section.
- Complete the exercises in the previous work section, which help to understand how to use the main commands for process management.
- Analyse and understand the examples. It is advisable to use these commands in the virtual machine and make only small changes until you are sure you understand their execution.

Work to be uploaded

- Results obtained in the resolution of the previous exercises.

Introduction

As it has already been introduced in the course, a process is a program that is executed by a processor. While a program is defined by the code that results from compiling and linking the program's source code, a process is an active structure that includes the program's code, its data, and the state of the resources the program needs to run.

The `ps` command

The operating system allows you to see the state of the processes at a given time by means of the '`ps`' command. The following may be an example of the list presented after the execution of the '`ps`' command (without arguments) at a certain time:

```
  PID TTY          TIME CMD
 4304 pts/3    00:00:00 sh
 4313 pts/3    00:00:00 ps
```

where PID is the process identifier, TTY is the shell associated with that process, TIME is the time in which the process is running and CMD is the command corresponding to the process. By means of the '`man`' command we can consult the different options to invoke the '`ps`' command.

The Linux command '`ps`' enables using different types of options (Unix, BSD, and GNU). It is advisable to learn the standard Unix options. Optionally, it may be useful to use some of the GNU options that are not included in Unix (such as '`--sort`', which enables the generation of sorted lists).

Some particularly interesting options are:

`ps -A, -e`
: selects all the processes (by default '`ps`' only shows the processes associated with the current shell)

`ps -C cmdlist`

selects processes whose executable file is 'cmdlist'; for example:

```
ps -C bash
```

shows the processes that are running the program 'bash', and

```
ps -C bash,mingetty
```

shows the processes that are running 'bash' or 'mingetty'

```
ps -u userlist
```

shows the processes of the indicated users (a comma-separated list of users)

```
ps -g grouplist
```

selects processes by group name

```
ps -f
```

shows additional information (*full-format*)

```
ps -o format
```

'format' must be a list of fields separated by commas indicating which columns should be shown; for example:

```
ps -u carlos -opid,command
```

shows all the processes for the user 'carlos' ; for each process, only the columns 'PID' and 'COMMAND' are shown (name of the executable file and the arguments)

```
ps -H
```

provides a hierarchical output (child processes are indented after their parent)

Previous exercise 1: understanding the `ps` command

This exercise should be performed in a shell of the 'base' machine in the NETinVM environment. In this exercise, the following types of lists of processes must be obtained using the `ps` command:

a. All the processes of the 'root' user, with full-format (*full-format*).
b. Processes of the user 'user1' in a hierarchical form. The following fields must appear in the list:
    - the PID of each process
    - the PID of its parent process
    - the name of the executable file **without the arguments**

As a suggestion, the section of the `ps` manual page that explains which specifiers can be used with '-o' is 'STANDARD FORMAT SPECIFIERS'.

Tasks to be completed

1. Determine the command that generates the list (a).
2. Check that it works properly
3. Find out how many processes the 'root' user is running (Hint: use `wc`, `tail` and pipes).
4. Determine the command that generates the list (b).
5. Check with at least two processes that the PID of its parent actually

correspond to the parent process according to the hierarchical structure shown by the `ps` command.

To be uploaded

- Commands used to solve questions (a) and (b).
- Command used for counting the 'root' processes and total number of processes.
- Justification of question (5).

The `top` command

The `top` command enables dynamically visualising the state of the system and the processes, the CPU time consumed by each one, the occupied memory, the priority of each process, and so on.

When used interactively, it has a built-in helper that is activated by pressing the letter 'h' (for *help*). Through this help it is easy to find out how to modify the default setting of `top` to best suit the user requirements. Some possibilities are highlighted below:

- Enable/disable (*toggle*) different types of information: summary of process and processor information ('t'), summary of the use of different types of memory ('m'), detailed information about each processor ('1') (example '1' for the first processor), show inactive processes (*idle*, 'i'), etc.
- Select the columns that are shown and the column by which they are sorted ('f' or 'F').
- Filter by user ('u').
- Modify the current state of a process by sending it a signal (*kill*, 'k') or by changing the *nice* parameter (*renice*, 'r').

Previous exercise 2: adjusting the `top` information as needed

Configure the `top` command to show the required information in an easy-to-read way. Specifically, it is required to show:

- Detailed information about the use of each processor in the system.
- The following fields in the order shown below:
    - Process identifier
    - Process parent identifier
    - User
    - Percentage of CPU usage
    - Percentage of memory usage
    - Order (program) corresponding to the process, including the arguments
- The information is sorted by memory usage (from more to less).
- Show only the first ten processes.

Determine the 'top' command required to show a list of your own processes sorted by % of memory used.

An example of the output that meets these conditions would be:

```
 top - 18:10:23 up 1 day,  6:53,  3 users,  load average: 0.04, 0.11, 0.30
 Tasks: 166 total,   1 running, 165 sleeping,   0 stopped,   0 zombie
 %Cpu0  :  3.7 us,  1.3 sy,  0.0 ni, 95.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
st
 %Cpu1  :  6.1 us,  1.4 sy,  0.0 ni, 92.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
st
 %Cpu2  :  4.0 us,  3.3 sy,  0.0 ni, 92.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
st
 %Cpu3  :  3.7 us,  2.0 sy,  0.0 ni, 94.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
st
 KiB Mem :  4032136 total,   795828 free,  1547672 used,  1688636 buff/cache
 KiB Swap: 10485756 total, 10485756 free,        0 used.  2170088 avail Mem

  PID   PPID USER      %CPU %MEM COMMAND
12221      1 user1      0.7 12.5 qemu-system-x86
 2849   2633 user1      0.3  7.7 firefox-esr
 2694   2691 user1     20.9  4.2 kwin_x11
 2828   2633 user1      1.3  2.2 konsole
 2642   2633 user1      0.3  1.4 kded5
 2700   2691 user1      0.3  0.8 xembedsniproxy
 2638   2633 user1      0.0  0.8 klauncher
 2657      1 user1      0.0  0.8 kaccess
 2633      1 user1      0.0  0.5 kdeinit5
 1867      1 user1      0.0  0.1 systemd
13968   2828 user1      0.0  0.1 bash
14665  13968 user1      0.0  0.1 top
 2570      1 user1      0.0  0.1 dbus-launch
 2571      1 user1      0.0  0.1 dbus-daemon
 2599   1870 user1      0.0  0.0 ssh-agent
 2632      1 user1      0.0  0.0 start_kdeinit
 1868   1867 user1      0.0  0.0 (sd-pam)
 1870   1861 user1      0.0  0.0 startkde
```

Tasks to be completed

1. Execute the `top` command in a shell.
2. Carry out the required configuration.
3. Check that the solution meets the requirements.
4. Screenshot showing the result obtained after the execution of the `top` command. (In KDE, the screenshot can be done with the program 'spectacle').

To be uploaded

- Explanation of the chosen way to configure the `top` command.
- Screenshot with the output of the `top` command configured properly.

The `kill` command

The `kill` command is actually a user interface for the use of the `kill()` function. This function enables sending a signal or notification to another process, if the one generating the signal owns the process or is the superuser. With the -l option, the `kill` command shows a list of all the signals. A more detailed description can be obtained by calling the `man 7 signal` command. If the signal is not specified, the kill command sends the SIGTERM signal (orderly completion of a process, signal 15) by default. Signals can be identified by name or by number (see the complete list of signals), although it is more advisable to use the name, as the numbers are not standard and can change from one system to another.

When killing a process, it is better to send it the SIGTERM signal first, because the process can deal with this signal and can perform an orderly termination. The SIGKILL signal should only be used as a last resort, when the process does not respond to the SIGTERM signal and it must be terminated.

Therefore, to send the SIGTERM signal to process 1234 the following command must be used:

```
kill -s SIGTERM 1234
```

However, since `kill` sends SIGTERM by default, the following command could also be used:

```
kill 1234
```

Note

Although there is a 'kill' program, shells usually include an internal 'kill' command, which is usually executed by default. This is the case with the 'bash' shell, which is the one commonly used in 'Linux'. To review the options for this command, see the 'SHELL BUILTIN COMMANDS' section of the 'bash' manual page.

Previous exercise 3: killing processes with `kill`

The 'kwrite' application is not responding and it must be terminated. The `kill` command must be used to send it the proper signal.

Tasks to be completed

1. Execute the 'kwrite' program.
2. Find out the PID of this new process.
3. Send it the right signal to terminate the program.

To be uploaded

- Explanation of the command use to obtain the PID of the process.
- Command used to terminate the process.

The `killall` command

The `killall` command enables sending a notification to all the processes that are running a program. This command allows terminating all processes generated by a wrong program simply by specifying the command used to execute it. The options, available on the corresponding manual page (`man killall`), are similar to `kill`. For example, to kill all the processes generated by the `cat` program, it is required to use the following command:

```
killall -TERM cat
```

Since `killall` can kill many processes, a very useful option, especially when acting as administrator ('root' user), is '-u user1', which indicates that signals should only be sent to processes of the 'user1' user.

Previous exercise 4: killing processes with `killall`

Many copies of the 'dolphin' application have been launched over time. To avoid shutting down all the instances of the program one by one, it is required to use the `killlall` command.

Tasks to be completed

1. Launch 25 instances of the 'dolphin' application. (Hint: use 'for' and 'seq', from the Previous lab-session *command scripts*, considering that 'dolphin' must be launched in the second plane).
2. Execute the following command and make sure it gives 25:

   ```
   ps -C dolphin | tail -n +2 | wc -l
   ```

   Note: it is possible that, in a virtual machine, 25 copies of dolphin launched at the same time take a few seconds to start up ;-)
3. Use `kilall` to send the appropriate signal to terminate all instances of the program.

To be uploaded

- Explanation of the command used to launch the 25 instances of the 'dolphin' program.
- Command used to terminate all the processes.

The `pgrep` and `pkill` commands

Although the commands seen so far (`ps`, `kill`, and `killall`) are usually enough to manage the execution of processes, it is useful to know that there are two commands with a similar syntax and with complementary purposes. These commands are `pgrep` and `pkill`. In this case, instead of describing them in this document, an introductory video is supplied. However, to carry out the exercise it will be necessary to consult the manual (command `man`).

Introductory Video

Video 'pgrep-pkill.mp4' shows how the `pgrep` and `pkill` commands are used, and why they are complementary.

Previous exercise 5: killing processes with `pkill`

Many copies of the 'dolphin' application have been launched over time. To avoid shutting down some of the instances of the program one by one, it is required to use the `pkill` command to terminate some of them.

Tasks to be completed

1. Execute the following command to launch several copies of the 'dolphin' application:

```
for x in /{usr,etc,bin,boot} /home/user1/{,*}; do dolphin
$x & done

Note: it is possible that, in a virtual machine, all these
'dolphin' instances
launched at the same time take a few seconds to start up
;-)
```

2. Use `pgrep` to list all the 'dolphin' instances running.
3. Use `pgrep` to list all the 'dolphin' instances that have objects starting with '/home/user1/D'.
4. Change `pgrep` to `pkill` to terminate only the instances that have objects starting with '/home/user1/D'.
5. Check that, indeed, those instances have been terminated.

To be uploaded

- Commands used and a copy of the resulting outputs obtained.

The `time` command

The `time` command is used to determine how long it takes to execute a command. It is useful for testing the performance of scripts and commands executed on Unix. For example, if there are two different scripts doing the same work and the goal is to establish which one completes the action most quickly, it is possible to use this command to determine the duration of the execution of each script.

The way to use the `time` command is to link it with the command that is going to be measured. For example, it is possible to measure the time it takes the `find` command to find all files larger than 10MB in the folders inside `/usr`, as seen in the lab session 1:

```
time find /usr -size +10M -exec ls -lh {} \;
```

Results of the `time` command are specified by three different time measurements:

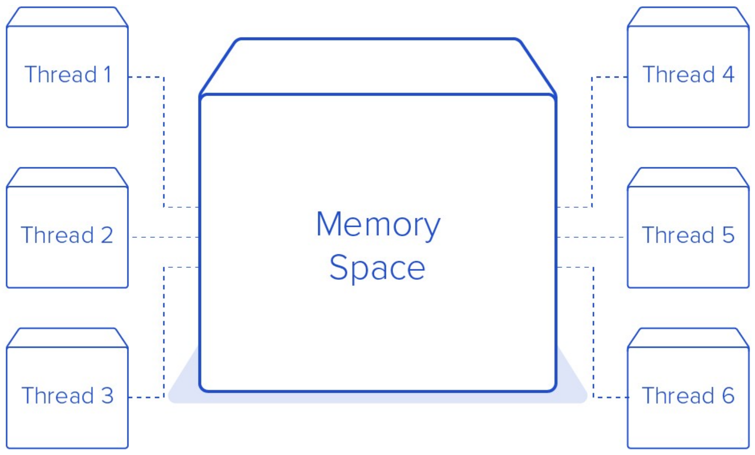
- Real: the time elapsed from the beginning to the end of the command execution. This is the time since the Enter key is pressed until the command is completed.
- User: the amount of time the CPU spends in user mode during the execution of the command. This time includes the time spent by a CPU core, or the sum of the times of several cores, if

the program is using several cores at the same time.
- Sys: the amount of time the CPU spends in kernel mode during the command execution.

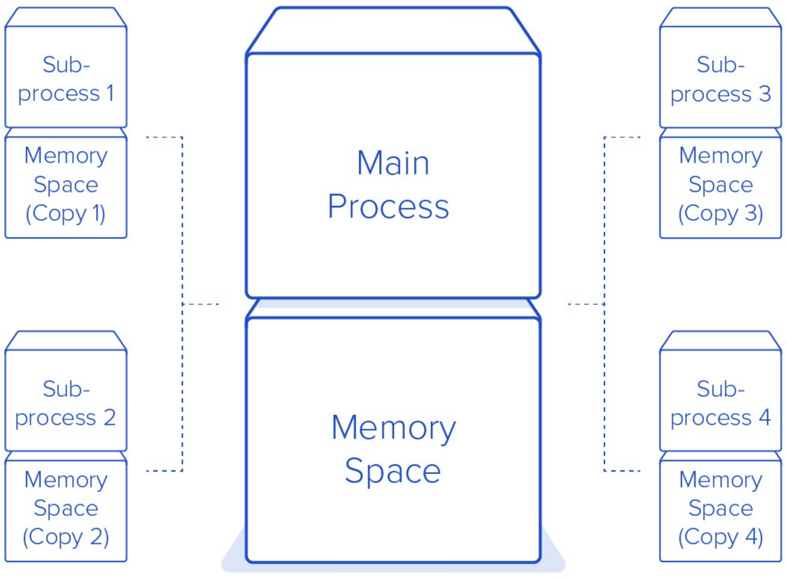
Work in the laboratory

Threads versus processes in Python

Multi-threading programming is one of most well-known techniques that can be used to achieve concurrency in Python. Multi-threading is a feature provided by the operating system. Threads are lighter than processes and share the same memory space. However, due to the limitations of the Python interpreter, threads generated in a program are executed concurrently but not in parallel, that is, the execution will change from one thread to another, whenever one of them is ready to do some work, but at a given moment only one thread is running. In this approach, the Python interpreter uses only one core of the CPU and it shares it between all the threads implementing a time division multiplexing.



Multi-processing programming in Python enables a program to spawn new processes that do not share memory. In this case, the execution can be truly parallel, with several processes running at the same time in different cores of the CPU. Thus, when new processes are generated each will have its own memory range assigned, in which a copy of the memory content of the original process is made.



This lab-session is devoted to learning which solution is most suitable depending on the type of application that must be programmed. Thus, if a program has a high computational cost, a multiprocessing approach is likely to provide better performance, especially if the computer's CPU has multiple cores. There are also applications that perform input and output data operations, where the programs are usually blocked and waiting to receive new data blocks. In this case, both solutions, multi-threading and multi-processing, can increase the program performance – although multi-processing may have a higher memory overload.

Exercise 1: multi-threading and multi-processing programming in Python

Multi-threading programming

The following program ('ex1_threads.py') is an example that shows how to use threads in Python. The program creates several threads equal to the number of the cores on the 'base' machine. Each thread executes the `info()` function which prints out the process PID and the parent PID. It then calls the `time.sleep(20)` function and waits 20 seconds. The main function creates the threads with the function `threading.Thread()`. Threads are started with the `start()` method and the main program waits until their completion with the `join()` method. The main program concludes when all the threads have finished. The code in this example is:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing
import time


# Function executed by the threads
def info():
   print ("PID y PPID:", os.getpid() , os.getppid(), flush = True)
   time.sleep(20)


def main():

   # The number of threads created is equal to the number of cores
   num_threads = multiprocessing.cpu_count()
   print ("Number of threads:", num_threads, flush = True)
   jobs = []

   # Variable num_threads has the number of threads to be declared
   for i in range(0, num_threads):
       mythread = threading.Thread(target = info)
       jobs.append(mythread)

   # Threads are started
   for j in jobs:
       j.start()

   # Main program waits until the completion of the threads
   for j in jobs:
       j.join()

   print ("Threads completed")

if __name__ == "__main__":
   main()
```

Download the program to the 'base' machine and execute it with the command:

```
python3 ex1_threads.py
```

The following example implements the same work but using processes.

Multi-processing programming

The following program ('ex1_processes.py') is an example that shows how to use processes in Python. The program creates a number of processes equal to the number of cores on the 'base' machine. In a similar way to the previous example, each process executes the `info()` function which prints out the values of its PID and the parent PID. The `time.sleep(20)` function is then called and it implements a 20-second delay. The main function creates the processes with the `multiprocessing.Process()` function and then starts them and waits for their completion with the `start()` and `join()` methods. The code in this example is:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing
import time

# Function executed by the processes to establish the PID
def info():
   print ("PID y PPID:", os.getpid() , os.getppid(), flush = True)
   time.sleep(20)

def main():

   # The number of processes created is equal to the number of cores
   num_processes = multiprocessing.cpu_count()
   print ("Number of processes:", num_processes, flush = True)
   jobs = []

   # Variable num_processes has the number of threads to be declared
   for i in range(0, num_processes):
       myprocess = multiprocessing.Process(target = info)
       jobs.append(myprocess)

   # Processes are started
   for j in jobs:
       j.start()

   # Main program waits until the completion of the processes
   for j in jobs:
       j.join()

   print ("Processes completed")

if __name__ == "__main__":
   main()
```

Download the program to the 'base' machine and execute it with the command:

```
python3 ex1_processes.py
```

Tasks to be completed

- Execute both programs. For each one, while it is running, on another console/shell run '`ps -C python3 -L -o pid,ppid,lwp,nlwp,stat,cmd`' and '`ps -C python3 -o pid,ppid,lwp,nlwp,stat,cmd`'.

  Both commands show information about the processes that use 'python3'. The difference between them is the '-L' option, which causes the threads to be included as well.

- Considering the output of the above command look at the manual page of `ps` (`man ps`):

  a. Check the PID and PPID of the processes and threads and

compare them with the values obtained during their execution. In the example with threads, why is the PID value constant? In all the cases, why is the PPID value constant?
b. Check the status of the processes during the execution of the wait function (time.sleep(20)). What is their status and what does it mean?
c. Explain the meaning of the values in the LWP and NLWP columns.

- Repeat the execution of the two programs, running in another shell the command, '`ps -C python3 -m -o pid,ppid,lwp,nlwp,stat,cmd`'.
- Use the information obtained to corroborate, correct, or complete your answers.
- Using the `time` command check the run time of the two programs (in this case it is not necessary to execute `ps`).

d. Is there any difference in the run time of the two programs? Why do we get this result?

To be uploaded

- Answers to the previous questions.

Exercise 2: passing parameters to threads/processes and shared memory access

Passing parameters to threads or processes during program execution requires the inclusion of a list of parameters in functions: `threading.Thread()` and `multiprocessing.Process()`. These functions define the list of arguments that will be passed when the thread or process starts running.

Multi-threading programming

The following program ('ex2_threads.py') is an example that demonstrates how to pass arguments to threads. A list of arguments is declared in the thread declaration. For example, in the case `threading.Thread(target = calculate, args=(out_list, i))`, it is specified that each thread receives the `out_list`, declared in the main process, and an integer value, defined with the `i` variable. When the thread is executed, the integer value received in the `i` variable is added to the list. The program creates several threads equal to the number of the CPU cores in the 'base' machine. The code of this example is:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing
import time

#Function executed by the threads to establish the PID
def info():
   print ("PID and PPID:", os.getpid() , os.getppid(), flush = True)
```

```
#Threads starting function to add an element to out_list
def compute(out_list, i,):
   time.sleep(i)
   info()
   out_list.append(i)
   print("Thread",i,"Result",out_list)


def main():

   #Number of threads equal to number of cores
   num_threads = multiprocessing.cpu_count()

   #List where the results will be saved
   out_list = [j for j in range(num_threads)]


   jobs = []
   for i in range(0, num_threads):
       mythread = threading.Thread(target = compute, args=(out_list, i))
       jobs.append(mythread)

   for j in jobs:
       j.start()


   for j in jobs:
       j.join()

   print ("Threads completed.")

   #Print the final result
   print ("Result:", out_list)

if __name__ == "__main__":
   main()
```

The program must be executed on the 'base' machine using the command:

```
python3 ex2_threads.py
```

The following example is almost the same program but implemented using processes.

Multi-processing programming

The following program ('ex2_myprocess_1.py') is an example that demonstrates how to pass arguments to processes. As in the previous example, a list of arguments is included in the process declaration: `multiprocessing.Process(target = calculate, args=(out_list, i))`. The meaning of the variables is the same as in the previous case and the program performs the same work and copies the value of the `i` variable into the list. The code in this example is:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing
import time

# Function executed by the processes to know the PID
def info():
   print ("PID y PPID:", os.getpid() , os.getppid(), flush = True)


# Processes starting function to add an element to out_list
def compute(out_list, i,):
   time.sleep(i)
```

```
    info()
    out_list.append(i)
    print("Process",i,"Result",out_list)


def main():

    # Number of processes equal to number of cores
    num_processes = multiprocessing.cpu_count()

    # List where the results will be saved
    out_list = [j for j in range(num_processes)]


    jobs = []
    for i in range(0, num_processes):
        myprocess = multiprocessing.Process(target = compute, args=(out_list,
i))
        jobs.append(myprocess)

    for j in jobs:
        j.start()

    for j in jobs:
        j.join()

    print ("Processes completed.")

    # Print the final result
    print ("Result:", out_list)

if __name__ == "__main__":
    main()
```

Execute the program on the 'base' machine using the command:

```
python3 ex2_myprocess_1.py
```

After running both programs, the following questions must be answered.

Tasks to be completed

- Check the result of the execution in both programs and the values saved in the output list `out_list`.
- Explain the differences in the obtained results. What is the reason for these differences?

To be uploaded

- Values obtained in output list `out_list` .
- Explanation of the obtained results for each program.

Programming with shared memory between processes

The `Manager` class of the `multiprocessing` library enables the implementation of shared variables between processes. The parent process will own an object, which is returned by calling the function

Manager(). Child processes can access this object to read or modify its content. Unlike parameters or objects of their own, child processes do not obtain a copy of the object but access the parent's copy. An object returned by Manager() supports different types of structures such as: lists; dictionaries; locks; variables; and arrays.

For example, in the following program ('ex2_myprocess_2.py') an object that implements a list is shared and used by all the child processes for saving the data in the output list `out_list`:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing
import time

#Function executed by the processes to know the PID
def info():
   print ("PID y PPID:", os.getpid() , os.getppid(), flush = True)


#Processes starting function to add an element to out_list
def compute(out_list, i,):
   time.sleep(i)
   info()
   out_list.append(i)
   print("Process",i,"Result",out_list)


def main():

   # Number of processes equal to number of cores
   num_processes = multiprocessing.cpu_count()

   # List where the results will be saved
   out_list = multiprocessing.Manager().list(range(num_processes))

   jobs = []
   for i in range(0, num_processes):
       myprocess = multiprocessing.Process(target = compute, args=(out_list,
i))
       jobs.append(myprocess)

   for j in jobs:#
       j.start()

   for j in jobs:
       j.join()

   print ("Processes completed.")

   # Print the final result
   print ("Result:", out_list)

if __name__ == "__main__":
   main()
```

The program must be executed on the 'base' machine using the command:

```
python3 ex2_myprocess_2.py
```

Tasks to be completed

- Check the execution result and explain the values obtained in the `out_list` variable.

- Compare the results with the previous example that used processes.

To be uploaded

- Answers to the above questions.

Exercise 3: parallelisation of programs that require high computing capacity

This section is focused on the evaluation of the performance increase achieved when applications that have a high computational cost are parallelised in Python. In addition, a comparison will be made depending on whether the parallelisation is done using threads or processes. The evaluation will be performed measuring the execution time of the programs with the command `time` using different workloads.

The problem to be solved is the calculation of the PI value using the Monte Carlo method. The calculation is based on the formula to calculate the area of the circumference A = PI*R^2. If the radius is 1, then A = PI and if we focus on the first quadrant of the circumference the area is divided by 4: A = PI/4. To calculate this area, we will use the Monte Carlo method, which is based on the generation of random points whose coordinates are in the first quadrant (1,1) of the plane. These points may or may not fall within the area of the circumference. They will fall within that area if the distance from the random point to the origin is less than 1 and they will be outside otherwise. The probability of falling within the area of the circumference in the first quadrant will be the number of random cases with distances less than 1 divided by the total cases generated. This probability gives an estimate of the area of the circumference in the first quadrant, and so if this result is multiplied by the four quadrants of the plane that make up the circumference with the centre at the origin of coordinates and radius equal to 1, the area of A, and therefore the PI value, will be obtained.

Program with only one process

The program ('ex3.py') in Python that calculates the PI value using this approach is shown next:

```
#!/usr/bin/env python3
import random
import threading
import os
import multiprocessing


# Simulation of the points falling within the circle in the first quadrant
def monte_carlo_pi(index, iterations, result):

   # Counter of the number of cases when the point falls within the
circumference
   count = 0
   for k in range(iterations[index]):
       x = random.random()
       y = random.random()
       if x*x + y*y <= 1:
           count=count+1

   result.append(count)
   print ("Index, PID, PPID, result:", index, os.getpid() , os.getppid(),
result, flush = True)
```

```
        return


    def main():

        np = 1
        print ('Example with %i process' % np)

        # Number of simulated points to calculate PI
        n = 10000000

        # Input list with the points to be simulated for each process
        in_list = [int(n/np) for j in range(np)]

        # Output list with the results of each process
        out_list = list()
        print("List of iterations:", in_list)

        index = 0
        monte_carlo_pi(index, in_list, out_list)

        # Result
        print("List of results:", out_list)
        print ("Estimated PI value:: ", sum(out_list)/(n*1.0)*4)

    if __name__ == "__main__":
        main()
```

In the program, the parameter `np` has the meaning of the number of parallel processes that are executed, in this case only 1, since we are using the non-parallel version `np = 1`. The `in_list` contains the number of iterations that each process or thread would have to perform, in this case the list contains a single element with all the iterations. The `out_list` would be the list where the processes would save their partial result. The variable `n` means the number of total iterations performed and it will be divided by the number of processes. In this first case, a single process will perform all the iterations. The execution of the program on the 'base' machine can be carried out using the command:

```
time python3 ex3.py
```

Tasks to be completed

- Check the execution of the program and find out with `top` the percentage of CPU and memory consumed by the Python process.
- Repeat the execution with different values of 'n', adding 10^7 iterations until reaching $6*10^7$ iterations. Fill in the following table with the execution times obtained.

Iterations	Time
10^7	
$2*10^7$	
$3*10^7$	
$4*10^7$	
$5*10^7$	
$6*10^7$	

Parallelising the program using threads

The objective of this section is the parallelisation of the previous version of the IP calculation program using multi-threading. The program must spawn a number of threads equal to the number of CPU cores. The value `n` will be divided by the number of threads, so that each thread calculates `int(n/np)` iterations. The partial results of each thread will be added to the `out_list` output list from which the final computation of the PI value is made.

Tasks to be completed

- Check the execution of the program and find out with `top` the percentage of CPU and memory consumed by the Python process.
- Repeat the execution with different values of `n`, adding 10^7 iterations until reaching $6*10^7$ iterations. Fill in the following table with the execution times obtained.

Iterations	Time
10^7	
$2*10^7$	
$3*10^7$	
$4*10^7$	
$5*10^7$	
$6*10^7$	

Parallelising the program using processes

The objective of this section is the parallelisation of the previous version of the IP calculation program using multi-processing. The program must spawn a number of processes equal to the number of CPU cores. The value `n` will be divided by the number of processes, so that each process calculates `int(n/np)` iterations. The partial results of each process will be added to the `out_list` output list from which the final computation of the PI value is made.

Tasks to be completed

- Check the execution of the program and find out with `top` the percentage of CPU and memory consumed by the Python process.
- Repeat the execution with different values of `n`, adding 10^7 iterations until reaching $6*10^7$ iterations. Fill in the following table with the execution times obtained.

Iterations	Time
10^7	
$2*10^7$	
$3*10^7$	
$4*10^7$	

Iterations	Time
$5*10^7$	
$6*10^7$	

- Once the table has been filled in, the results obtained for the three versions of the program must be compared. Which version provides the best performance? Give an explanation for the results obtained.

To be uploaded

- Programs for the versions with multiple threads and multiple processes.
- The data collected in the previous tables for each version of the program.
- By analysing the results, explain which parallelisation method is the most suitable for applications with a high computational cost.

Exercise 4: parallelisation of programs that require data input and output

The objective of this section is the evaluation of applications that require the download of information from a networked server and its subsequent processing. In this case, programs perform data input/output transactions that influence the performance of the application, since much of the program execution time will be spent transferring the information from the server. During the download periods, the program will be blocked while waiting to receive new data to resume the execution. Once new data is available, the program will continue its execution until completing the processing of the downloaded block of data.

Setting up a web server in the 'extf' machine

The evaluation of this application involves two networked machines within the NETinVM environment. Specifically, the 'extf' machine, which is configured as a web server, and the 'base' machine where the application that downloads the files and processes them is executed. The backup file of the KVM machines in which the Apache web server has been enabled in the 'extf' machine can be downloaded from the link 'kvm_machines_2020-11-03_12-19.tgz'. This file must be downloaded to the 'base' machine on NETinVM to restore the KVM machines using the option `Restore KVM machines` on NETinVM, with all KVM machines turned off. The 'extf' machine must then be booted. To do this, open the configuration file in the menu of the KVM machines, `Configure my machines` comment on the boot of all other KVM machines, since they will not be used in this session, and add the line:

```
netinvm_run -E extf
```

that starts the 'extf' machine opening a console on the current desktop. Finally, the command 'Run my machines' must be executed.

Once the machine has started, it is possible to check that the web server is working correctly by using the `wget` command. To do this, the following command on the 'base' machine must be executed:

```
wget http://10.5.0.15/2015.tmax
```

where `http` is the protocol used to download the file, `10.5.0.15` is the network address of the 'extf' server machine and `2015.tmax` is the file that is going to be downloaded.

Tasks to be completed

- Using the command `wget http://10.5.0.15/2015.tmax`, the time and average bandwidth obtained at the end of the download can be measured.

The download speed of the files can be changed by limiting the bandwidth of the connections accepted by the Apache server. This limitation can be introduced modifying the server configuration. To do this, open a console on the 'etxf' server for the user 'root'. Edit then the 'ratelimit.conf' file located in the '/etc/apache2/conf-available' folder and change the bandwidth. The content of the file is initially:

```
<Location "/">
   SetOutputFilter RATE_LIMIT
   SetEnv rate-limit 0
</Location>
```

When `SetEnv rate-limit` is 0 there is no limit set. To configure a download limit measured in KB/s, the value 0 must be changed to the new limit. For example, to set a limit of 800 KB/s the file would look like:

```
<Location "/">
   SetOutputFilter RATE_LIMIT
   SetEnv rate-limit 800
</Location>
```

Once the file has been modified and saved, the new configuration must be reloaded on the server 'etxf' using the command:

```
systemctl reload apache2
```

Execute again with this new configuration the command `wget http://10.5.0.15/2015.tmax` on the 'base' machine.

Tasks to be completed

- Measure again the time and average bandwidth obtained at the end of the download.

To be uploaded

- The download time and average bandwidth obtained in both cases.

Program for processing files with environmental parameters

The objective of this section is to evaluate applications that perform the download and processing of environmental parameters saved in data files. Specifically, the estimated daily temperature information on Earth positions identified with their longitude and latitude will be used (a grid covering the entire globe is established). The objective is to evaluate the maximum and average temperature on an annual basis. The temperature files to be used do not have the meaning of the real temperature, but they include the temperature anomaly measured over those points. These daily anomalies files were created considering the temperatures for the 1961–1990 reference period for each contributing station. Therefore, the data files for each year each include the maximum daily temperature anomaly. The format of the files consists of six columns of data:

```
1ª column: Month
2ª column: Day
3ª column: Grid box ID (value range: 1 to 7002, grid spacing = 3.75 deg×2.5
deg)
4ª column: Longitude of lower left corner of grid box (degrees)
5ª column: Latitude of lower left corner of grid box (degrees)
6ª column: Temperature anomaly (whole degrees Celsius)
```

All the information on the temperature files can be found at 'https://www.metoffice.gov.uk/hadobs/hadghcnd/'.

The objective is to create a program that computes the maximum daily temperature anomaly value in a year and the average of the maximum daily anomaly values for that year. The years considered are from 2015 to 2018 and the name of the files in the server is: `2015.tmax`, `2016.tmax`, `2017.tmax` and `2018.tmax`.

Below is the code of the program that computes the maximum and average temperature anomaly within each of the files considered sequentially. The program is called 'ex4.py' and it downloads and processes the four files sequentially. The `download_process` function accepts two parameters: the link to the server where the file to be processed resides and the year to which that file refers. The program repeats the call four times to complete the processing of the four files::
http://10.5.0.15/2015.tmax http://10.5.0.15/2016.tmax http://10.5.0.15/2017.tmax http://10.5.0.15/2018.tmax

The function `download_process` performs the download and processing of the file concurrently. The download is accomplished by the Python library `urllib.request`. Specifically, the `urlopen()` method, which returns a file-like object that can be used to access the downloaded data as if it were a file. The data is processed by lines and the temperature field used to calculate the average of the whole file and the maximum value is extracted.

The program code is:

```
#!/usr/bin/env python3

from urllib.request import urlopen


# Function that performs the download and computes the maximum and average
temperature values
def download_process(link, year):

    points_number = 0

    # Open the link to the file on the server and start the download
    link=urlopen(link)
    print("Downloading and processing", link)

    # When a new line is downloaded, it is processed
    for l in link:

        # Split the line and extract the temperature value
        line = str(l)
        line = line.strip()
        data = line.split()
        temp = float(data[5][:-3])
        # If point_number is greater than 0, update the fields with the new
values
        if points_number != 0:

            points_number += 1

            if (temp > maximum):
                maximum = temp

            difference = temp - average
            average += difference / (number_points)

        # The first line initializes the values
        else:
            maximum = temp
            average = temp
            number_points += 1

    # Close the connection
    link.close()

    # When finished, it prints out the calculated values
    print("Obtained results: year, temp max, temp avg and number of points.")
    print("year:%s tmax:%5.1f tavg:%5.1f n:%i" % (year, maximum, average,
number_points))


def main():

    # List of years to process
    year_list = [ j for j in range (2015,2019,1)]

    # Server address
    address='http://10.5.0.15/'

    # List of links to files on the server
    links_list = [ address+str(year_list[j])+'.tmax' for j in
range(len(year_list))]

    for k in range(len(links_list)):

        download_process(links_list[k],year_list[k])

    print('All the files have been processed')

if __name__ == "__main__":
main()
```

This program must be executed with the command:

```
time python3 ex4.py
```

With this command, the time needed by the program to download and process the four files can be obtained. In this case, the server will be configured without any limit on the download speed, that is, the server will be configured with the parameter `SetEnv rate-limit 0`.

Once the program has been tested, the objective of this section is to create new two parallel versions of the program using threads and processes. The goal is that each thread or process downloads and processes a different file. In this way, it is not necessary to wait until the completion of a call to the `download_process` function before starting the next call, and the four files can be downloaded and processed concurrently. The new versions of the program with threads or processes will create four threads/processes and each one will perform the download and processing of a different file within the range (2015.tmax - 2018.tmax). Evaluate the performance of the new parallel versions of the program measuring the time it takes them to download and process the four files.

Tasks to be completed

- Measure the time and average bandwidth obtained at the end of the download for the sequential program.
- Program the new two concurrent versions (one version using threads and the other version using processes) and check that they give the same results regarding the values of maximum temperature, average temperature, and number of points for each file.
- Obtain the execution times of the three programs without download speed limit in the server, that is, the parameter is `SetEnv rate-limit 0`.
- Repeat the evaluation of the execution times of the three programs when a download speed limit is configured in the server with `SetEnv rate-limit 800`.
- Complete the following table with the measured times:

Program	**Time without download speed limit**	**Time with download speed limit**
Sequential		
With threads		
With processes		

- Check with the command `top` the %CPU obtained in 'base' during the execution of each program with the two download speed limits and complete the following table:

Program	**%CPU without download speed limit**	**%CPU with download speed limit**
Sequential		
With threads		
With processes		

To be uploaded

- Times obtained for each version of the program with the two download speed limits.
- In view of the results, explain which version of the program (processes or threads) is better if the download speed is very fast and the delay in the data access is little compared to the processing time. This situation corresponds to the case with no download speed limit.
- Explain which version of the program (processes or threads) is better if the download speed is low, and the delay in accessing the data is large compared to the processing time. This situation corresponds to the case with a download speed limit.

DSI - Laboratory Session 4: Package, User and Group Administration

27 May 2021

Contents

Goals

- Understand the basis for managing packages, users, and groups.
- Learn to perform simple administration tasks related to software installation, package maintenance, user, and group creation.

Previous work

Before attending the lab session, students must:

- Read this laboratory session document and become familiar with concepts such as repositories, package managers, users, and groups on a Linux machine. These concepts have been used in lectures and problem classes and should be carefully reviewed before the session.
- Review the lab session about the shell, since part of what was learned in that session is going to be used in this session.
- Prepare the machine NETinVM to be able to develop the work in the laboratory.
- Complete the exercises marked as previous work.

To be uploaded

Everything requested in exercises defined as previous work.

Introduction

Tasks to be completed in this session are considered operating system administration tasks. For those who want to complete the explanations on system administration, the following reference may be useful:

> *Unix and Linux System Administration Handbook.* Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley. Prentice Hall.

Moreover, handbooks of common Linux distributions may also be useful, both those included in the distribution itself and those available online. In the case of the NETinVM virtual machine, Debian 9.5 is used, and the online documentation can be found at www.debian.org.

Session preparation

NETinVM virtual machine

In this lab session, you must work as system administrator and make significant changes to the system. To work with freedom and security, the NETinVM virtual machine will be used. For this reason, it is necessary to consider that:

- Students must come to the laboratory with a ready virtual machine, either on the student's own laptop or on a USB hard drive (it must be a hard drive as a USB memory stick will not work for this purpose).
- It may be necessary to undo the changes if mistakes are made, or simply repeat an exercise. To do so, a **snapshot** must be captured before starting the lab session and, subsequently, each time an exercise is completed. In this way, if something goes wrong, it is possible to return to the previous state.
- In this session, it is not recommended to use the NETinVM copy installed on the laboratory computers. Nevertheless, they can be used if none of the students in the group has a laptop or a USB hard drive, but it is necessary to consider that:
  - It is not going to be possible to turn off the machine during the whole session, because it will provoke a full data loss.
  - The use of snapshots is not going to be possible during the session.

How to become administrator

During the session, it will be necessary to perform the administration tasks such as 'root' (the administrator on Unix systems). To **become an administrator**, simply use the following command:

```
su -
```

Additionally, during the session it will be necessary to **edit text files as system administrator**. This can be done with the 'nano' editor in the terminal (after becoming administrator with 'su -'). The 'nano' editor is pre-installed in NETinVM. A graphical application such as Kwrite is not recommended, because many lines that implement the graphical interface are executed with administrator privileges, and this is a potential security threat.

Previous exercise 1: remembering how to work from the command line

After reviewing (if needed) the first two lab sessions, briefly answer the following questions:

a. What numeric identifier does the administrator of Linux systems have?
b. And what name? (hint: become an administrator with 'su -' and use 'id' to find the answer. You must work on the machine `base` of NETinVM).
c. Write the command to create the folder 'd1' in the directory '/home/user1'
d. Write the command to copy the file 'f1' (which is in the folder '/home/user1') to the folder '/tmp'.
e. Write the command to delete the file 'f1' (which is in the current directory).
f. Write the command to delete the directory 'd1' (which is in the current directory), assuming it is empty.
g. And, if it is not empty, what command should be used to delete that directory and everything it contains?
h. With what command can the file name 'f1' be changed to 'f2'?
i. With what command can the directory 'd1' be renamed to 'd2'?

Tasks to be completed

1. All the questions must be answered.

To be uploaded

- The answers to the questions.

Package management system

A *package manager* program keeps a record of the software installed on the computer. Additionally, it enables installing new programs, updating already installed programs to more recent versions or easily removing programs. A program generally consists of several files that are grouped together in the concept of a package. A particular package can also have dependencies on other packages that must also be installed to ensure the proper functioning. A package manager program solves all dependencies and recommends the installation of other needed packages (as can be read in the Debian manual):

- If a package A **depends** on another package B, then B is necessary for A to work properly. For example, the `gimp` package depends on the `gimp-data` package to enable the GIMP graphical editor to access its critical data files.

- If package A **recommends** another package B, then B offers significant extra functionality for A that would be desirable in most circumstances. For example, the `mozilla-browser` package recommends the `mozilla-psm` package which adds secure data transfer capability to Mozilla's web browser.
- If package A **suggests** another package B, then package B offers functionality to A that may improve A but is not necessary in most cases. For example, the `kmail` package suggests the `gnupg` package which contains encryption software that can be used by KMail.
- If a package A **conflicts** with another package B, the two packages cannot be installed at the same time. For example, `fb-music-hi` conflicts with `fb-music-low` because they offer alternative sets of sounds for the Frozen Bubble game.

There are several software packaging formats and this depends on the Linux distribution. In the Debian case and its derived distributions, like Ubuntu, packages have the extension **.deb**. *Red Hat*-based distributions have packages with the **.rpm** extension. In contrast, the **.tgz** format is common on Unix, compressed with the zip GNU compressor. There are programs to convert packages from one format to another such as alien, but a package does not always work correctly if it is installed after it has been transformed from a different type of package, designed for another distribution.

A package manager helps the administrator in managing the set of packages that are installed on the system. There are several package managers, depending on the Linux distribution. Package managers are high-level tools, some even with a graphical interface, that are based on lower-level commands. For example, in the case of Debian and its derived distributions, the low-level tool is **dpkg**, which is used to install Debian packages, but does not manage dependencies, and so it is always more useful to use higher-level tools than take dependencies into account. In the case of Debian, **apt** (short for *advanced package tool*) offers more advanced package management than `dpkg`. Additionally, the `apt` program groups the most common commands and options of existing programs for managing dependencies, such as `apt-get` or `apt-cache`, offering a simple interface. There are also other programs that offer an interactive interface in text mode, such as **aptitude**. As additional documentation, at the end of the session is an introduction to the `aptitude` program, along with several voluntary exercises to be completed outside the session (section 'The aptitude program interface').

Users and security

On UNIX systems, users who have access to the system are identified by a unique number called **user ID**. Each system process has an **effective user ID** associated with it, which indicates which are the access permissions of the process (initially, the same as the user it is associated with).

To be able to control, in a unified way, the access of groups of users to resources, **groups** are defined; each user belongs to one or more groups and each process has associated one or more **group IDs** that set which group permissions it has (all those that the associated groups possess).

All system files have associated a user ID and a group ID and some permissions (read, write, and execute) associated with those identifiers (in addition to a third group of permissions that indicates the level of access for users and groups that do not own the file).

Databases are used to store the information of users and groups of the system. These data are traditionally kept in files ('/etc/passwd' for users and '/etc/group' for groups). For a description of the format and the information stored in these files, the manual pages of the files can be read ('`man 5 passwd`' and '`man 5 group`').

Traditionally, user passwords were stored in the file '/etc/passwd' in encrypted form, but since the file is visible to all users, this represented a security threat because the encryption system is known and if the encrypted version of a key is copied, thousands of tests can be made to try and decrypt it without anyone's knowledge.

To solve this problem, the **shadow passwords** system was introduced, which instead of storing the keys in '/etc/passwd' saves them in the file '/etc/shadow', a file that is used to validate and change passwords and is only available to users of a special group. The system that is going to be used in the lab session is configured to use this system. For more information on the file format see '`man 5 shadow`'.

File owners, groups, and permissions

File system objects have the user who creates them as default owner, a group (the group of the user who creates them) and a set of permissions. For example, from the file 'file':

```
labdi:~/tmp # ls -l
total 8
drwxr-xr-x 2 root root 4096 Nov 16 12:21 dir
-rw-r--r-- 1 root root   29 Nov 16 12:19 file
labdi:~/tmp #
```

It can be said that:

- it is a normal file, since the first letter is '-' (notice the difference with 'dir', which is a directory, with a 'd' as the first letter)
- permissions for the owner are 'rw-' (read and write, but not execution)
- the permissions for the group are 'r--' (read only)
- the permissions for the rest of the world (that is, those users who are neither the owner nor the group of the file) are 'r--' (read only)
- has a unique alias
- belongs to the 'root' user
- belongs to 'root' group
- its size is 29 bytes
- was created on 16 November at 12:19

The command `chown` enables changing the owner of the file and, optionally, its group. For example:

`chown juan file`
: makes the user 'juan', the owner of the file, if this user exists;

`chown juan.DSI file`
: makes the user 'juan' the owner of the file, and also makes the group of the file 'DSI'. Again, as long as both exist.

In the same way, `chgrp` enables change the group of a file. For example:

`chgrp DSI file`
: makes the file group 'DSI'.

Also, the video 'chown-chgrp.mp4' shows some examples of the use of these two commands.

Finally, the command `chmod` is used to change the permissions, for which there is also an explanatory video ('chmod.mp4'). This command expects a series of actions to be carried out, as in the following example:

```
chmod u=rw,go=r file
```

This command includes two actions separated by commas. The first 'u=rw' indicates that the owner (the **user**, letter 'u') must have read ('r') and write ('w') permissions on the file 'file'. The second, 'go=r', indicates that the members of the file's group (**group**, 'g') and those of the rest of the world (**other**, 'o') must have read permissions exclusively.

In addition to setting permissions, permissions can be added to existing ones using the '+' action:

chmod u+w file

This command would add the write permission ('r') for the owner (user, 'u') to the file 'file'. (Note that in this case there is only one action).

Finally, permissions can be removed:

chmod go-w file

In this case, write permission ('w') is removed for both group (group, 'g') and the rest of the world (other, 'o').

Finally, a quite common case is to remove all permissions for the group and/or the rest of the world. For example:

```
chmod go= file
```

This command removes all permissions for the group and the rest of the world, as it applies an empty permission set (no permission to the right of the '=').

Both `chown`, `chgrp` and `chmod` can be applied to groups of files using the shell expansion and also recursively using the '-R' option.

Previous exercise 2: becoming familiar with the user and group files

After reading the introduction to the section Users and security and checking, if needed, the manual pages of the 'passwd' and 'group' files, the following questions must be briefly answered:

a. What user of NETinVM has the numeric identifier 101?
b. What shell does the user 'gnats' of NETinVM have?
c. What is the name of the default group of the user 'user1' in NETinVM?
d. What numerical identifier does this group have?

Tasks to be completed

1. All the corresponding information must be read.
2. All the questions must be answered.

To be uploaded

- The answer to the questions.

Package management

Warning

- Before each exercise it is a good idea to take a snapshot of the virtual machine, and save it with a descriptive name, to be able to revert the machine in case something goes wrong. In particular, during the lab session updating the entire system must be prevented, as it would require a lot of time and disc space.

The **apt** tool enables installing, updating, or removing programs, managing repositories and performing queries. The most common tasks are going be introduced in this section:

```
apt update
```

`update` is used to update available package information from all available configured sources (repositories). For each installed package, the latest available version is verified. This option does not update or change any system package. A typical output when executing this command would be (it may be different):

```
root@base:~# apt update
Ign:1 http://ftp.es.debian.org/debian stretch InRelease
Get:2 http://ftp.es.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:3 http://security.debian.org/debian-security stretch/updates InRelease [94.3
kB]
Hit:4 http://ftp.es.debian.org/debian stretch Release
Fetched 185 kB in 0s (391 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
243 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

`Ign` expresses that this repository is ignored - probably because the font is outdated and there are no valid updates. `Get` shows that the repository does contain relevant information on new packages and versions and that it is therefore updated from that source. `Hit` indicates that the system already has this information and is up to date. In the case of the example shown, it is also reported that 243 packages can be updated.

In fact, if the command `apt update` is immediately executed again, in the case of the previous example we would get:

```
root@base:~# apt update
Hit:1 http://security.debian.org/debian-security stretch/updates InRelease
Ign:2 http://ftp.es.debian.org/debian stretch InRelease
Hit:3 http://ftp.es.debian.org/debian stretch-updates InRelease
Hit:4 http://ftp.es.debian.org/debian stretch Release
Reading package lists... Done
Building dependency tree
Reading state information... Done
243 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

It must be noted that now there is no `Get`. Those previous `Get` have passed to `Hit`, since the information in the repositories had just been updated. In either case, no packages are updated. To do this, you must execute:

```
apt upgrade
```

`upgrade` is the option used to update the system using a sequence of actions. First, it lists the packages that were previously installed but are no longer needed. Additionally, it shows new packages that will be installed, plus the packages that will be kept unchanged and finally the packages that will be upgraded (and indicating the amount of additional disc required). New packages required to satisfy dependencies will be installed, but installed packages will not be removed. If it is necessary to remove a package to update another one, this action is not performed.

```
apt full-upgrade
```

`full-upgrade` does the same as `upgrade`, but it also removes packages, if needed, to update others.

```
apt search name
```

This is used to search for the string `name`, both in the list of packages in the repositories and in their description. This is useful if the name of a package is unknown, but instead some keywords are known that can describe it.

```
apt show package
```

`show package` shows information about the `package`: version; priority; source; maintainer; dependencies; size and a description of its functionality.

```
apt list [--option]
```

`apt list` displays the list of packages specified with `--option`. For example, this option can be

`--installed` to show installed packages or `--upgradeable` to show packages that are upgradeable.

`apt depends package`

`depends package` shows the packages that `package` depends on. Typically, three categories of packages are shown: *dependencies* (packages that are required and therefore must be installed); *recommendations* (packages that are not necessary but are installed by default); and *suggestions* (packages that could be installed but which are not installed by default).

`apt install package`

`install package` installs the package `package`. It asks for confirmation after reporting the amount of disc required. To prevent the installation of the *recommended* packages use:

```
apt --no-install-recommends install package
```

Additionally, to install a package and its *suggested* packages (which are not installed by default, unlike the *recommended* packages), execute:

```
apt -o APT::Install-Suggests="true" install package
```

`apt remove package`

`remove package` removes a package. It asks for confirmation after reporting the amount of disc to be released.

`apt purge package`

`purge package` removes the `package` package and any additional software configuration files. It asks for confirmation after reporting the amount of disc to be released.

`apt autoremove`

`autoremove` removes packages that were installed to satisfy dependencies but are no longer required. It is very useful for removing unnecessary software and freeing disc space.

`apt rdepends package`

`rdepends package` shows the **reverse dependency**, that is, packages that depend on `package`.

Exercise 1: updating system packages

The status of the NETinVM machine must be known. To do this, the information in the repositories must be updated and so it will be possible to discover how many updates are pending.

Warning

- Be careful and avoid updating the entire system in the lab session, as the process would be slow and would require a lot of disc space. For this reason, it is important to read carefully any message appearing in the shell when prompted for command confirmations.

Tasks to be completed

- The repositories must be updated. Additionally, it must be established which sources are already up to date, which have been updated at that moment, and which are out of date (just copy into a text file and describe the output of the command `apt update`).
- A list of upgradeable packages must be generated, and saved as a text file:

`upgradable-packages1.txt`

- Upgrade the installed version of **firefox-esr**. First, it is necessary to stablish which version is installed and which is the latest version. In the lab session, a different application may be proposed for updating. This makes no difference and the work to be done will be the same.
- The name of the packages that are going to be installed in a mandatory manner with the proposed update (appearing after the word `Depends`), recommended packages, that will be installed by default (appearing after the word `Recommends`), and suggested packages (appearing after the word `Suggests`) are saved in a file with the name: `dependencies-firefox-esr.txt`.
- How much disc is needed for the update must be described and finally the software must be updated (`apt install firefox-esr` in this case). The process should not take more than a couple of minutes.
- It should be checked that all the dependencies and recommendations have been installed. To do this, it is proposed checking only the first package of each category (use `apt list package-name`). Check that the suggestions have not been installed: for instance, testing with the first package that appears as a suggestion.
- The output of the command `apt update` must be copied. The package installation, together with its associated packages, should have decreased the number of upgradeable packages.
- Finally, a new list of upgradable packages must be generated with the name `upgradable-packages2.txt`. Packages that have already been updated should not be present.

To be uploaded

- Output of the command `apt update` before the package update.
- File `upgradable-packages1.txt`.
- What version of the proposed upgradable software was previously installed and to which version has it been upgraded?
- Output of the command `apt update` after the package update.
- File `upgradable-packages2.txt`.

Exercise 2: installing new software

Installing new software is a similar procedure to updating already installed packages. It is proposed as an exercise to install the classic games package that includes **minesweeper** on the NETinVM machine.

Tasks to be completed

- The package that includes *minesweeper* in its name must be found. Use the search option: `apt search minesweeper`.
- Information on the version of the package to be installed, packages it depends on, recommended and suggested packages, and the amount of disc required, must be obtained in a similar way to the previous exercise.
- The package must be installed, together with its recommended packages (default option).
- Several games are going to be installed in `All Applications > Games >`. Verify that at least one minesweeper works correctly :)

To be uploaded

- Output of the command `apt search minesweeper`.
- Output of the command `apt show games-minesweeper` before the software installation.
- Output of the command `apt list games-minesweeper` after the software installation.

Metapackages and *tasksel*

A metapackage is a list of packages that install with a common purpose a group of applications, libraries with their documentation, and dependencies. Typically, a set of packages which offer a functionality is called a task. This possibility offers an easy way to install software to dedicate a machine to a specific purpose: print server, office management, general-purpose laptop, etc. It is even possible to use tasks to dedicate a machine to a more specialised use: such as astronomy or android application development. A task is a list of packages and/or metapackages with a specific purpose. Metapackages can also be installed with `apt`, as described in the previous section, but a more user-friendly application such as `tasksel` is commonly used.

`tasksel` is a program that presents a simple interface for users who want to configure their system to perform a specific task. This program is usually used at the beginning when installing a machine, but it can also be used any time later. For example, when executing:

```
tasksel --list-tasks
```

The tasks available on the system are displayed, with an 'i' to show that they are installed or a 'u' to show that they are not (uninstalled). It is possible to see the tasksel options in the manual. Task descriptions available in the system are at `/usr/share/tasksel/descs` and have the `.desc` extension. They are typically configuration files. Each metapackage has the following fields (not all are essential):

- `Task`: is the name of the task or metapackage.
- `Relevance`: a single digit from 1 to 9 that specifies the relevance of the task, where 1 is higher and 9 is lower. 5 is the default value.
- `Enhances`: Informs us that the task should only be installed if the tasks described in this field are also installed.
- `Description`: These are two lines that describe the task. The first line should be very short. The following text can be longer, but must be indented with the first line.
- `Key`: Necessary packages that must be previously installed so that the task can be installed.
- `Packages`: List of packages that shape the task and which will be installed.

- `Test-*`: Informs us that the task can be installed automatically from a test program located in `/usr/lib/tasksel/tests/`
- `Section`: Area where the type of task, location, server, or user is described.
- `Parent`: Enumerates the tasks that can be parents of this task or metapackage.

If the command `tasksel` is executed directly, a graphical interface is displayed in text mode, which shows both installed and available tasks. It is possible to move through the tasks with the keyboard arrows. A task is selected by moving the cursor to its position and pressing the space bar. When all the tasks to be installed/uninstalled have been selected, it is possible to select OK by pressing TAB, and then just pressing ENTER.

Exercise 3: testing tasksel

The following tasks must be performed with `tasksel`.

Tasks to be completed

- Execute `man tasksel` and take note of the two main options to install and remove tasks.
- Make a list of which tasks are installed in the NETinVM machine. To do this, `tasksel` can be used, either with the text-mode interface or from the command line.
- Being root, the games-minesweeper task must be removed using `tasksel` in text-mode. The system will no longer have games and the `Games` folder should not appear within `All Applications`.
- It is proposed to build a task file that will install the chess games available in the games-board metapackage. To do this, it is proposed:
    - Discover which packages may be worthwhile including. To do this, simply use the appropriate option of `apt` to search for which packages include games-board. Those that include the word **chess** (at least 2 games) would be suitable :)
    - As root, create a file named *my-task.desc* in `/usr/share/tasksel/descs`. The task **chess** with the following options must be described:

```
Task: chess
Relevance: 8
Description: Chess Games
 A couple of chess games to play for a while
Key:
Packages: list
 package1
 package2
Section: user
```

  It is a good idea to see other .desc files in the directory to confirm the syntax of the fields.

- The chess game package must be installed using `tasksel` in graphic mode. The `Chess games` task should appear.
- Check that the programs have been installed. Note **gnuchess** does not have a graphical application and is invoked from the shell.
    - Remove the chess game package using `tasksel` in command mode: `tasksel remove chess`

to be uploaded

- Description of the two main tasksel options.
- List of tasks installed in NETinVM.
- File .desc with the creation of the chess task.

Backups

A simple way to make a backup is with the command '`rsync`'. Its main characteristic is that it uses an algorithm that copies only the files and directories that have changed, and within each file it only transfers the information that has changed, ans so avoids sending the complete file.

To make a local copy (as opposed to making it over the network) simply the source and destination folders must be specified. For example, to recursively copy 'd1' into the directory 'd2' (like 'd2/d1') use:

```
rsync -aP d1 d2
```

The '-a' option is used to apply the *archive* mode, which respects the owner of the file, its group, and permissions. The '-P' option is used to show a progress indication (especially useful when copying large files.) Note that the destination directory may well be on a removable device, such as a USB stick.

It is **important** to note that to copy the contents of the directory 'd1' into the directory 'd2', the source path must end in '/'

```
rsync -aP d1/ d2
```

In this second example, the file 'd1/f1' would be copied as 'd2/f1'. Instead, in the first example (without '/' at the end), 'd1/f1' would be copied as 'd2/d1/f1'.

Note

`rsync` can work across the network and is often used to synchronise directory tree structures in different machines. By sending only the changes, with the additional option of sending them compressed, `rsync` is usually very efficient. In addition, `rsync` can work with SSH, a protocol commonly used to manage servers and that will be studied in the 'Networks and security' subject, in the second semester. For example:

```
rsync -avz server1:/copies/documents desktop/documentation
```

With this command, the files from the 'documents' folder of the server1 are going to be copied into the 'desktop/documentation' folder of the local machine. The *archive* parameter ('-a') is used to specify that user names and permissions that documents may have must be kept. The '-z' parameter is used for sending the files compressed and the '-v' option to force the display of information during the process.

Exercise 4: making backups with `rsync`

A backup copy of the '/home/user1/Documents' directory on a *pendrive* must be made. If no pendrive is available, the exercise is identical, except that the copy will be made in another directory on the machine.

If a pendrive is available, when the pendrive is connected, if prompted, it must be specified to connect to the NETinVM virtual machine and not to the host. Once this has been done, the device is going to be automatically detected and the file system mounted. Typically, it is going to be mounted in '/media/user1/', but to confirm this, it is possible to find the mounting location with the command `df`. In '/media/user1/' a directory is going to appear with the name of the pendrive. The name of the pendrive is going to be assumed to be MYPENDRIVE, but of course it must be replaced by the given name in each case. If no pendrive is available, the directory '/tmp/backup' is going to be the destination directory, which must be created previously with the command `mkdir`.

Note that, by default, rsync does not delete objects in the destination directory. If this possibility must be allowed, then the '--delete' option must be included.

Tasks to be completed

- Using `rsync`, create a backup copy of the directory '/home/user1/Documents' on the pendrive: '/media/user1/MYPENDRIVE'.
- Any file must be changed in '/home/user1/Documents'. The backup must then be updated using `rsync`. Verify that the backup has been correctly updated.
- A file in '/home/user1/Documents' must be deleted. The backup copy must then be updated again.
- Check that the file has also been deleted in '/media/user1/MYPENDRIVE'. (If not, the command must be properly modified to achieve this).
- Finally, changes must be made in '/home/user1/Documents' after the backup: edit a file, create a file, and delete a file. The backup should be used to automatically and transparently restore the state of '/home/user1/Documents'. It must be verified that all modifications have been discarded.

To be uploaded

- The command used to backup the directory and to update it.
- The command used to undo changes and restore the state of '/home/user1 /Documents' from the backup.

User and group creation

A basic task of a system administrator is to add, modify, or delete user accounts and/or user groups. In this section we are going to register new users and groups. To create accounts, it is enough to simply modify the password and group files with the user information, create their base directories, and copy the files found in the '/etc/skel' directory to the base directories of the users. This task can be prone to errors, so several utilities available in the operating system are used (such as `useradd` and `groupadd`) that help simplify the process.

Exercise 5: creating the initial users and groups

The group 'students' and the two user accounts detailed below must be created, after reading the corresponding manual pages, using the commands `groupadd` and `useradd`:

Login	UID	GID	COMMENT
stu1	5001	students	Student one
stu2	5002	students	Student two

It should be noted that:

- First, the group must be created and then the users can be created to establish 'students' as the default group for 'stu1' and 'stu2'.
- The numerical identifier of the group 'students' must be 700.
- The command `useradd` does not create the user's home directory by default, so it is necessary to include the option '-m'.
- Both `groupadd` and `useradd` assign right default values (established in the system configuration). In this way, to create the group, it will be enough to specify, the name of the group and its identifier. To create the users specify: the user name; the numeric user identifier; the default group; the comment (with the full name) and the '-m' option.

To finish the exercise, a correct password for each of the previous users must be assigned (the easiest method is to use the `passwd` command) **and check that users can connect to the system**. (Hint: in a terminal that is 'user1', the command '`su - stu1`' can be used to test 'stu1' user. Next, in another terminal, in a similar way, '`su - stu2`' can be used.

There are also commands that enable deleting users or groups from the system: `deluser` or `delgroup`. Moreover, the first command can be used to delete a user from a group. The manual should be reviewed if they are going to be used.

Tasks to be completed

1. The group 'students' must be created according to the requirements.
2. Users 'stu1' and 'stu2' must be created according to the requirements.
3. A valid password must be assigned to 'stu1'.
4. Check that 'stu1' can work in the system. (Hint: use 'su - stu1').
5. Steps 3 and 4, must be repeated for user 'stu2'.
6. The starting directory for 'stu1' and 'stu2' has to be found.
7. The shell for 'stu1' and 'stu2' must also be found.

To be uploaded

- Brief description of the commands executed to accomplish the task.
- The output of the command '`grep stu1 /etc/{passwd,group,shadow}`'.
- The output of the command '`grep stu2 /etc/{passwd,group,shadow}`'.
- The output of the command '`ls -laR /home/stu*`'.
- The output of the command '`ls -l /home`'.
- The output of the command '`id stu1`'.
- The output of the command '`id stu2`'.
- What is the home directory of the new users?
- Which shell do they have?

Exercise 6: creating a shared directory ('/home/datascience')

The two users recently created are going to work together on a secret project called 'datascience' and so they must share all the files that are in the '/home/datascience' directory.

The system must be prepared, so that both have full access to the files in this directory and no one else in system can have any kind of access to the directory or its content.

To do this, it must be considered that:

- Users participating in the project must all belong to an additional group created specifically for that project (for example, 'datascience'). To create the group use the command `groupadd`, and to add users to that group use the command `usermod`.
- It must be considered that the same user can work on several projects, so the project group (in this case 'datascience') must be an additional group. Their default group (in this case) should remain 'students'.
- Owner, group, and permissions of the directory are those that authorise which users can access the content and which operations they can perform with it.
- The group must logically be the group 'datascience'. The most appropriate owner would be the administrator ('root' user), since in this way only users included in the group can access the directory (the administrator can access it anyway, even if he is not the owner). (Hint: use `chown` and optionally `chgrp`).
- The permissions must enable full access to all members of the group, and no access rights to the rest of the system users. (Hint: to assign permissions use `chmod`).
- In addition, it will be helpful to activate the *SGID bit* in the permissions associated with the directory group. (Find out why it is proposed as an exercise).

Tasks to be completed

1. Create a group for the project with GID 701.
2. Add the users 'stu1' and 'stu2' to the new group.
3. Create the directory '/home/datascience'.
4. The owner and group of the directory must be changed.
5. Appropriate permissions must be assigned.
6. The *SGID bit* in the directory must be set: '`chmod g+s /home/datascience`'.
7. Check that both 'stu1' and 'stu2' can create files in '/home/datascience'.
8. Check the group of files created by stu1 and stu2 in '/home/datascience'.
9. Check that other users cannot access the directory. (Hint: use 'user1').

To be uploaded

- Description of the commands executed to accomplish the task.
- The output of the command '`egrep 'stu|datascience' /etc/group`'.
- The output of the command '`ls -l / home`'.
- The output of the command '`id stu1`'.
- The output of the command '`id stu2`'.
- Brief explanation of what is the *SGID bit* applied to a directory group.

Exercise 7: sharing files through the '/home/datascience' directory

To check the result of 'Exercise 6: creating a shared directory ('/home/datascience')', the user 'stu1' has to create a file (for example, 'stu1.txt') within the directory '/home/datascience' with the right permissions so that all members of the 'datascience' group can modify it.

To do this, users 'stu1' and 'stu2' must:

Tasks to be completed

1. 'stu1' must create the file 'stu1.txt' in the shared directory.
2. Check that 'stu2' can modify the file 'stu1.txt'.
3. If needed, 'stu1 must modify file permissions (owner and group should not be necessary) and check again as 'stu2'.
4. Check that both users can modify the file.

To be uploaded

- Description of the executed commands to accomplish the task.
- The output of the command '`ls -laR /home/datascience`'.

Software installation and maintenance with *aptitude*

This section is left as additional optional work that can be completed at home on a voluntary basis. If students had time in the laboratory session, after completing the compulsory part, work can be started in the laboratory. In any case, this *aptitude* tutorial is considered optional and no material from this section needs to be uploaded.

Optional exercise 1: installing the aptitude program

To install any program, after becoming an administrator, it is necessary to update the information about the available packages and their sources. To do this, execute:

```
apt update
```

Once the package information has been updated, the installation of `aptitude` can be done. To do this the following command must be executed:

```
apt install aptitude
```

Tasks to be completed

1. The information from the apt man page must be read.
2. The following questions must be answered:
   - How many packages can be updated after executing `apt update`?
   - What is the 'purge' option used for?
   - What is the 'autoremove' option used for?
   - What is the output of the command '`apt show aptitude`'.

The aptitude program interface

In the previous exercise, the `aptitude` program was installed, so it should now be available on the NETinVM machine. Although any modification of the installed packages must be made as administrator, it is possible to execute it as **user1**. In this way, it is possible to navigate and become familiar with the menus and the package system without risk. Later, it is possible to enter the administrator password, and proceed to modify the system.

From a shell, and as `user1`, the program `aptitude` must be launched. In the same shell, without opening any additional window, the graphical environment of the program will appear in text mode. The screen is divided into several areas:

The upper part, with a blue background, is the menu bar. The top line shows the main options:

```
Actions  Undo  Package  Resolver  Search  Options  Views  Help
```

The second line shows information about the keys that must be pressed to execute some important commands. The third line shows information about the version of aptitude and about the chosen view. The commands on the top line can be accessed with the mouse, selecting the appropriate option, or by using the key combination 'Crtl' and 'T' (this information also appears in the help of the second line, as 'C-T: Menu'). Other options are 'Crtl' and 'Space' or directly 'F10'. The menus can then be navigated with the keyboard arrows. Navigation on the top line can be exited by pressing the same key combination again. The 'Q' key from the 'Actions' submenu is used to completely exit the program, and for doing this a confirmation is needed. If the answer is 'No', it exits the menus, but not the program. Also 'Q' is used as an escape in other menus. Additionally, if we have mistakenly selected one or more packages to delete or install, pressing 'Crtl' + 'u' will undo these changes (similarly to 'Crtl' + 'z' in Windows).

Below the blue area there is a drop-down information area through which it is possible navigate, either with the mouse or with the cursor arrows and then pressing 'Enter':

```
--- Security Updates (17)
--- Upgradable Packages (37)
--- Installed Packages (2147)
--- Not Installed Packages (54716)
--- Virtual Packages (14431)
--- Tasks (217)
```

The names are self-descriptive and the numbers in each category may be different. It is possible to press 'ENTER' on a category to display it, then it will appear with the prefix '--'. Pressing again closes the category. If '[' is pressed, all the subcategories that depend on a category are opened. If ']' is pressed, the present subcategory is closed. The cursor is used to move through the lists. Common keys, such as 'Up', 'Down' or 'Home' and 'End', can be used as in any editing program. In this way, if categories from top to bottom are described, the following is obtained:

- Security updates.
- Upgradable packages: installed packages for which an update is available.
- Installed packages but without any update.
- Not installed packages: packages available in the distribution that are not installed.
- Virtual packages: there are sometimes several packages that offer similar functionalities. In this case, it is useful to define a virtual package whose name describes that common functionality. (Virtual packages only exist logically, not physically; that is why they are called virtual.) So any other package that requires that function can simply depend on the virtual package without having to specify all possible packages individually.
- Tasks: groups of packages that provide an easy way to select a predefined set of packages for a specific purpose.

The final part of the screen, after a blue horizontal line, includes a description of the navigation zone selection: package groups, packages, etc.

Optional exercise 2: becoming familiar with aptitude. Installing and removing packages

Install the 'aptitude' hmtl documentation in Spanish from the 'aptitude' program itself. To do this:

- As `user1`, `aptitude` must be executed.

- First, available packages sources must be updated. To do this, select `Actions> Update package list` or press the 'u' key. It will be notified that the user is not an administrator, and the option to become root will be offered.
- The option to become root must be chosen, and the password must be introduced (copy and paste from the passwords.txt file).
- The package, whose name is **aptitude-doc-es**, is not installed, so it should be searched under `Not Installed Packages > doc> main`, which could be quite laborious. Another option is to select *Search* (or just pressing the slash '/') and type the string 'aptitude'. Packages with the referenced string will appear. By pressing 'n' to go forward and 'N' to go backwards, it is possible to move between the various occurrences until the desired package is found.
- Once the cursor is on the package, it can be selected for being installed by pressing '+'. Similarly, it can be deselected with the '-' sign. In this case, it must be installed and so the '+' key must be pressed, which will show an 'i' next to the 'p' indicating that this package is going to be installed when the action is consolidated. The required disc space for installation will be shown.
- To install the package, select the menu `Actions > Install/remove packages` or directly press 'g'. The program warns us of possible dependencies on additional packages. Press 'g' to install the package with its dependencies.
- Check that the package now appears in the category of installed packages. Additionally, it is possible to go to the installation path to verify that the package has been installed: `/usr/share/doc/aptitude/html/es/`

Tasks to be completed

1. The requested documentation, in html and Spanish, must be installed.

Optional exercise 3: fixing faulty installations

The aptitude program detects if any package is *broken* because dependencies are not met (not all mandatory packages are installed on the system). This should not happen if software is always installed/removed with a tool that considers dependencies between packages. Despite this, a software installation/removal tool may have been misused and this error should be corrected. To illustrate this capability, we will simulate a mistake, remove a package that should not have been removed, and see how aptitude helps to detect and correct this problem. **Before starting the exercise, a snapshot must be taken, and any instance of aptitude should be closed**.

GIMP or *GNU Image Manipulation Program* is a bitmap drawing program that is similar to Windows *paint*. As *user1*, it can be checked that the program works correctly by invoking it from a shell (the executable file is called '`gimp`') or from the application launcher (KDE menu> Applications > Graphics > Image Editor). The program should then be closed as it has only been executed to verify that it is working properly.

To force a wrong installation, the package `gimp-data`, on which the main package *gimp* depends on, is going to be removed, and then it is going to be fixed. To remove gimp-data, with superuser privileges, the following command must be executed:

```
dpkg -r gimp-data
```

What messages appear? Why is not recommended to remove the package `gimp-data`?

Ignore the warnings and execute:

```
dpkg --force-all -r gimp-data
```

What has happened in this case? Open the gimp program again and see if it works.

The `dpkg` command should never be used, unless we know very well what we are doing. Moreover, warnings should never be ignored, and the system should never be forced into this incorrect behaviour.

Now aptitude must be opened (acting as user1 for security reasons). The message *#Broken: 1* appears on the third line in the top blue area. This indicates that there is a broken package. Additionally, a red message should appear at the bottom of the window, indicating actions to fix this problem. By pressing '.' or ',' we go forward or backwards between the possible actions to solve the problem, which consist of either installing the removed package (`gimp-data`) or removing the package (`gimp`) which needs the removed package. The action number, among all possible actions appear in square brackets. In this case, installation of the missing package will be chosen to fix the problem, so that option should be left active.

By pressing 'b' or choosing from the `aptitude` menu `Search > Find Broken` the broken package will appear in red. With what character is the broken package marked?

It is possible press ENTER and see information related to the package and its dependencies. In this section, the broken dependencies appear (red lines). What package is shown as required and which version of it?

To solve the problem, just select the installation of the gimp-data package (by pressing '+'), or from the `gimp` package, apply the proposed solution with '!'. Warning: ensure that the chosen option has been to install the dependencies and do not delete gimp. Note: with 'Crtl' + 'U' any selection can be undone.

The installation solution must be applied. The red message is going to disappear and a 'B' will appear showing that, when the installation is consolidated, the broken package will be fixed. If it has been mistakenly chosen to delete gimp as a solution, the gimp line will appear in purple, and the label will be 'Bd' instead of 'B'. In all cases, the next character is 'A' which indicates that the package is upgradeable, but this is not relevant for the exercise.

Pressing 'g' proceeds to consolidate the actions, fixing the broken package by installing the missing package. Information about the package to be installed appears. The 'g' key must be pressed again to confirm this installation. When requested, the root password must be written to proceed with the installation. Once supplied, 'g' has must be pressed again to complete the installation. Does the version of the installed package match the version that was needed to fix the broken package?

Once this is done, if 'b' is pressed in aptitude (check for broken packages), no broken packages should appear.

Tasks to be completed

- Answer the questions:
    1. After executing the command: `dpkg -r gimp-data`, what messages appear? Why is it not recommended to remove the package `gimp-data`?
    2. After running the command: `dpkg --force-all -r gimp-data`. What has happened in this case?
    3. With which character are broken packages marked?
    4. What package and version is listed as required?
    5. Does the package version match the one needed to fix the broken package?

DSI - Laboratory Session 5: Discs Management, Partitions, LVMs, and SFs

28 May 2021

Contents

Goals

The main goals of this session are focused on:

- Understanding the basic concepts related to discs, partitions, logical volumes (LVMs), and file systems (SFs).
- The management of discs, partitions, logical volumes (LVMs), and file systems (SFs).

Previous work

Before attending the lab session, students must:

- Read this document and become familiar with concepts such as discs, partitions, and swap. Most of these concepts have been introduced in the lectures.

- Answer the questions made in the different parts of the document.

Work to be uploaded

- Results obtained in the resolution of the previous exercises.

Disc management

In UNIX systems the computer's devices are usually accessible using special files in the '/dev/' directory; these special files have an associated type (block or character file) and a pair of device numbers (known as the *major* and *minor* device number) that the system uses to know which device driver to use when someone accesses them.

Normally these devices are of the character type, which means that the reading and writing processes are performed by accessing only one octet at a time without *buffering*. However, as discs are block devices, since in their case it makes sense to access specific positions within the device and the driver can use *buffering*.

In Linux, discs have names like '/dev/sda', '/dev/sdb'...; usually the prefix '/dev/sd' is used for SCSI and SATA (*Serial ATA*) discs; while for IDE discs (ATA or *Parallel ATA* discs) '/dev/hd' is used. These letters enable distinguishing between different devices, depending on the order of detection. For example, '/dev/sda' is the first disc, '/dev/sdb' is the second disc and so on.

It is usual to divide discs into *partitions* which are then accessible with the disc name, followed by a number (e.g. the first partition of the disc '/dev/sda' will be the device '/dev/sda1').

Disc partitioning

The partition tables of the hard discs can be managed using special programs ('`fdisk`, `cfdisk`, ...'). These programs know how partition tables should be organized depending on the boot system (*firmware*) used by the computer; thus, PCs with BIOS (*basic input/output system*) use a different partition table format than PCs with UEFI (*unified extensible firmware interface*) or systems with *open firmware* (also known as *OpenBoot*).

Although partition tables have traditionally used the format 'MBR' (*master boot record*), also known as 'DOS' format (introduced by the MS-DOS operating system), currently there is a tendency to a greater use of format 'GPT' (*GUID partition table*), which is more flexible, since it allows more partitions. Traditionally systems with BIOS used 'MBR' and systems with UEFI used the new one 'GPT'. Today's main operating systems can work with 'GPT', even if the system uses the older BIOS.

Previous questions 1

a. Is the device '/dev/sdj' a disc or a partition? Why?
b. Which of the following programs are used to edit the partition table of a disc: '`dd`', '`fdisk`', '`ls`', '`mkdir`', '`cfdisk`'?

Previous exercise 1: adding new discs to *exta*

During the session, the work will be performed in the *exta* machine, of NETinVM. In the first part of the session the work will be done with the 20 GiB 'sda' disc, which will be divided into two classical partitions. The first one will be used to set up a file system and the other one will be configured as 'swap' space. In the second part of the session, the work will be performed with eight discs of 1 GiB, to create physical volumes (PVs), arranged in groups of volumes (VGs) and to build logical volumes (LVs) on them.

A script that creates the nine discs and connects them to *exta* has been prepared. Once NETinVM is started, all the machines should be stopped by using the *Shutdown all* option in the *KVM machines* window of the *base* machine. Once this is done, with the script 'prepare-disks.sh' and the using the account 'user1', it is possible to add additional discs to the specified machine. If these discs are added to the *exta* machine the command is:

```
./prepare-disks.sh exta
```

Similarly, it will be possible to remove all the discs added to *exta* with the following script ('remove-disks.sh'). In this case, it would be possible to remove the disc with (do not execute the command!):

```
./remove-disks.sh exta
```

Since only the *exta* KVM machine will be used, the following command is enough to start it up:

```
netinvm_run exta
```

However, it may be more convenient to set the *Run my machines* script to start only this machine and then use the *Run my machines* icon in the *KVM machines* window of the *base* machine.

Hint

If the *exta* console window is closed by mistake, it is possible to retrieve it by running in *base*, as 'user1', the following command:

```
netinvm_console exta [Console number]
```

Notice that the console number will be 1, 2 or 3.

Once *exta* has started, it is possible to check that the new discs have been detected by the operating system. Thus, the command '`lsblk`' should include the nine new discs, from '/dev/sda' to '/dev/sdi'. It should be noted that '/dev/sda' is a 20 GiB disc, as opposed to '/dev/sd[b-i]', which are 1 GiB discs.

Create two primary partitions ('/dev/sda1' and '/dev/sda2') on that disc using the program `cfdisk`: the partition table will be of type 'GPT', the first partition will be of type *Linux files system* and will have a size of 18 GiB. The second partition will occupy the rest of the disc and will be of type *Linux SWAP*.

Warning

- The command '`cfdisk`', which must be run as 'root', selects the first disc on the system by default. As in this case, when the 'sda' disc is going to be configured, it is not necessary to specify anything else. If the partition table of a different disc must be edited, it must be explicitly indicated as an argument. For example, for editing the disc '/dev/sdb' the command would be:'`cfdisk /dev/sdb`'.
- The program '`cfdisk`' does not ask if the user wants to save the changes. Therefore, if the 'Quit' option is selected, without having previously saved the changes, ('Write' option), the changes made will be lost.

Tasks to be completed

1. Add the new discs to NETinVM (*base* machine).

2. Check that the discs have been detected.
3. Configure the '/dev/sda' partitions as it was indicated in the exercise.
4. Verify that the command '`fdisk -l /dev/sda`' lists the partitions and they have the requested features.

To be uploaded

- The output of the '`lsblk`' command.
- The output of the '`fdisk -l /dev/sda`' command.

Partitions and file systems

File systems can be created in partitions (formatting the partitions) using the command `mkfs`:

```
mkfs -t file_system /dev/device
```

where '`-t`' indicates the file system type (for example, *ext3*, *ext4*, *xfs* or *btrfs*) and '/dev/device' indicates the special device file that corresponds to the partition to be formatted (for example, '/dev/sda1').

Once the file system has been created on the partition, it must be mounted in order to enable access. The partition can be mounted manually using the '`mount`' command. For instance, for mounting a partition with the default options:

```
mount /dev/sda3 /home
```

All the files and directories that can be seen in the path '/home' will be files that will be stored in the third partition of the disc '/dev/sda'.

If different options are needed with the '`mount`' command, they must be explicitly included (these options can be checked in the '`mount`' manual page).

A partition can be mounted automatically by just adding a new line to the file '/etc/fstab' that includes the information about the device, the directory, and additional options (see '`man fstab`' for an explanation of the meaning of the fields). The partition can then be mounted using the '`mount`' command specifying only the mount point (e.g. '`mount /home`') or it can also be mounted by executing the command for all the file systems configured for automatic mounting ('`mount -a`'). When the machine is rebooted the partitions that are marked in the file '/etc/fstab' as auto-mount will be mounted automatically.

Note

The identifier (UUID) of a partition can be obtained by using the command '`blkid`' as 'root'. For example:

```
root@base:~# blkid /dev/sda1
/dev/sda1: UUID="ec6f8b83-9499-427c-9633-977ff0f26655" TYPE="ext4"
PARTUUID="5578845b-b6e3-4477-b358-7278d03f6584"
root@base:~#
```

Information about the mounted devices can be obtained with the output of the '`mount`' command without arguments or, alternatively, by checking the contents of the file '/etc/mtab'.

A partition can be unmounted with the '`umount`' command followed by the device name or mount point.

Previous question 2

In which order does it make more sense to execute the following commands: '`mkfs`', '`mkdir`', '`umount`', '`cfdisk`', '`mount`'? Justify the answer.

Previous exercise 2: creating a file system

A file system on the device '/dev/sda1' must be created and it should be mounted in a directory with the name '/mnt/adic'. Since the mount point is a directory to which the mounted file system is attached, the directory '/mnt/adic' must be created before executing the '`mount`' command.

Tasks to be completed

1. Create a file system with the *ext4* format in the '/dev/sda1' device.
2. Create the mount point '/mnt/adic'.
3. Mount the newly created file system on that mount point.
4. Check that '/mnt/adic' has been created using '`mount`' (without arguments). If there is too much information the '`grep`' command can be used to filter. For example, the output of the command '`mount | grep adic`' should show the required information.
5. Unmount it and check that it is unmounted.
6. Add '/dev/sda1' to the '/etc/fstab' file using its UUID (suggestion: use '`blkid`' to obtain it).
7. Mount it using '`mount -a`' and check that it is mounted.

To be uploaded

- The output of the '`mount | grep adic`' command.
- The content of the '/etc/fstab' file with a brief explanation of it.

Swap memory in partitions and files

In addition to using the partitions to hold files, they can be used to increase the available memory on the system by using them as swap space (virtual memory).

The partition will be formatted using the command '`mkswap`' and activated using the command '`swapon`' followed by the partition name.

The command '`swapoff`' followed by the name of the device can be used to deactivate a swap partition.

The automatic activation of swap partitions can be configured by adding the corresponding entries in the file '/etc/fstab':

```
/dev/sda2    none    swap    sw     0      0
```

The command '`swapon -a`' can be used to start using them and '`swapoff -a`' to stop using them. (When the system boots, the command '`swapon -a`' is automatically executed).

Additionally, files can also be used as swap memory with the only restriction being that the files used must not have any gap.

An example of how to create a file with name '/var/local/swapfile' and 512MiB of free space is below (the creation of the file will take a few seconds):

```
dd if=/dev/zero of=/var/local/swapfile bs=1M count=512
```

Once the file is created, it can be used in the same way as a swap partition. In this case, the activation command includes the path to the file instead of the name of a swap partition. It must also be formatted with the '`mkswap`' command and then activated/deactivated with '`swapon/swapoff`', as in the case of swap partitions.

The disadvantage of using files instead of partitions is that access is less efficient, but the advantage is that they facilitate the management of swap memory (it is possible to add, delete, or modify swap memory without touching partition tables). However, since these files may end up containing private information from running applications, only the ‘root’ should be able to access them.

Previous exercise 3: adding swap memory to the system

The swap partition '/dev/sda2' must be permanently added to the system. Additionally, it should also be added 1 GiB of space in a swap file.

Tasks to be completed

1. Add the '/dev/sda2' partition as swap space to the system:
   - Prepare it to be used as swap memory using the command '`mkswap`'.
   - Add an entry in the file '/etc/fstab' to be automatically activated at boot up.
   - Activate the use of the system using the command '`swapon`'.
2. Prepare a 1024 MiB file to be added to the system as swap and activate it for use. (A good place to create the file is in the '/var/local' directory. If the directory does not exist, it must be created.)

To be uploaded

- Content of the file '/etc/fstab'.
- The output of the command '`mount | grep '/dev/sd'`'.
- The output of the command '`swapon -s`'.

Physical Volumes, Volume Group and Logical Volumes

Logical volume management (LVM) is a useful technique for implementing logical storage – which provides greater flexibility than physical storage. With LVM, 'logical' partitions can be extended across physical hard discs and resized as needed, unlike traditional partitions. In this way, it is possible to increase the size of a partition. It is also possible to reduce the size of a partition, providing that the space is not in use, without data loss. It is therefore possible to allocate minimal amounts of space for each logical volume and leave part of the disc unallocated. Later, when the partitions start to fill up, they can be expanded as needed.

A physical disc is divided into one or more physical volumes (PV). Each PV is made up of fixed-size physical extents (PE). A volume group (VG) is created by grouping PVs. A VG is where a logical volume (LV) is created, which is formed by logical extents (LEs). Later, it is possible to define a file system (FS) on the LV and mount it on the system for normal use.

LVs provide more flexibility than traditional partitions. For example, it is possible:

- Any number of discs can be used as a single large disc, spreading logical volumes over several discs.
- Small logical volumes can be created and their size can be changed 'dynamically' as they fill up.
- Logical volumes can be resized regardless of their order on the disc. It does not depend on the position of the LV inside the VG, there is no need to ensure surrounding available space.
- Resize/create/delete/substitute logical volumes or physical discs in a running system.

The power of using LVs will be illustrated by showing the whole process, starting with the creation of the PVs, VGs, LVs and concluding with the implementation of an FS in the LV. In addition, the physical replacement of a disc will be simulated, with the system running and avoiding data loss. All these tasks are going to be done on NETinVM.

Previous question 3

Which of the following statements is more accurate?

a. LVs are stored on discs, which can hold one or more VGs.
b. PVs are stored on LVs, which can hold one or more VGs.
c. VGs can hold several LVs and they are stored in one or more PVs.

Work in the laboratory

Working with physical volumes, volume groups and logical volumes

Exercise 1: preparing NETinVM to create PVs, VGs, and LVs

This exercise is a type of tutorial that the student must follow. Answers to the questions must be copied to a text file, which will be the result of this exercise.

In the first exercise of the previous work, the discs required for this section were created. Thus, once logged as superuser in *exta*, the block devices must be listed with the command '`lsblk`'. Therefore, it is possible to check that the discs 'sd[b-i]' of 1 GiB are available.

It is possible to check the performance of each disc (in the example 'sdb') with the command:

```
hdparm -Tt /dev/sdb
```

It is necessary to install additional packages to complete the work of this session, therefore, the repositories must be updated, and two packages must be installed, with the commands:

```
apt update
apt install lvm2
apt install thin-provisioning-tools
```

Warning

If the '`apt update`' command does not work, the internet connection of the exta machine must be checked (the command '`ping www.uv.es`' can be used to check it). If the internet connection does not work, it can be due to a clock malfunction in the *exta* machine. This can happen if the machine has been suspended. It is possible to correct this by setting the hardware time properly with the command '`hwclock -s`', or alternatively, the *exta* machine can be halted with the command '`shutdown -h now`' and then restart it again with the command '`netinvm_run exta`' on the *base* machine.

Tasks to be completed

- Check that eight discs of 1 GiB have been added to the *exta* machine.
- Find out their capacity.
- Evaluate the performance of the discs with the '`hdparm`'.
- Install the additional packages indicated.

To be uploaded

- Result of the command '`lsblk`' and checking that the discs have been rightly created.
- Capacity of the discs and explanation of how these numbers have been found.
- Read speed using the disc buffer, measured with '`hdparm`' (choose a 1 GiB disc to test).

Exercise 2: creating PVs and VGs

The command '`lvmdiskscan`' is useful to obtain the list of devices that can be used to create PVs. The list shows if they are already formatted for this purpose. The output of the command must be copied to a text file.

The commands '`pvscan`', '`pvdisplay`' and '`pvs`' provide information about the PVs of the system. Initially there should not be any (this can be checked with any of these commands).

A PV can be created with the command '`pvcreate`'. For example, to create a PV in the 'sdb' disc, the command is:

```
pvcreate /dev/sdb
```

The commands '`vgscan`', '`vgdisplay`' and '`vgs`' provide information about the VGs that are present in the system. Initially there is no VG and it can checked with any of the previous commands.

The creation of VGs is performed with '`vgcreate`'. For example, to create a volume group called 'vg1', with three PVs ('sdb', 'sdc' and 'sdd'), use:

```
vgcreate vg1 /dev/sd[b-d]
```

There are more commands that enable expanding, reducing, mixing, renaming and deleting VGs. For example, it is possible to add PVs with '`vgextend`'. Thus, the PVs 'sde' and 'sdf' can be added to 'vg1' with the command:

```
vgextend vg1 /dev/sd[ef]
```

Similarly, '`vgreduce`' can be used to reduce a VG. In this way, the PVs 'sdc' and 'sdd' can be removed from the VG 'vg1' with the command::

```
vgreduce vg1 /dev/sd[cd]
```

The command '`vgmerge`' merges two existing VGs. In the following example 'vg2' is merged into 'vg1':

```
vgmerge vg1 vg2
```

The command 'vgsplit' splits one VG into two VGs or moves one or more PVs from the source VG into a destination VG. In the following example, the PVs '/dev/sd[gh]' are moved from 'vg1' to 'vg2':

```
vgsplit vg1 vg2 /dev/sd[gh]
```

In the example 'vg1' is the source VG and 'vg2' is the destination VG. If the destination VG does not exist, it is created.

The commands 'vgrename' and 'vgremove' have the utility that their name indicates, and it is left to the student to check the manual if these commands are needed.

Once the VGs have been created, the PVs commands 'pvscan', 'pvdisplay' and 'pvs' provide information about the VG in which the PVs have been included. It is also possible to obtain the PVs included in a particular VG with the command:

```
pvs --select vg_name=vg1
```

Tasks to be completed

- Eight PVs must be created with only one command using 'pvcreate'. Every PV consists of one 1 GiB disc. Show the result with 'pvscan'.
- Create two VG named 'vg1' and 'vg2' assigning half of the available PVs to each VG. Thus, 'vg1' will have the discs from 'sdb' to 'sde' and 'vg2' the rest. Recover the information of the VGs created using the 'vgs' and 'pvs' commands.
- Remove the discs 'sdd' and 'sde' from 'vg1'. Check the result with 'vgs' and 'pvs'.
- Add the discs 'sdd' and 'sde' to 'vg2'. Show the result with 'vgs' and 'pvs'.
- Merge both VGs in 'vg1'. Obtain the information of the resulting VG with 'vgs' and 'pvs'.
- Split 'vg1' to recover the two original VGs, 'vg1' and 'vg2', each one with half of the PVs (as they were initially). Check the result with 'vgs' and 'pvs'.
- Use the command 'vgdisplay' to obtain information about the capacity of the PEs, the number of PEs that are assigned to each VG and the number of PEs that are free.

To be uploaded

- Commands used in the previous tasks.
- Output of the commands 'vgs' and 'pvs', showing the capacity of the resulting VGs and the discs assigned after each modification.
- Information obtained with the command 'vgdisplay' about the capacity of the PE, number of PEs assigned to each VG and free/occupied status.

Exercise 3: creating LVs from VGs

Once there is a VG, it is possible to create an LV from it and finally to format the LV to have an FS. The flexibility of the LVs will then be tested, which can be resized on-line.

Create an LV from an existing VG using the command 'lvcreate'. Jointly with the command, it is necessary to include: the '-L' option to specify the size of this LV, the '-n' option to set the name of the LV, and finally the VG to create the LV. In this way, to create a 1GiB LV, called 'lv1', from the VG 'vg1', the command is:

```
lvcreate -L 1G -n lv1 vg1
```

It is also possible to create an LV with the option '-l', indicating the percentage of free space assigned to that LV. To create an LV called 'lv1', with 50% of the free space of 'vg1', the command is:

```
lvcreate -l 50%FREE -n lv1 vg1
```

Similarly to the commands for PVs and VGs, it is possible to discover with the command '`lvs`' the number of LVs (including their features). With '`lvscan`' it is possible to search for the existing LVs on all the discs. Detailed information about the LVs present in the system is obtained with '`lvdisplay`'.

An LV can be removed using '`lvremove`' jointly with its name, including VG/LV:

```
lvremove vg1/lv1
```

The capacity of an LV can be increased with '`lvextend`' and the '-L' option to set the size (default in MiB). Thus, the capacity can be increased by half a GiB with:

```
lvextend -L +512 vg1/lv1
```

An increase of 1GiB can be achieved with:

```
lvextend -L +1G vg1/lv1
```

If the '+' sign is not present, the size is taken as absolute value, i.e., the resulting size is fixed to that value (not adding the value). If the new absolute value is less than the current size, it is not changed.

It is also possible to directly add a PV:

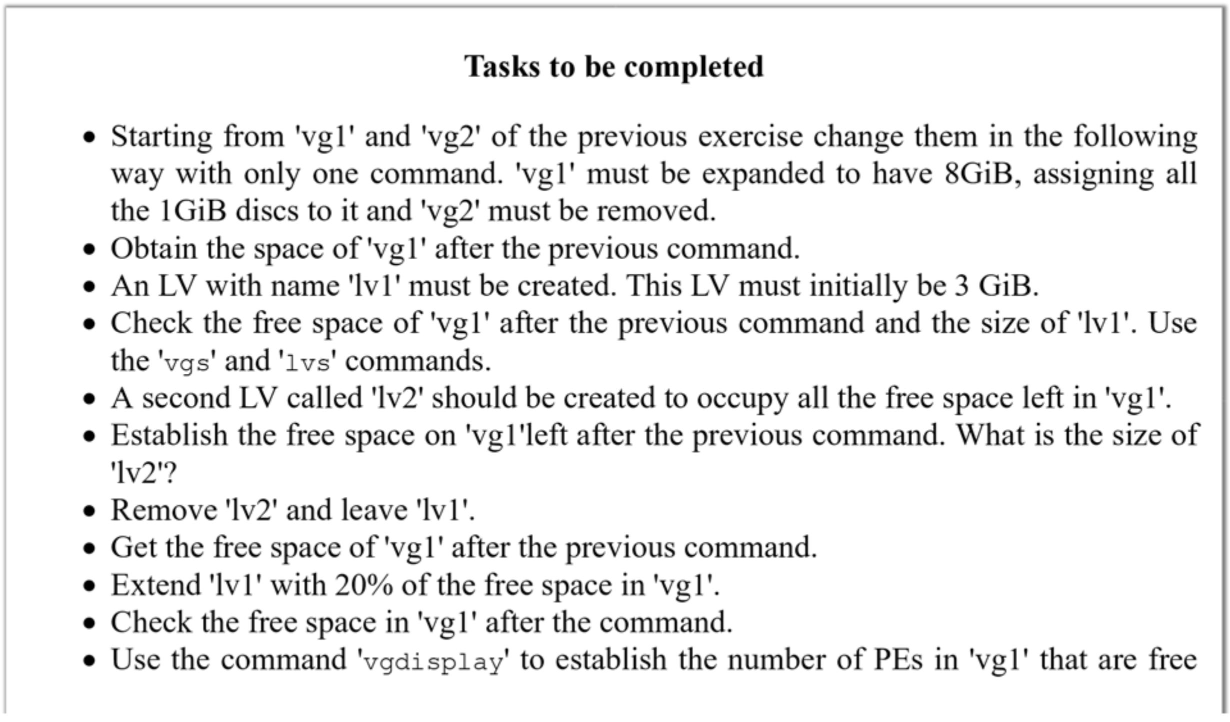
```
lvextend vg1/lv1 /dev/sdc
```

With the '-l' option the command enables specifying the number of PEs. If the '+' sign is included this value is added to the existing values. If not, it is taken as an absolute value. In addition, it can be added [%{VG|LV|PVS|FREE|ORIGIN}] to indicate the percentage to set or add (with '+'). The meaning of each option is:

- %VG is the percentage of the total space in the VG
- %LV is the percentage of space in the LV
- %PVS is the percentage of free space remaining in the PVS
- %FREE is the percentage of free space remaining in the VG
- %ORIGIN is the percentage of the total space of the original LV (if it is a snapshot of another LV)

For example, the 50% of the free space of the VG can be added to the LV:

```
lvextend -l +50%FREE vg1/lv1
```



Tasks to be completed

- Starting from 'vg1' and 'vg2' of the previous exercise change them in the following way with only one command. 'vg1' must be expanded to have 8GiB, assigning all the 1GiB discs to it and 'vg2' must be removed.
- Obtain the space of 'vg1' after the previous command.
- An LV with name 'lv1' must be created. This LV must initially be 3 GiB.
- Check the free space of 'vg1' after the previous command and the size of 'lv1'. Use the '`vgs`' and '`lvs`' commands.
- A second LV called 'lv2' should be created to occupy all the free space left in 'vg1'.
- Establish the free space on 'vg1'left after the previous command. What is the size of 'lv2'?
- Remove 'lv2' and leave 'lv1'.
- Get the free space of 'vg1' after the previous command.
- Extend 'lv1' with 20% of the free space in 'vg1'.
- Check the free space in 'vg1' after the command.
- Use the command '`vgdisplay`' to establish the number of PEs in 'vg1' that are free

and the number that are occupied.

To be uploaded

- Commands used to modify 'vg1', 'lv1' and 'lv2' and their results.
- Sizes of 'vg1', 'lv1' and 'lv2' after each command.
- Number of PEs assigned to 'vg1' and their status, i.e., free or occupied.

Exercise 4: creating a file system (FS) on an LV

A common use of LVs is to create a file system. For example, it is possible to format 'lv1' with an *ext4* file system as would be done in a classic partition:

```
mkfs.ext4 /dev/vg1/lv1
```

With '`lvs`' it is possible to check that the attributes of the LV do not change: 'w' (read/write mode), 'i' (inherited space allocation policy) and 'a' (active). If the created FS is mounted (with the '`mount`' command), after again running '`lvs`' the 'o' (open) attribute appears, thus indicating that it is in use. In this case, it is not possible to delete it with the command '`lvremove`'.

For most FS, the size of the LV can be increased without unmounting the FS. However, to decrease the size of the LV there must be free space not used by the FS, and it must be unmounted. This does not have to be done beforehand, since when '`lvreduce`' is run, it is unmounted and then mounted again. If there are processes using the file system, then it will fail when trying to unmount. For example, the LV 'lv1' can be decreased by 1 GiB and the FS on it resized with the command:

```
lvreduce -L -1G --resizefs vg1/lv1
```

If the file system is mounted, the system will ask if it must be unmounted to reduce the size.

Similarly, '`lvextend`' enables increasing the size of the LV. For example, for extending the LV to occupy 100% of the free space in the VG, use the command:

```
lvextend -l 100%FREE --resizefs vg1/lv1
```

In this case, the file system can remain mounted during the process.

Tasks to be completed

- Create a file system in 'lv1'.
- Create the mount point '/mnt/lv1'.
- Mount 'lv1' on the mount point.
- Check the available storage space in '/mnt/lv1'.
- Part of the free space in '/mnt/lv1' will then be occupied copying the files from the folder '/usr/bin/':

  ```
  cp -r /usr/bin/ /mnt/lv1/bin
  ```

- Check now the space that is used and available in '/mnt/lv1'.
- The space should be reorganized and the size of the 'lv1' must now be decreased to half its capacity without loss of data.

- After resizing the LV, check the available space in '/mnt/lv1'.
- Finally, extend the size of the LV 'lv1' so that it occupies all the space in 'vg1'.
- Check again the space available in '/mnt/lv1'.

To be uploaded

- Sequence of commands used and their output.

Exercise 5: replacing a disc from a VG/LV

A disc failure is a situation that every system administrator will eventually have to face. Discs usually start to show symptoms of fatigue when the components wear out and they perform error correction techniques more often than is average. When this happens, it is worthwhile replacing the disc with a new one as soon as possible.

This exercise is going to start with an LV with an active (mounted) file system. One of the VG discs is going to be replaced and the LV will be created without interrupting service, that is, without unmounting the file system.

The procedure is conceptually simple, although it can be time consuming, especially if the disc to be replaced is large and has many occupied PEs. The process has the following stages:

- Add a new disc to the VG. This step is only necessary if the VG does not have enough free space to copy the information of the disc that is starting to fail.
- Move the PEs from the damaged disc to the newly added disc. To do this, use the command '`pvmove`'. For example, the PEs from the PV '/dev/sde' must be moved to the PV '/dev/sdf' with the command:

  ```
  pvmove /dev/sde /dev/sdf
  ```

- Remove the damaged disc. Use the command '`vgreduce`', which was explained previously.

Tasks to be completed

- It is intended to start without any VG or LV created, so the first step is to remove 'lv1' and 'vg1'. (If 'lv1' is mounted, it must be dismounted first with '`umount /mnt/lv1`').
- Create 'vg1' with 2 PVs: 'sdb' and 'sdc'.
- Create 'lv1' from 'vg1', using all the free space.
- Format LV 'lv1' with *ext4*.
- Create the mount point '/mnt/lv1'.
- Mount 'lv1' in this mount point.
- Check the space occupied by 'lv1' (suggestion: use the command '`df -hT /mnt/lv1`').
- Now part of the free space of 'lv1' is used to copy the folder '/boot':

  ```
  cp -a /boot /mnt/lv1/boot
  ```

- Check the space occupied in 'lv1'.
- Unfortunately, the PV 'sdc' is starting to fail and it seems a good idea to replace it, but the system cannot be stopped and, of course, data loss is not allowed. Therefore, a new PV must first be added to 'vg1': 'sdh'.
- Check the usage of the PVs using the '`pvs`' command.
- The data from the disc to be replaced ('sdc') must be moved to the new disc ('sdh').
- Check again the usage of the PVs with the command '`pvs`'. What has changed?
- The defective disc can now be removed ('sdc').
- Get the space that is free and occupied in '/mnt/lv1', the PVs that form 'vg1' and check that the information is kept in '/mnt/lv1'. Go through '/mnt/lv1/boot' and check that the file structure has been preserved.

To be uploaded

- Sequence of commands used and their output.

Snapshots of LVs

A very interesting feature of LVMs is that they enable making snapshots of LVs. These can be used as a backup, which enables returning an LV to its initial state (when the snapshot was made) if the new changes introduced are not satisfactory. If the changes are correct, these changes become permanent, and the snapshot can be removed.

It should be noted that if the snapshot runs out of space it would be unusable. Therefore, its state must be controlled, and if it is about to fill up, its space must be expanded, as with any normal LV.

The snapshot is created with the same '`lvcreate`' command already shown, but with the '-s' option, indicating the name of the snapshot and the name of the source LV. For example, a snapshot of 'lv1', called 'lv1s1', with a size equal to 100% of the space of 'lv1' can be created using the command:

```
lvcreate -s --name lv1s1 -l 100%ORIGIN vg1/lv1
```

With the '`lvs`' command it is possible to check the state of both LVs. It can be observed how the attribute 'o' (*origin*) is activated in 'lv1' and how 'lv1s1' is marked as LV-SH ('s', snapshot) with origin in 'lv1'.

It is possible to mount the new LV and see how its content is identical to the source LV at the time of making the snapshot. It is also possible to show how the original LV changes with respect to the moment in which the copy was made. On the other hand, the snapshot 'lv1s1' will occupy more space of the LV when changes are made because it must keep the original content of the files that are modified.

Since the snapshot LV serves as a backup, all the modifications made to the content of 'lv1' can be discarded and its original content can be recovered using the snapshot 'lv1s1'. This is possible with the command:

```
lvconvert --merge vg1/lv1s1
```

This makes the snapshot disappear. If the changes made to the original LV must be kept, simply remove the snapshot with '`lvremove`'.

Exercise 6: creating snapshots as backups

The exercise aims to simulate the scenario in which irreversible changes are going to be made, so it is a good idea to create a backup before applying the changes. In this case, the snapshot of the LV can be used as a mechanism to undo the changes made.

In this exercise, the 'lv1' is going to be used in the state in which it was left at the end of the previous exercise. The 'lv1' has been created on the VG 'vg1' with a capacity of 2GiB and it has no free space.

Tasks to be completed

- Add the disc '/dev/sdd' to 'vg1'.
- Create a snapshot of 'lv1', which will be called 'lv1s1'. It should save changes up to 50% of the size of the original LV.
- Check with '`lvs`' the size of the snapshot created.
- The mounting point '/mnt/lv1s1' must be created and the snapshot must be mounted there.
- Check that the % of use is identical in both 'lv1' and 'lv1s1'.
- In both mount points, the 'boot' directory is already there, copied in the previous exercise to '/mnt/lv1' and automatically copied to '/mnt/lv1s1' after creating the snapshot. Check that both directories are identical. Suggestion: use '`diff -r /mnt/lv1/boot /mnt/lv1s1/boot`' to compare both directories. Also compare the original directory '/boot' with '/mnt/lv1/boot' and with '/mnt/lv1s1/boot'.
- The directory '/mnt/lv1/boot/grub' should now be removed as it is not considered necessary:

  ```
  rm -rf /mnt/lv1/boot/grub
  ```

- Compare again the content of the three directories '/boot' , '/mnt/lv1/boot' and '/mnt/lv1s1/boot' to check the differences between them.
- Unfortunately it has been discovered that someone has mistakenly deleted the directory '/mnt/lv1/boot/grub', and the information must be recovered from the snapshot. To do this, both 'lv1' and 'lv1s1' must be unmounted. Retrieve the information from the snapshot and mount 'lv1' again.
- Compare the directory '/mnt/lv1/boot' with the original directory '/boot' and check that the information has been recovered from the directory '/mnt/lv1/boot/grub'.

To be uploaded

- Sequence of commands used and their output.

Generated on: 2021-05-28 14:43 UTC. Generated by Docutils from reStructuredText source.

DSI - Laboratory Session 6: Redundant Array of Independent Discs

27 May 2021

Contents

Goals

- Understand the basic concepts related to several kinds of RAIDs, in particular their features: size, read speed, and write speed.
- Learn to create and manage most types of RAIDs.

Previous work

Students, before attending the lab session, must:

- Read the file of the lab session and be familiar with the concept of RAID: advantages and disadvantages of each type, capacity, speed, etc. These concepts have been introduced in lectures and problem classes.
- Carry out the tasks marked as 'Previous exercise'.

To be uploaded

- Required details of installation of packages and creation of discs in exta.
- Answer to the questions marked as 'Previous question ...'.

RAID concept

The term RAID was born in a work published in 1988 [1]. Currently the term RAID is an acronym for redundant array of independent discs. A RAID is a way of organising information on multiple hard drives with the intention of obtaining protection against disc failures, and/or faster access to information.

A RAID for the operating system appears to be a single logical hard drive. Data is written in chunks to multiple discs simultaneously. There are diverse RAID arrangements that offer greater fault tolerance and higher levels of performance than a single hard drive or a group of independent hard drives. Each RAID level offers a specific combination of fault tolerance, performance, and cost, designed to meet different storage needs. There are RAID levels that focus on fault tolerance, without increasing speed. Other levels are valid to increase access speed, but without protection against failures. Finally, there is another group of RAIDs in which two objectives are combined, protection against disc failures and improved performance.

Originally, more RAID levels were defined than those that will be seen in this lab session. Some theoretical RAID configurations are not implemented nowadays in a practical way, being outmatched by other RAID levels in terms of performance/cost. Other levels that originally had great acceptance, have become obsolete due to cheaper discs and increased disc capacity or speed. Cost initially discouraged certain RAID configurations, which over time have become feasible and therefore more popular. Many real configurations are mixes of several RAID levels and tailored to specific problems. The most popular RAID configurations are described below

[1] D. Patterson, et al. 'A Case for Redundant Arrays of Inexpensive Discs (RAID).' SIGMOD International Conference on Management of Data, Chicago, IL, USA, 1-3 June 1988. SIGMOD RECORD (Sept. 1988) vol.17, no.3, p. 109-16.

RAID 0

This RAID level is called **disc striping**. It consists of dividing the information of the original files into pieces of data that are written or read in parallel on the discs. The more discs there are, the more fragmented the information when writing and/or reading from N discs at the same time. This type of RAID offers the highest speed performance: with `N` discs with `X` speed, there is a reading and writing speed of `N·X`. (Assuming that reading and writing speed is the same). The weak point of this RAID configuration is that it has no fault tolerance.

The storage space is fully used. Thus, with `N` discs of size `S` the storage capacity is `N·S`.

The minimum number of discs for RAID 0 is two.

RAID 1

This case is the opposite of RAID 0. Instead of *striping*, mirror copies of the original disc are made: **mirroring**. The advantage of this method is that there is fault tolerance, but on the downside, no speed improvement is achieved. In this way, a RAID 1 with `N` discs of speed `X` has a read speed of `N·X` since it is possible to make read requests of different fragments to the different discs in parallel. However, it is not possible to speed up writing, as the same chunk must be written to the `N` discs at the same time, so the writing speed will be `X`.

Similarly, storage space is used in backup copies and therefore having more discs does not mean increased space. With `N` discs of size `S` there is a space of `S`, as with a single disc.

The minimum number of discs to build a RAID 1 is two. With two discs the RAID can tolerate the failure of one of them. With more discs, a greater redundancy is achieved and so the failure of more discs is tolerated.

RAID 10

Also known as RAID 1+0. This consists in combining both strategies: *striping* and *mirroring*. This makes it possible to increase speed, while providing fault tolerance through redundancy. A RAID 10 with `N` discs of speed `X` would have a reading speed of `N·X`, since it is possible to read the information by fragments of all discs at the same time, even from those that are redundant copies. On the contrary, with writes it is not possible to take advantage of the access to the N discs at the same time, since half of the discs are used to *mirror* the other half. Therefore, the writing speed is `N·X/2`.

The visible storage space will therefore be that of half the discs. With `N` discs of size `S` there is a space of `N·S/2`.

The minimum number of discs to build a RAID 10 is four. With four discs the failure of one disc is tolerated.

RAID 5

This offers *striping* with distributed parity. Instead of having mirrored discs or discs fully dedicated to parity, the parity information is distributed in an interlaced manner across the N discs of the RAID, so that it is possible to rebuild the information in real time if any disc in the RAID fails.

With `N` discs with `X` speed, there will be a reading speed of `N·X`. However, to do a write it is necessary to read the data, read the parity, then write the data and the parity, so writing speed is penalised by four, yielding `N·X/4`.

The storage space is also reduced from the original N discs due to redundant information. With `N` discs of size `S` the available storage space is `(N-1)·S`.

The minimum number of discs to build a RAID 5 is three. In this type of RAID, the failure of one disc is tolerated.

RAID 6

This level of RAID is similar to RAID 5, but offering a double parity system, so that there is more redundancy and therefore fault tolerance.

With `N` discs with `X` speed, the reading speed is `N·X`. However, a different result is obtained for writing. To make a write, it is necessary to read the data, read the two parity fragments, then write the data and the two parity fragments, and so the writing speed is penalised by six and yields `N·X/6`.

The storage space is also reduced compared to the original N discs, but more markedly due to the double parity scheme. With `N` discs of size `S` the available storage space is `(N-2)·S`.

In this case, the minimum number of discs to build a RAID 6 is four. The most interesting part of this type of RAID is that it tolerates the failure of two discs.

Summary of benefits

The following comparison table summarises the capabilities of the various types of RAID described in this lab. It is assumed that `N` discs are equal, with speed `X` and with capacity `S`. The **RIOPS** column shows the read speed: *Read Input/output Operations Per Second*. The **WIOPS** column shows the write speed: *Write*

Input/output Operations Per Second. The **Capacity** column shows the available storage space on the RAID. The **Minimum** column indicates the minimum number of discs required to build a RAID of the indicated type, and the **Failures** column shows the number of discs that are tolerated to fail simultaneously.

RAID	RIOPS	WIOPS	Capacity	Minimum	Failures
0	N·X	N·X	N·S	2	0
1	N·X	X	S	2	(N-1)
10	N·X	N·X/2	N·S/2	4	Between 1 and N / 2-1 [*]
5	N·X	N·X/4	(N-1)·S	3	1
6	N·X	N·X/6	(N-2)·S	4	2

[*] Depends on which specific discs in the RAID configuration fail.

Previous question 1

Initially, there are four equal discs of 500 GB. The goal is to build a RAID with a capacity of at least 1.5 TB, offering fault tolerance of at least one disc and with the highest possible speed, for both reading and writing. What type of RAID would be best suited to these requirements? The answer must be reasoned, and the advantages and disadvantages of the selected RAID must be explained.

Previous question 2

The cost of 1TB magnetic discs is considered a critical parameter in this example, so the number of discs should be minimized in this case. The goal is to build a RAID with a capacity of at least 3 TB, tolerating at least one disc failure, and at least doubling the write speed of an individual disc. What would be the type of RAID that with fewest discs would meet the specifications? Justify both choosing a RAID type and discarding the rest.

Previous exercise 1: preparing the discs in exta of NETinVM

During the session, the exta machine, inside NETinVM, is going to be used.

In this session eight equal discs of 300 MB are going to be used. For this reason, if the discs of the previous version are still in place, they must be removed with the script `remove-discs.sh` that was provided with that laboratory assignment.

The eight discs in this session are different from those in the previous session. To be able to clearly observe the speed gain in the various types of RAID, the speed of the discs will be limited to 200 iops. A script has been prepared that creates the eight 300 MB discs with limited speed and connects them to exta. Once NETinVM has been started, all machines must be stopped by pressing the *Shutdown all* button in the *KVM machines* desktop widget. Once this is done, with the script 'prepare-discs.sh', **as user1 in 'base'**, the additional discs can be added to any KVM machine. For example, to add the discs to *exta*:

```
./prepare-discs.sh exta
```

Similarly, it would be possible to remove all the discs added to exta with the script 'remove-discs.sh' (although it should not be necessary):

```
./remove-discs.sh exta
```

As only the KVM machine *exta* will be used, it is enough to execute the following command:

```
netinvm_run exta
```

However, it may be more convenient to configure the 'Run my machines' script to only start this machine and then use the 'Run my machines' icon.

Hint

If by mistake the *exta* console windows are closed, it is possible to recover them by executing in *base*, as 'user1', the following command:

```
netinvm_console exta [Console number]
```

Note that the console number will be 1, 2 or 3.

Once *exta* is booted, it can be verified that the new discs have been detected by the operating system. To do this, the command `lsblk` should include the eight new discs, from '/dev/sda' to '/dev/sdh'. It should be noted that the eight discs are equal and 300 MB.

Tasks to be completed

1. Add the new discs to NETinVM (base machine).
2. Check that the discs have been detected.

To be uploaded

- The output of the lsblk command execution in exta.

The mdadm command

The exta machine does not have the mdadm command installed (useful for creating and managing RAIDs in Linux environments). This tool originally developed by Derek Vadalam was published on the internet in 2002 [2]. The article is not now available at the original address, since the same author published a book in 2009 with the extended content of that article [3]. The documentation provided by the Linux manual, once the command is installed, is enough for the development of the session.

Previous exercise 2: installing mdadm

To install the mdadm command in exta, several tasks must be performed:

- Update the system repository sources using the apt command, update option:

```
# apt update
```

- Install the mdadm command:

```
# apt install mdadm
```

Tasks to be completed

1. Update the system repository sources.
2. Install the mdadm command.

To be uploaded

- The result of the execution of both commands. Just copy the text.

Introduction to mdadm

mdadm has seven main modes of operation. Usually only *Create*, *Assemble* and *Monitor* modes are used. The rest of the modes are useful to fix or modify the RAID in some way. In general, mdadm commands have the following format:

```
# mdadm [mode] <raiddevice> [options] <component discs>
```

To run mdadm, administrator privileges are needed, and so it is assumed that mdadm is executed as root.

The modes can be specified with options. Sometimes, when a certain option is set, a certain mode is assumed, without the need to explicitly specify it. If a certain mode is needed, the following options can be used:

--create or equivalently **-C**

Creates a new RAID, with the specified name, specifying the level after the `--level` option, and the devices specified with the `--raid-devices` option. For example, to create a RAID 5, called `myraid5`, with three active discs forming the RAID: sda, sdb and sdc, leaving sdd as *hot spare*:

```
# mdadm --create /dev/md/myraid5 --level=5 --raid-devices=3 --spare-
devices=1 /dev/sd[a-d]
```

--assemble or equivalently **--A**

Starts a previously defined RAID. The array may have been previously defined from the command line or it may be specified in /etc/mdadm.conf. This is usually done in start-up scripts after a system reboot. For example, the following command will boot all RAIDs defined in mdadm.conf:

```
# mdadm --assemble --scan
```

--follow or **--monitor** or equivalently **--F**

It monitors one or more RAIDs and acts on any change of state, being able to configure mdadm as a daemon to send alerts and/or execute commands when a disc fails. By default, the alert messages are sent to the root user of the system. The following command monitors events every 60 seconds in the RAID myraid5, sending an email to the system administrator:

```
# mdadm --monitor --mail=sysadmin --delay=60 /dev/md/myraid5
```

--build or equivalent **--B**

Build a RAID without superblocks. For these types of RAIDs, mdadm cannot differentiate between the initial build and later assembly. Moreover, it is not possible to check that the discs are suitable. This mode should not be used unless the command is fully understood.

--grow or equivalently **--g**

Useful to grow, shrink or reshape a RAID in some way. An example would be, given the RAID level 5 example, myraid5, with 3 active discs and a spare disc, it would be possible to increase the number of discs in the RAID using the spare disc, for this it would be possible to do:

```
# mdadm --grow --raid-devices=4 /dev/md/myraid5
```

To add another spare disc, in this case sde, it is possible simply add it with the --add-spare option. This causes it to enter manage mode, which does not have an option as such to be selected. The command would be:

```
# mdadm /dev/md/myraid5 --add-spare /dev/sde
```

--misc or equivalent **--M**

This includes all options that are not included in the rest of commands. For example, if a disc is included as the first parameter, or if the first option is --add, --re-add, --add-spare, --fail, --remove, or --replace, then it is assumed to be in **manage** mode. Any other option will cause misc mode to be assumed.

Useful examples within this mode would be the following to see the status of the RAID or one of its components:

```
# mdadm --misc --detail /dev/md/myraid5
# mdadm --examine /dev/sdd
```

An interesting option is to mark a disc as faulty. This means that if the RAID implements redundancy, the system is properly reconfigured. It must be noted that in the case of a RAID with large discs this time can be a long time. In the case of the session, with 300 MB discs, these times will not exceed one minute. In the example below, the sda disc is marked as faulty:

```
# mdadm /dev/md/myraid5 --fail /dev/sda
```

In the lab session students will be guided through the sequence of actions to create various types of RAID, make them grow, remove them completely, etc.

[2] Derek Vadala. 'mdadm: A New Tool for Linux Software RAID Management'. 2002. Originally published in http://www.linuxdevcenter.com/pub/a/linux/2002/12/05/RAID.html.

[3] Derek Vadala. 'Managing RAID on Linux. Fast, Scalable, Reliable Data Storage'. Publisher: O'Reilly Media. March 2009.

The fio command

The hdparm command is not the best choice for measuring the read and write speed of a RAID. It operates at a low level, and the results are often inconstant for the same measurement, and there are not many options to adjust exactly what to measure and how.

The fio command has been created specifically to allow benchmarking of I/O disc workloads. It has many options and it is possible to make measurements taking into account different file sizes, access types, cache use, etc. During the session, the full fio command will be given, with all of its options set for reading in one case and writing in another, so that only the name of the RAID on which the measurement is going to be made will have to be changed. As it is a long command to write, intensive use of COPY and PASTE the first time, and then CTRL+R, is recommended to retrieve the command from the shell history.

Previous exercise 3: installing fio

To install the fio command in exta, the following instructions must be completed:

- Update the sources of the system repository using the apt command, update option. If the sources were just updated by the mdadm installation, this step would not be necessary:

  ```
  # apt update
  ```

- Install the command fio:

  ```
  # apt install fio
  ```

Tasks to be completed

1. Update the system repository sources.
2. Install the fio command.

To be uploaded

- The result of the execution of the fio installation. Just copy the text.

Work in the lab

The starting situation is that the external machine must have eight discs of 300 MB with the speed limited to 200 iops. It is important to make sure that the work is done on the exta machine and that the correct version of the discs is installed, along with the mdadm and fio commands.

Creating RAIDs and measuring their performance

The creation of the first RAID is going to be guided, and so it is important that not only the commands are executed, but that it is understood what is being done. In this way, the rest of the session will be straightforward, as CRTL+R will simply be used to retrieve the commands and edit them properly.

Create a RAID 0 with two discs: sda and sdb, with the name myraid0. No hot-spare discs are used because this type of RAID does not implement redundancy. This could be done with the command:

```
# mdadm --create /dev/md/myraid0 --level=0 --raid-devices=2 /dev/sd[a-b]
```

To establish how many RAIDs there are in the system, one possibility is to execute the command:

```
# cat /proc/mdstat
```

It is possible to obtain details of the RAID using `mdadm --detail`, specifying the name of the RAID. The command will return RAID information, such as size, status of the discs, etc.

```
# mdadm --detail /dev/md/myraid0
```

In this case, it should be noted that the size is as expected with a RAID 0 of two discs of 300 MB, the final space available in the RAID is that of the sum of the discs, as indicated in RAID 0 .

To measure speed performance, the fio command shown below must be executed. Warning, the RAID name should be changed as necessary. In this first case of RAID 0 (myraid0), the correct name is already written, so just copy and paste:

```
# fio --filename="/dev/md/myraid0" --direct=1 --norandommap --rw=randread
--ioengine=libaio --bs=512B --iodepth=16 --runtime=60 --name=totaliops200
```

The reading speed measurement will start, and the progress percentage will be displayed on the console. Some time is needed until it reaches 100%. The final information in the command will look like the following (it is recommended that the terminal be large enough to avoid being confused by line breaks):

```
fio-2.16
Starting 1 process
Jobs: 1 (f=1): [r(1)] [100.0% done] [202KB/0KB/0KB /s] [404/0/0 iops] [eta 00m:00s]
totaliops200: (groupid=0, jobs=1): err= 0: pid=1765: Thu Oct 24 13:11:28 2019
        read : io=11868KB, bw=202285B/s, iops=395, runt= 60075msec
          slat (usec): min=53, max=948, avg=74.37, stdev=23.30
          clat (usec): min=2, max=136319, avg=40414.65, stdev=33629.90
                lat (usec): min=345, max=136384, avg=40489.02, stdev=33619.49
          clat percentiles (usec):
                |  1.00th=[  354],  5.00th=[  442], 10.00th=[  454], 20.00th=[
478],
                | 30.00th=[ 3280], 40.00th=[22656], 50.00th=[41728], 60.00th=
[60160],
                | 70.00th=[76288], 80.00th=[78336], 90.00th=[80384], 95.00th=
[81408],
                | 99.00th=[82432], 99.50th=[83456], 99.90th=[83456], 99.95th=
[87552],
                | 99.99th=[127488]
           lat (usec) : 4=0.01%, 250=0.01%, 500=24.47%, 750=4.48%, 1000=0.40%
           lat (msec) : 2=0.06%, 4=1.47%, 10=3.09%, 20=5.38%, 50=15.49%
           lat (msec) : 100=45.09%, 250=0.03%
        cpu          : usr=0.03%, sys=3.57%, ctx=19278, majf=0, minf=11
        IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=99.9%, 32=0.0%, >=64=0.0%
                submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
                complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.1%, 32=0.0%, 64=0.0%,
>=64=0.0%
                issued    : total=r=23735/w=0/d=0, short=r=0/w=0/d=0,
drop=r=0/w=0/d=0
                latency   : target=0, window=0, percentile=100.00%, depth=16
Run status group 0 (all jobs):
READ: io=11867KB, aggrb=197KB/s, minb=197KB/s, maxb=197KB/s, mint=60075msec,
maxt=60075msec
```

The fifth line (the one that begins with `read`), contains the final result of the measurement of input/output operations (in this case reads). The figure in the example is 'iops=395' but it may be slightly different. It is interesting to compare this result with the expected result. As it is a RAID 0 (striping), the expected speed will be that of a disc, multiplied by the number of discs. In this case, as the read speed of a disc is 200 iops, and there are two discs in the RAID, the expected speed is 400 iops. The experimental result of 395 iops is very close, the relative error being (400-395)/400*100=1.25%.

To measure the writing speed, the command to execute is:

```
# fio --filename="/dev/md/myraid0" --direct=1 --norandommap --rw=randwrite
--ioengine=libaio --bs=512B --iodepth=16 --runtime=60 --name=totaliops200
```

The measurement process will start again, taking a few seconds to reach 100%. The final result will be something similar to:

```
fio-2.16
Starting 1 process
Jobs: 1 (f=1): [w(1)] [100.0% done] [0KB/195KB/0KB /s] [0/391/0 iops] [eta 00m:00s]
totaliops200: (groupid=0, jobs=1): err= 0: pid=1773: Thu Oct 24 13:14:56 2019
        write: io=11742KB, bw=200224B/s, iops=391, runt= 60049msec
          slat (usec): min=10, max=9190, avg=54.56, stdev=77.67
          clat (usec): min=885, max=84057, avg=40852.66, stdev=9742.20
            lat (msec): min=1, max=84, avg=40.91, stdev= 9.74
          clat percentiles (usec):
                |  1.00th=[18560],  5.00th=[24448], 10.00th=[28032], 20.00th=
[33536],
                | 30.00th=[35584], 40.00th=[38656], 50.00th=[40704], 60.00th=
[43264],
                | 70.00th=[45824], 80.00th=[49408], 90.00th=[53504], 95.00th=
[57088],
                | 99.00th=[63232], 99.50th=[66048], 99.90th=[70144], 99.95th=
[71168],
                | 99.99th=[74240]
           lat (usec) : 1000=0.01%
           lat (msec) : 2=0.03%, 4=0.14%, 10=0.02%, 20=1.35%, 50=80.76%
           lat (msec) : 100=17.71%
        cpu          : usr=0.00%, sys=2.77%, ctx=12518, majf=0, minf=8
        IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=99.9%, 32=0.0%, >=64=0.0%
                submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
                complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.1%, 32=0.0%, 64=0.0%,
>=64=0.0%
                issued    : total=r=0/w=23483/d=0, short=r=0/w=0/d=0,
drop=r=0/w=0/d=0
                latency   : target=0, window=0, percentile=100.00%, depth=16
Run status group 0 (all jobs):
WRITE: io=11741KB, aggrb=195KB/s, minb=195KB/s, maxb=195KB/s, mint=60049msec,
maxt=60049msec
```

In this case, the write speed is 391 iops, also quite close to the expected 400 iops write speed.

Delete the created RAID and create a new RAID 0 as an exercise – and in this case with four discs.

To eliminate the RAID, as a file system has not yet been created and it has not been mounted on the system, it will be enough to stop the RAID, indicating the name of the RAID, with the command:

```
# mdadm --stop /dev/md/myraid0
```

It is a good idea to remove any configuration information from the RAID, and so make the disc ready for use in subsequent RAIDs. This can be done with the command:

```
# mdadm --misc --zero-superblock /dev/sd[a-b]
```

RAID myraid0 should no longer be present. This can be checked with the command:

```
# cat /proc/mdstat
```

The system is now ready for the creation of a new RAID.

Exercise 1: creating a RAID 0

Create a RAID 0 with four discs, /dev/sd[a-d], which will be called myraid0. The capacity of the system must now be established. Does the observed size match the predictions explained in the RAID 0 section? Similarly, the RAID read and write speed must be measured in iops. What is the percentage difference in both measurements from the expected value?

Tasks to be completed

1. Create the RAID 0 with the four discs.
2. Take note of the available storage space on the RAID. Does it match the theoretical model?
3. Measure read and write speed of the new RAID.
4. Stop the RAID 0 and delete any configuration information to leave the discs clean.

To be uploaded

- The output of the command `cat /proc/mdstat` once the RAID has been created.
- The output of the command `mdadm --detail /dev/md/myraid0`
- The output of the `fio` commands to measure reading and writing speed.
- Commands used to stop the RAID 0 and to delete configuration information.
- Answers to the questions in this section, together with the calculation of the percentage difference of the experimental speeds versus those expected.

Exercise 2: creating a RAID 1

Create a RAID 1 with two discs called myraid1. In this case, by supporting this type of fault-tolerant RAID, it is possible to add hot-spare discs so add an additional disc of this type. The create section of the mdadm command shows the use of the `--create` option to create RAIDs with spare discs. Warning, the command that appears there must be customized in the appropriate way. In this case, the name `myraid1` is proposed, the option `--level` must be 1, as it is a RAID 1. The number of active discs will be 2, so the option `--raid-devices` will be two. The spare disc will be specified with `--spare-devices=1`. The discs used will be /dev/sd[a-c]. The system will take a few seconds to create. It is possible to observe the state of the system, even the percentage of creation with `cat /proc/mdstat`.

Similarly, as has been done before, the size of the available storage in the RAID must be measured, along with its speed. Does the observed size match the predictions explained in the RAID 1 section? What is the percentage difference in speed measurements from the expected value?

When the RAID speed measurement has been completed, a disc failure, for example sda, will be simulated and what happens in the system must be observed. The command to mark the sda disc as faulty is:

```
mdadm /dev/md / myraid1 --fail /dev/sda
```

As soon as the sda disc is marked as faulty, it is possible to see how the system is rebuilt using `cat /proc/mdstat` or `mdadm --detail /dev/md/myraid1`.

Tasks to be completed

1. Create a RAID 1 with the three discs: two forming RAID 1 and one spare disc.
2. Take note of the available storage space on the RAID. Does it match the theoretical model?
3. Measure RAID 1 read and write speed.
4. Mark sda as faulty and wait for the system to reconfigure.
5. Take note of the available storage space on the RAID again. Has there been any change?
6. Stop the RAID 1 and leave the discs clean by deleting any configuration information.

To be uploaded

- Command used to create RAID 1, with two discs in RAID plus a * spare * disc.
- The output of the command `cat /proc/mdstat` once the RAID has been created.
- The output of the command `mdadm --detail /dev/md/myraid1`.
- The output of the `fio` commands to measure the reading and writing speed.
- Calculation of the percentage difference of the experimental speeds versus those expected.
- The output of the command `mdadm --detail /dev/md/myraid1` after marking sda as faulty and waiting for the RAID to be reconfigured.
- Commands used to stop the RAID 1 and to delete configuration information.
- Answer to the questions in this section.

Including RAIDs in the file system

RAID 10, as described in RAID 10, combines the advantages of RAID 0 (striping) and RAID 1 (mirroring).

Create a RAID 10 with the name myraid10 – initially with four discs in RAID and two hot-spare. The creation time will be longer than in the previous case when making redundancy with more discs. It is a good idea to watch the build process with the command `cat /proc/mdstat` until it completes.

It is possible to observe the RAID configuration with the command:

```
# mdadm --detail /dev/md/myraid10
```

In this case, not only the size is relevant, but the configuration they form. There are two sets of discs called `set-A` and `set-B`. Each set is a mirror copy of the other (RAID 1). Additionally, the two discs in a set form a RAID 0.

Once any type of RAID has been created, it is possible to create a file system on it and to mount it in an existing file system. To create an ext4 file system on the created RAID 10, the command is:

```
# mkfs -t ext4 /dev/md/myraid10
```

To mount the RAID in the file system, the directory `/mnt/test`:

```
# mkdir /mnt/test
```

The RAID will be mounted with the command:

```
# mount /dev/md/myraid10 /mnt/test/
```

Finally, `/etc` will be copied to verify that the RAID works:

```
# rsync -avP /etc /mnt/test
```

Exercise 3: creating a RAID 10

Create a RAID 10 that will also be incorporated into the file system. In this case, the speed and space characteristics of this type of RAIDs will also be measured.

Tasks to be completed

1. Create RAID 10 with six discs: four forming the RAID 10 and two as spare discs.
2. Take note of the available storage space on the RAID. Does it match the theoretical model?
3. Measure the read and write speed of the RAID 10. What is the relative difference from the theoretical model?
4. Create an ext4 file system on top of the RAID 10.
5. Create the directory `/mnt/test` and mount the RAID at that point.
6. Copy the folder `/etc` in `/mnt/test` and check the occupied space.

To be uploaded

- Command used to create the RAID 10, with 4 RAID discs plus two *spare* discs.
- The output of the command `mdadm --detail /dev/md/myraid10` once the RAID has been created.
- The output of the `fio` commands to measure reading and writing speeds.
- Calculation of the percentage difference of the experimental speeds versus those expected.
- The output of the command `df` once the RAID has been mounted and /etc has been copied.

Exercise 4: testing the operation of RAID 10

In this case, in addition to capacity and speed, simulate the fault tolerance of the RAID, and carry out the following actions:

Tasks to be completed

1. Mark sda as faulty and wait for the system to reconfigure.
2. Take note of the available storage space on the RAID again. Have there been any changes? What happened to the disc marked as faulty?
3. Simulate a new failure marking as defective the disc that was initially spare and that has just been incorporated into the RAID (sdf). What happens now? Is the capacity and speed of the system maintained? (Only test the read speed, as the write test would destroy the file system).
4. Two discs have been removed sequentially: first sda, the system was allowed to reconfigure, and then sdf, the disc that had replaced sda, was removed. The system has not failed. Would this behaviour have been the same with the sequential removal of any combination of discs?
5. If two discs had been removed at the same time, would the system have supported the removal of any pair of discs without showing failure? In which cases would the system have tolerated the simultaneous failure of two discs and in which cases would it not?
6. Delete the two discs marked as faulty. To do this, the following command must be executed, after modifying it appropriately: `# mdadm --remove /dev/md/myraid10 /dev/sda`
7. Unmount the RAID: `# umount /mnt/test`
8. Stop the RAID 10 and delete any configuration information, to leave the discs clean.

To be uploaded

- The output of the command `mdadm --detail /dev/md/myraid10` after marking sda as faulty and waiting for it to be reconfigured.
- The output of the command `mdadm --detail /dev/md/myraid10` after marking the disc just incorporated into the system as faulty and waiting for it to be reconfigured.
- The output of the command `fio` to measure the reading speed.
- The output of the command `mdadm --detail /dev/md/myraid10` after removing the two defective discs.
- Commands used to unmount, stop RAID 10, and to delete configuration information.
- Answer to the questions in this section.

Growing RAIDs and testing their Fault Tolerance

It is possible to grow a RAID that is included in the file system by adding discs to the system. It should be noted that growing a RAID consists of adding active discs to the RAID, not adding spare discs, although it is also possible to add a disc, which by default will be added as spare, and then actively incorporate it into the system.

As an example for this section create a RAID 5 with four active discs:

```
# mdadm --create /dev/md/myraid5 --level=5 --raid-devices=4 /dev/sd[a-d]
```

The creation process can be observed by:

```
# cat /proc/mdstat
```

Once the RAID is created, format it and mount it in `/mnt/test`:

```
# mkfs -t ext4 /dev/md/myraid5
```

```
# mount /dev/md/myraid5 /mnt/test/
```

Use the RAID to copy /etc:

```
# rsync -avP /etc /mnt/test
```

It is possible to add directly an active disc. To do this, the option `--grow` must be used, indicating the new disc, in this case sde, and the new RAID configuration with five discs with the option `--raid-devices`:

```
# mdadm --grow /dev/md/myraid5 --add /dev/sde --raid-devices=5
```

Reconfiguration time will be appreciable as parity must be recalculated and written for each group of data segments.

The RAID size can be checked with:

```
# mdadm --detail /dev/md/myraid5
```

However, the file system has not changed its size. It can be observed if the command `df -h` is executed. To make the file system grow and occupy all the RAID space, execute the command:

```
# resize2fs /dev/md/myraid5
```

Check with `df -h` how the file system has grown to occupy all the space in the RAID.

Add spare discs directly with `--add`. This process is almost instantaneous as they are not actively incorporated, so no additional information must be recalculated or written. To add sdf as spare:

```
# mdadm --add /dev/md/myraid5 /dev/sdf
```

To make the spare disc active, simply set the new size of the RAID as six discs:

```
# mdadm --grow /dev/md/myraid5 --raid-devices=6
```

Again, the process can be expensive as parity must be recalculated and rewritten on the discs.

Exercise 5: creating a RAID 5 and growing it

Create a RAID 5 with four active discs, observe its performance (capacity and speed), and subsequently increase its size. First, an active disc will be directly added to the system. A spare disc will then be added, which will later become active, thus growing the array.

Tasks to be completed

1. Create a RAID 5 with four discs: sd[a-d]. Immediately with `cat /proc/mdstat` observe the creation time.
2. Take note of the available storage space on the RAID. Why is this space less than that of the sum of four discs?
3. Measure the RAID 5 read and write speed. What is the difference between the read and write speed? Does the result match the theoretical model?
4. Create an ext4 file system on top of RAID 5.
5. Mount the RAID on the directory `/mnt/test`.
6. Check the size of the file system with `df -h`.
7. Copy the / etc directory with the command `rsync -avP /etc/mnt/test`.
8. Directly add sde as active disc. Now RAID 5 has five active discs.
9. Check the size of the RAID.
10. Check the size of the file system with `df -h`. Has the size of the RAID changed equally?
11. Grow the file system to occupy all the space on the RAID. Check now that it has changed.
12. Add sdf as a spare disc. What will be the space available in the RAID and its theoretical speed?
13. Make sdf active by resizing the RAID. Now RAID 5 has six active discs.

14. Make the file system grow so that it takes advantage of all the RAID space.
15. Check the available space in both the RAID and the file system.
16. Unmount the file system.
17. Measure the RAID read speed (the write test would destroy the file system).
18. Mount the file system.

To be uploaded

- Command used to create RAID 5 and time required.
- The output of the command `mdadm --detail /dev/md/myraid5` once the RAID 5 has been created, with four discs.
- The output of the `fio` commands to measure the reading and writing speed.
- The output of the command `df -h` once the RAID has been mounted and /etc has been copied.
- Command to add the sde disc as active to the RAIDs.
- The output of the command `df -h` once sde has been added.
- The output of the command `mdadm --detail /dev/md/myraid5` once the RAID 5 has five discs.
- Command to make the FS occupy the entire RAID, once sde has been added.
- Commands to add the sdf disc as spare initially and then make it active.
- The output of the command `mdadm --detail /dev/md/myraid5` once the RAID 5 has six discs.
- Output of the command `df -h`.
- The output of the command `fio` to measure reading speed with six discs.
- Answer to the questions in this section.

Exercise 6: testing RAID 5 fault tolerance

The starting situation is a RAID 5, with six active discs in the RAID, mounted in `/mnt/test`, and with the /etc directory copied in it. In this section, the fault tolerance of the system is going to be checked, simulating a disc failure.

As described in the initial section on RAID 5, this type of RAID supports the failure of one disc, whichever it may be, since the data fragments are distributed over the N-1 discs, leaving one disc to store the error-correcting code. This parity information is also distributed in an interleaved manner across the discs, so there is no dedicated parity disc. In this way, it does not matter which disc is failing.

Tasks to be completed

1. Add as spare the sdg disc.
2. Mark the sda disc as faulty. It must be immediately watched what happens to `cat /proc/mdstat`. What is the time required for reconstruction?
3. During and after rebuilding, it is possible to browse the file system in `/mnt/test/etc` and check that the system is still working.
4. Now mark a second disc as faulty, in this case the sdb disc. It is possible to immediately see what happens to `cat /proc/mdstat`. What is the time required for reconstruction? The answer must be explained.
5. Browse the RAID file system. Why is RAID still working if two discs, sda and sdb, have really failed?
6. Finally, the RAID must be unmounted, stopped, and any configuration of the discs

erased, to leave them ready for the next exercise.

To be uploaded

- Output of the command `mdadm --detail /dev/md/myraid5` once the spare disc has been added.
- Output of the command `mdadm --detail /dev/md/myraid5` once the reconstruction has finished after the failure of sda.
- Output of the command `mdadm --detail /dev/md/myraid5` after the failure of sdb.
- Commands used to unmount the RAID, stop it, and remove the configuration from the discs.
- Answer to the questions in this section.

Exercise 7: creating a RAID 6 and measuring its performance

The RAID 6 offers a double distributed fault tolerance scheme, thereby tolerating the simultaneous failure of two RAID discs.

Create a RAID 6 (`--level=6`) with four discs, the minimum number of discs in a RAID 6. This RAID will be called myraid6. It is possible to try to create it with fewer discs, but mdadm will give an error and warn of the minimum number needed.

Tasks to be completed

1. Create a RAID 6 with four discs: sd[a-d]. Immediately watch how long has the RAID creation taken with `cat /proc/mdstat`.
2. Make a note of the available storage space in the RAID once it is created. Why is this space less than that of the sum of four discs and less than that of RAID 5?
3. Measure the RAID 6 read and write speed with four discs. What is the difference between reading and writing speed? Does the result match the theoretical model for writes? Calculate the percentage error in the writing speed.
4. Create an ext4 file system on top of RAID 6.
5. Mount the RAID on the directory `/mnt/test`.
6. Copy the /etc directory with the command `rsync -avP /etc/mnt/test`

To be uploaded

- Command used to create RAID 6 and time needed.
- The output of the command `mdadm --detail /dev/md/myraid6` once the RAID 6 has been created, with four discs.
- The output of the `fio` commands to measure reading and writing speed.
- The output of the command `df -h` once the RAID has been mounted and /etc has been copied.
- Answer to the questions in this section.

Exercise 8: testing RAID 6 fault tolerance

In the following exercise observe the performance of this type of RAID. To do this, the RAID is going to be grown by adding two spare discs, and the simultaneous failure of two discs will later be forced. Finally, grow RAID 6 to reach six active discs and measure the speed.

Tasks to be completed

1. Add sde and sdf as spare discs. Does available space or speed change?
2. Mark the sda and sdb discs as faulty, with the command `mdadm /dev/md/myraid6 --fail /dev/sd[a,b]`. Immediately observe what happens to `cat /proc/mdstat`. Does a system rebuild occur? Has the RAID tolerated the simultaneous failure of 2 discs?
3. At this point it is possible to remove the faulty discs from the RAID with the `--remove` option.
4. Add two new active discs (not *spare*) to the RAID, in this case sdg and sdh.
5. The file system must be resized with `resize2fs /dev/md/myraid6`
6. Unmount the file system.
7. Measure the read and write speed of RAID 6 with six discs. Does the result match the theoretical model for writes? Calculate the percentage error in the writing speed.
8. Finally, RAID 6 must be stopped. Additionally, any disc configuration must be removed.

To be uploaded

- Command to add the discs sde and sdf to the RAID as spares.
- The output of the command `mdadm --detail /dev/md/myraid6` once the spare discs have been added.
- The output of the `fio` commands to measure reading and writing speed.
- The output of the command `mdadm --detail /dev/md/myraid6` once sda and sdb have been marked as faulty.
- The output of the command `mdadm --detail /dev/md/myraid6` once sdg and sdh have been added as active discs.
- The output of the command `df -h` once the file system has been resized.
- The output of the command `fio` to measure the speed of reading and writing with the RAID with six discs.
- Commands used to unmount the RAID, stop it, and remove the configuration from the discs.
- Answer to the questions in this section.

DSI - Laboratory Session 7: DSI Review

28 May 2021

Contents

Goals

- Reinforce the learning of several aspects of the subject, with special emphasis on storage management and shell scripts.

Previous work

Before attending the lab session, students must:

- Review previous laboratory assignments, especially those related to shell scripting and storage.
- Do the exercises labelled as 'previous exercise'.

To be uploaded

- Results of previous exercises.

Previous exercise 1: preparing NETinVM for the laboratory session

During the lab session, a modified version of 'exte' will be used, so the first step is to update, if needed, the script 'Run my machines' and restore the backup labelled 'p7' (the full name of the file is 'kvm_machines_2020-11-19_12-24_p7.tgz').

Tasks to be completed

- Use the 'Configure my machines' icon to boot exclusively the 'exte' KVM machine.
- Download the file 'kvm_machines_2020-11-19_12-24_p7.tgz' and save it in the backup folder '/home/user1/netinm/backups'.
- Restore the newly downloaded backup.
- Execute, as 'root' in 'exte', the command 'lsblk' and check that the disc '/dev/vda' has two partitions.

To be uploaded

- Justification that the disc '/dev/vda' of 'exte' has two partitions.

Previous exercise 2: find out the configuration of 'exte'

Using the logical volumes and RAID tools studied during the course, find out what type of storage is used in this version of 'exta'.

Tasks to be completed

- Find out the storage configuration of 'exte' and answer the following questions, adequately justifying the answers:
    1. On which partition is the root file system hosted?
    2. Is RAID used?
    3. Are logical volumes used?
    4. What type of storage (partition, RAID, and logical volume) is used for '/home'?
    5. What file system is used for '/home'?

To be uploaded

- Answers to the questions posed, including the justification requested.

Previous exercise 3: script to prepare test data

After reading the introduction to the section 'Playing with data and backups ', make a script for the *bash* command interpreter that creates the directories and files indicated by a CSV file with the following format:

- Each line is a record with three fields: directory, file, content for file.
- Fields are separated by commas.
- The first line is the header.

The script should work like this:

- It must create the directories in the working directory.
- It must read the listing from the standard input.
- It must ignore the header.
- For each record:
    - If the directory does not yet exist, it must create it. (If it exists, the script should not fail).
    - It must create the file in the indicated directory.
    - The file must contain exactly the indicated data, without adding line breaks or any other additional character.

During development, it is recommended to use a reduced version of the listing. For example:

```
user1@exte:~/p7$ ./list_generation.py > list.txt
user1@exte:~/p7$ head -n 5 list.txt >short_list_5.txt
user1@exte:~/p7$ cat short_list_5.txt
directory,file,file_content
process,f0012,570d2de0b8df0e6cface5f51956c69bf90659964192e5cd28be7987464772620
process,f0924,3da71f773af23f68f2a8d284fe2c1331d9b387b1ce3796283ce62700d6c58307
process,f0878,5e075233c6d6160c9329c18cddce59e6a5d26980e1d3dea2bf1786765155e156
process,f0432,9290851ba77c5b69df944ffc9a671cdc32384229e2b4606729d1e65b48a63c2a
user1@exte:~/p7$ mkdir tests
user1@exte:~/p7$ cd tests
user1@exte:~/p7/tests$ ../list_processing.sh < ../short_list_5.txt
user1@exte:~/p7/tests$ ls -l
total 4
drwxr-xr-x 2 user1 user1 4096 Nov 18 15:37 process
user1@exte:~/p7/tests$ ls -l process
total 16
-rw-r--r-- 1 user1 user1 64 Nov 18 15:37 f0012
-rw-r--r-- 1 user1 user1 64 Nov 18 15:37 f0432
-rw-r--r-- 1 user1 user1 64 Nov 18 15:37 f0878
-rw-r--r-- 1 user1 user1 64 Nov 18 15:37 f0924
user1@exte:~/p7/tests$ cat process/f0012
570d2de0b8df0e6cface5f51956c69bf90659964192e5cd28be7987464772620user1@exte:~
/p7/tests$
```

Note that since the file 'process/f0012' does not include a line break at the end, the shell prompt appears after the content of the file. Also, note that all files must have the same length: 64 bytes.

Additionally, when everything seems right, a test should be made with the verifier. Of course, the verifier only works with a complete list (25 directories and 100 files per directory). To avoid dragging errors from previous tests, it is best to remove the previous tests and generate everything from scratch:

```
user1@exte:~/p7/tests$ cd ..
user1@exte:~/p7$ rm -rf tests
user1@exte:~/p7$ mkdir tests
user1@exte:~/p7$ cd tests
user1@exte:~/p7/tests$ ../list_processing.sh < ../list.txt
user1@exte:~/p7/tests$ ../list_check.sh < ../list.txt
Comprobación correcta  /  2500 files :-)
user1@exte:~/p7/tests$
```

Note

- It has been assumed that the script to be developed in this section has been called 'list_processing.sh'.
- The programs 'list_generation.py', and 'list_check.sh' are explained in the section 'Playing with data and backups'.

To be uploaded

- *Script* done.

Work in the laboratory

Installing a RAID 1

'exte' it is going to be used to learn what to do in the following example:

- There is a desktop computer on which some previous work was done and the results are stored in the '/home' folder.
- This project, which began as small tests, has become more complex and takes up more space, so the '/home' file system is almost full.
- In addition, it is our goal to have an independent file system to store data.
- It is an objective to tolerate failures of a disc, both in '/home' and in the new file system for data, so it has been decided to switch to using a RAID 1 system.
- Two new 15 GB discs each have been purchased, hot-pluggable (without turning off the machine).
- The transition must / done without neither the computer nor the '/home' data becoming unavailable.

The following steps are necessary to make this project a reality:

- Add the discs to 'exte'.
- Build the RAID 1.
- Add it to the existing volume group.
- Move the logical volume containing '/home' to that group.
- Reuse the space freed by '/home' on the disc '/dev/vda'.
- Create a logical volume for the data.
- Create a file system for the data.
- Mount the file system and configure the system to use it automatically (at boot time).
- Prepare the new system so that 'user1' can use it comfortably and access it as '/home/user1/data'.

Exercise 1: add the discs to 'exte'

Tasks to be completed

- Using the 'Virtual Machine Manager', create two 15 GB discs with a QCOW2 format, of type *virtio*, stored in the *Storage Pool* called 'netinvm-development'.
- Check that the discs are visible to 'exte'.

To be uploaded

- Evidence that the discs have been correctly added to 'exte'.

Exercise 2: create the RAID 1

Tasks to be completed

- On each disc:
    - Create a GPT partition table.
    - Create a 'Linux RAID' type partition that occupies all the free space on the disc.
- Create a RAID 1 named 'iad' that includes both partitions.
- Check that it has been created correctly, waiting, if necessary, for the initial configuration to finish.

To be uploaded

- Output of the command '`mdadm --detail`' on the newly created RAID.

Exercise 3: move '/home'

Tasks to be completed

- Add the RAID disc to volume group 'vg1'.
- Move, using '`pvmove`', the logical volume 'home' to the new disc.
- Remove '/dev/vda2' from the volume group.
- Using '`pvremove`', remove the physical volume from '/dev/vda2'.
- Check that '/home' is still available. (Hint: log in as 'user1' using another terminal).
- Edit the partition table of '/dev/vda'
    - Delete the freed partition ('/dev/vda2').
    - Extend the partition '/dev/vda1' to occupy the entire disc.
    - Write the changes.
    - Exit.
- Check the partition size with '`lsblk`'. In case it has not been changed, run '`partprobe`' and check again.
- Expand the root file system to occupy the 20 GB partition '/dev/vda1'.

- Expand the logical volume 'home' to occupy 5 GB, while extending the file system it contains.
- Check that both '/' and '/home' have expanded properly.

To be uploaded

- Output of the command '`df -hT / /home`'.
- Output of the '`pvs`' command.
- Output of the '`vgs`' command.
- Output of the command '`lvs -o + devices`'.

Exercise 4: creating the file system for data

Tasks to be completed

- Create a new logical volume ('data') that occupies all the free space in the volume group.
- Create an 'ext4' file system in it.
- Edit '/etc/fstab' so that it is automatically mounted in '/mnt/data'.
- Mount it using '`mount -a`'.
- Check that it has been mounted correctly.
- Create a folder '/mnt/data/user1'.
- Change the owner and group to be 'user1.user1'.
- Remove all permissions for the rest of the world.
- Create a symbolic link in '/home/user1' named 'data' and pointing to '/mnt/data/user1'. (This way, 'user1' will be able to access her folder in 'data' using '/home/user1/data').
- Log in as 'user1' in another terminal and check that it is possible to work on '/mnt/data/user1' using the symbolic link '/home/user1/data'.

To be uploaded

- Output of the command '`df -hT /mnt/data`'.
- Output of the command '`grep data /etc/fstab`'.
- Output of the '`lvs`' command.
- Output of the command '`ls -l /mnt/data /home/user1`'

Playing with data and backups

Developing tools for data processing often involves the following:

- Generate a certain set of files and directories that, although they meet certain requirements or follow certain patterns, change each time. (The fact that they change each time makes it

easier to detect possible programming errors).
- Make copies of the starting data and/or the results.
- Carry out functional tests and quickly return to the initial state.

During this section, exercises will be done that follow this pattern, and for this, the following programs are available:

list_generation.py
: This is a program that generates a random list of directories, files, and content that are going to be used to carry out the tests. The program generates the listing on standard output. The list is a CSV file with the following format:

  - Each line is a record with three fields: directory; file; file content.
  - The fields are separated by commas.
  - The first line is the header.

  The usual way of using it will be:

```
./list_generation.py > list.txt
```

  An example line is:

```
process,f0012,570d2de0b8df0e6cface5f51956c69bf90659964192e5cd28be7987464772620
```

  Where 'process' is the folder in which the file 'f0012' must reside, and whose content must be the string '570d2de0b8df0e6cface5f51956c69bf90659964192e5cd28be7987464772620'.

list_processing.sh
: Shell script that must have been developed before the laboratory session. (See 'Previous exercise 3: script to prepare test data').

list_check.sh
: Shell script that checks that the content of the current directory is consistent with the listing it reads from its standard input. For example:

```
user1@exte:~/p7$ ./list_generation.py > list.txt
user1@exte:~/p7$ mkdir tests
user1@exte:~/p7$ cd tests
user1@exte:~/p7/tests$ ../list_processing.sh < ../list.txt
user1@exte:~/p7/tests$ ../list_check.sh < ../list.txt
2500 files perfectly checked! :-)
user1@exte:~/p7/tests$
```

makes_changes.py
: Program that makes changes to files and directories in the current folder. For example:

```
user1@exte:~/p7/tests$ ../list_check.sh < ../list.txt
2500 files perfectly checked! files :-)
user1@exte:~/p7/tests$ ../makes_changes.py
user1@exte:~/p7/tests$ ../list_check.sh < ../list.txt
ERROR: Wrong content in file: USB/f0424 :-(
user1@exte:~/p7/tests$
```

Exercise 5: using *rsync*

In this exercise, a disc for backups must be added and then use the rsync program both to perform backups and to discard the changes made.

Tasks to be completed

- Using the script developed in previous exercise 3, fill in '/home/user1/data/' using the list of folders and files generated by 'list_generation.py'.

- Check the correct creation of the content using the program 'list_check.sh'.
- Add a new 15GB disc to 'exte' for backup.
    - The disc must be 'virtio' and it must appear in 'exte' as '/dev/vde'.
    - A single partition must be created using the entire disc.
    - An 'ext4' file system must be created on it.
    - The disc should be automatically mounted in '/mnt/bkup'.
    - The folder '/mnt/bkup/user1' must exist, and its owner and group must be 'user1.user1'.
- User 'user1' must have a symbolic link called 'bkup' that allows access to '/mnt/bkup /user1' using the path '/home/user1/bkup'.
- Using *rsync*, make a full copy of the contents of '/home/user1/data' in '/home/user1 /bkup/data'.
    - **Important:** this is needed to verify that the contents of the folder pointed to by the symlink are copied, and not the symlink itself.
    - *Hint:* use '/home/user1/data/' as source and include 'data' explicitly in the destination.
- Check the correct creation of the backup content using the program 'list_check.sh'.
- Make random changes to '/home/user1/data/.' using the program 'makes_changes.py'.
- Check, using the verifier, that '/home/user1/data/.' no longer passes the verification process.
- Discard changes made to '/home/user1/data/.' using *rsync* and the recently made backup.

To be uploaded

- Output of the following commands:

```
df -hT /mnt/bkup
ls -l /home/user/data/.
ls -l /home/user1/bkup/data
ls -l /home/user1
```

- Copy of the complete command to make the backup copy.
- Copy of the complete command to discard the changes.

Optional exercise 1: using snapshots

Assuming that the backup disc does not have to be used every time a test is done, perform the following actions using snapshots of the logical volume 'data', bearing in mind that logical volume management must be done as root.

- Verify that the content of '/home/user1/data/.' is correct using the program 'list_check.sh'.
- Prepare logical volume 'data' for testing.
- Verify that the content of '/home/user1/data/.' is correct using the program 'list_check.sh'.
- Make random changes to '/home/user1/data/.' using the program 'makes_changes.py'.
- Check, using the verifier, that '/home/user1/data/.' no longer passes the verification process.

- Discard changes made to logical volume 'data'.
- Verify that the content of '/home/user1/data/.' is correct again.

Tasks to be completed

- Prepare the logical volume management orders necessary for the exercise. (Note that volume group 'vg1' will not have free space for new logical volumes).
- Carry out the actions described.

To be uploaded

- Explanation of the commands used to manage the logical volume 'data'.
