Astroinformatics I (Semester 1 2024)

Working with Linux

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

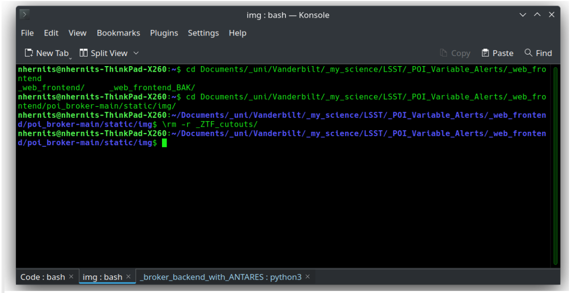
May 7, 2024

Motivation

You might have Linux on your own computer, or might encounter it when logging on to a cluster.

In many cases you will use the command line.

This week you will learn how to efficiently use Linux for working with (astronomical) data.

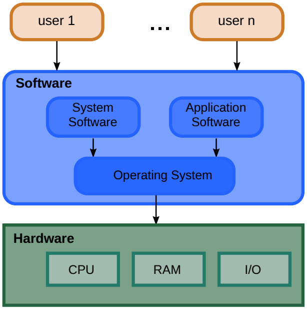


```
nhernits@nhernits-ThinkPad-X260:~$ cd Documents/_uni/Vanderbilt/_my_science/LSST/_POI_Variable_Alerts/_web_fro
ntend
_web_frontend/     _web_frontend_BAK/
nhernits@nhernits-ThinkPad-X260:~$ cd Documents/_uni/Vanderbilt/_my_science/LSST/_POI_Variable_Alerts/_web_fro
ntend/poi_broker-main/static/img/
nhernits@nhernits-ThinkPad-X260:~/Documents/_uni/Vanderbilt/_my_science/LSST/_POI_Variable_Alerts/_web_fronten
d/poi_broker-main/static/img$ \rm -r _ZTF_cutouts/
nhernits@nhernits-ThinkPad-X260:~/Documents/_uni/Vanderbilt/_my_science/LSST/_POI_Variable_Alerts/_web_fronten
d/poi_broker-main/static/img$ 
```

Intro: Operating Systems

An **operating system (OS)** is the software that **manages the hardware resources** associated with a computer (laptop, desktop, server, smartphone...) and allows other software to interact with it. In that way it interfaces between user and computer hardware devices.

In detail, it performs the basic tasks like file management, memory management, process management, handling input and output, user management, and controlling peripheral devices.



Intro: Operating Systems

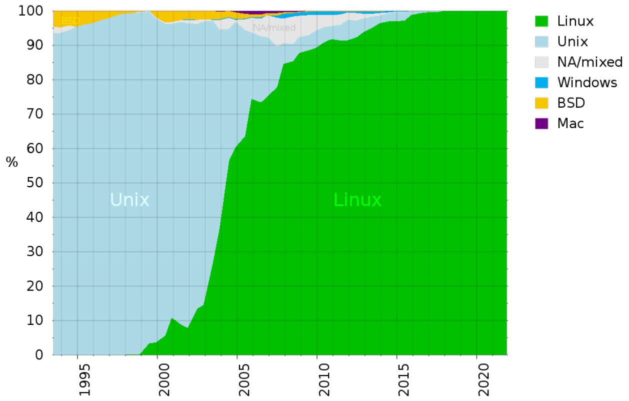
Some popular operating systems include Linux, Windows, macOS, iOS, Android (which uses the Linux kernel), etc.

In the modern days, when speaking about servers, more than 96% of the top 1,000,000 web servers use Linux. And talking about mobile devices, around 83% of smartphones are powered by Linux (Android)[1].

[1]source:
`https://www.enterpriseappstoday.com/stats/linux-statistics.html`

Intro: Operating Systems

Supercomputer Operating Systems



This figure shows the operating systems used on the supercomputers listed on the Top500 list. Data compiled from `http://top500.org/stats`.

Parts of Operating Systems

Operating System usually consist of the following components, where we here especially refer to Linux (I):

The Bootloader manages the way the computer boots. Most of the time this is seen by the users as the splash screen that pops up before giving way when the operating system takes over.

The Kernel is the most basic level of the OS. This is also the piece which also referred to as ”Linux”. The kernel is the core piece of an operating system and manages the CPU, memory, and peripheral devices.

Daemons are background services that start up on boot or once you have accessed the operating system.

The Shell is similar to the DOS prompt seen in Windows OS, this is the command process which enables a user to control the computer by entering commands typed into a text interface (commonly known as the terminal).

Parts of Operating Systems

Operating System usually consist of the following components, where we here especially refer to Linux (II):

The Graphical Server is the sub-system which displays graphics on your monitor.

The Desktop Environment is the part we normally interact with. This includes the built-in applications.

Applications are software not readily available when a basic operating system has been installed on a system. These are available for download onto a system.

Unix

To better understand what Linux is, we start with Unix:

Unix is a family of multitasking, multi-user computer operating systems deriving from the AT&T Unix, which was developed from 1969 on at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.

Unix was designed as a multi-user system on mainframe computers, with users connecting to it remotely via individual terminals. Everything was sent and received as text, without nowadays graphical user interface (GUI).

The DEC VT100 terminal, widely used for Unix timesharing in the 1980s.

Linux

Linux is an open-source operating system designed to behave similarly to a Unix system, such that most of the old shells and other text-based programs run on it.

Linux was first published on 17 September 1991 by Linus Torvalds.

Today, Linux is an open-source operating system that is used for computers, servers, mainframes, mobile devices, and embedded devices. Linux is one of the most widely supported operating systems as it is available on almost every major computer platform, including x86, ARM, and SPARC.

It is freely available for commercial as well as for non-commercial purposes. Any programmer can do some changes in the Linux kernel by doing some code and finally make it as the new distribution.

Linux Distributions

Linux is available in different **distributions**.

A Linux distribution (also called as Linux distro) is a **collection of software** that is based upon the Linux kernel. Each Linux distribution has the same core components, but each will have a different appearance and some of the software will differ. There are more than 600 Linux distributions, but only a few are widely used.

Distros can be commercially backed such as Fedora (Red Hat), openSUSE (SUSE), and Ubuntu (Canonical Ltd.) or entirely community driven such as Debian, Slackware, Gentoo, Arch Linux.

Linux Distributions

Linux is available in different **distributions**.

A Linux distribution (also called as Linux distro) is a **collection of software** that is based upon the Linux kernel. Each Linux distribution has the same core components, but each will have a different appearance and some of the software will differ. There are more than 600 Linux distributions, but only a few are widely used.

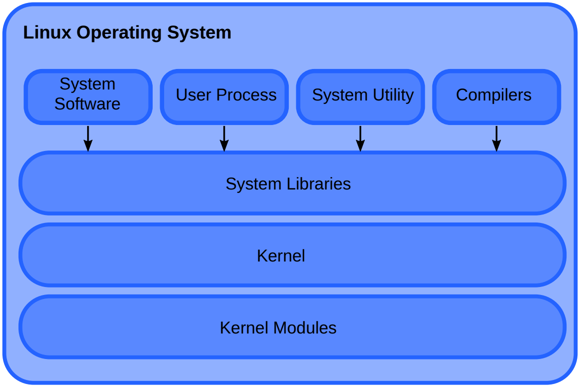
Distros can be commercially backed such as Fedora (Red Hat), openSUSE (SUSE), and Ubuntu (Canonical Ltd.) or entirely community driven such as Debian, Slackware, Gentoo, Arch Linux.

I'm using Kubuntu (Ubuntu 22.04.3 LTS with KDE desktop).

fedora openSUSE ubuntu kubuntu

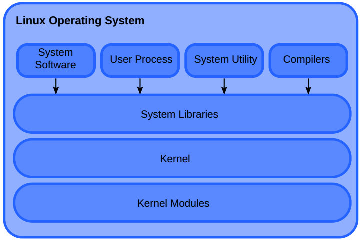
Linux Architecture

The Architecture of the Linux Operating System can be depicted as follows:



Linux Architecture

The most important three components are:



Kernel: This is the core part of Linux which is responsible for all major activities. It consists of various modules and it interacts directly with the underlying hardware. The Kernel provides the required abstraction of low-level hardware details to system or application programs.

System Libraries: These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.

System Utility: System Utility programs are responsible to do specialized, individual level tasks.

Linux Architecture

Kernel Mode vs. User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire computer.

Linux Architecture

Kernel Mode vs. User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire computer.

In **user mode**, the executing code has no ability to directly access hardware or reference memory. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

Linux Architecture

Kernel Mode vs. User Mode

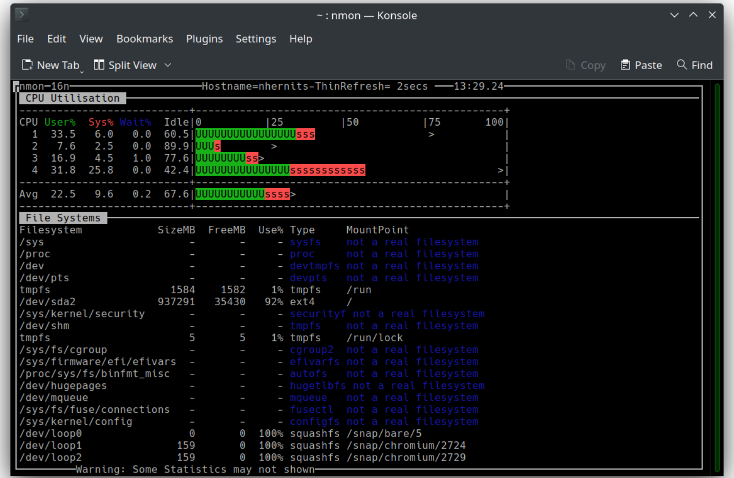
Kernel component code executes in a special privileged mode called **kernel mode** with complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire computer.

In **user mode**, the executing code has no ability to directly access hardware or reference memory. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

It is possible to display the usage of resources in kernel mode and user mode as shown in the following screenshot.

Linux Architecture

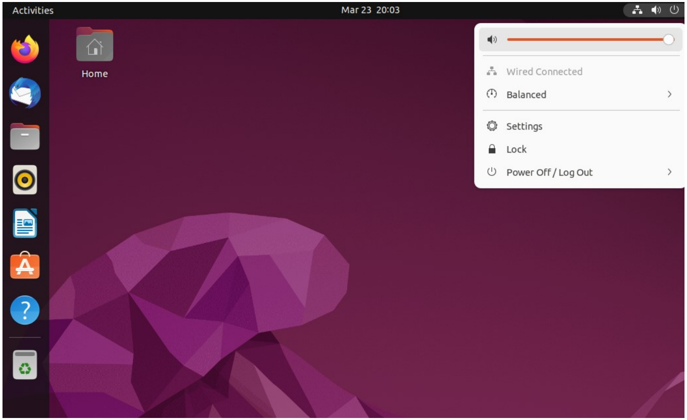
usage of computer resources, shown using `nmon`:



```
nmon—16n————————————————————Hostname=nhernits-ThinRefresh= 2secs ——13:29.24—
 CPU Utilisation
---------------------------+-------------------------------------------------+
CPU User%  Sys% Wait%  Idle|0          |25         |50          |75       100|
  1  33.5   6.0   0.0  60.5|UUUUUUUUUUUUUUUUsss                   >          |
  2   7.6   2.5   0.0  89.9|UUUs             >                               |
  3  16.9   4.5   1.0  77.6|UUUUUUUUss>                                      |
  4  31.8  25.8   0.0  42.4|UUUUUUUUUUUUUUUsssssssssssss                    >|
---------------------------+-------------------------------------------------+
Avg  22.5   9.6   0.2  67.6|UUUUUUUUUUUssss>                                 |
---------------------------+-------------------------------------------------+
 File Systems
Filesystem                SizeMB   FreeMB  Use% Type      MountPoint
/sys                           -        -     - sysfs     not a real filesystem
/proc                          -        -     - proc      not a real filesystem
/dev                           -        -     - devtmpfs  not a real filesystem
/dev/pts                       -        -     - devpts    not a real filesystem
tmpfs                       1584     1582    1% tmpfs     /run
/dev/sda2                 937291    35430   92% ext4      /
/sys/kernel/security           -        -     - securityf not a real filesystem
/dev/shm                       -        -     - tmpfs     not a real filesystem
tmpfs                          5        5    1% tmpfs     /run/lock
/sys/fs/cgroup                 -        -     - cgroup2   not a real filesystem
/sys/firmware/efi/efivars      -        -     - efivarfs  not a real filesystem
/proc/sys/fs/binfmt_misc       -        -     - autofs    not a real filesystem
/dev/hugepages                 -        -     - hugetlbfs not a real filesystem
/dev/mqueue                    -        -     - mqueue    not a real filesystem
/sys/fs/fuse/connections       -        -     - fusectl   not a real filesystem
/sys/kernel/config             -        -     - configfs  not a real filesystem
/dev/loop0                     0        0  100% squashfs  /snap/bare/5
/dev/loop1                   159        0  100% squashfs  /snap/chromium/2724
/dev/loop2                   159        0  100% squashfs  /snap/chromium/2729
         —Warning: Some Statistics may not shown—
```

Linux Architecture

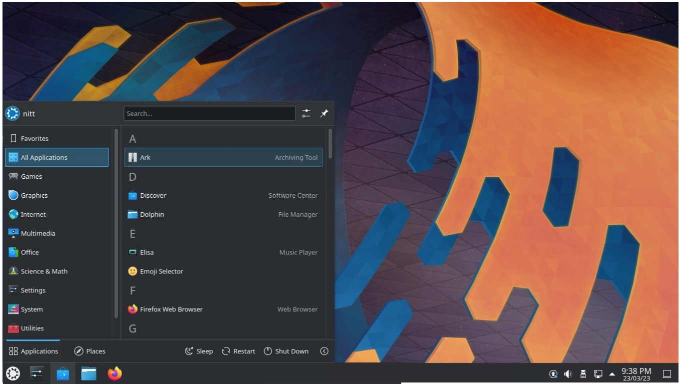
In Linux, a **desktop environment** (such as KDE Plasma or Gnome) provides the graphical user interface (GUI), while shells facilitate a command line interface.



GNOME 42 on Ubuntu 22.04 LTS

Linux Architecture

In Linux, a **desktop environment** (such as KDE Plasma or Gnome) provides the graphical user interface (GUI), while shells facilitate a command line interface.



KDE Plasma version 5.24.7 on Kubuntu 22.04 LTS

Linux Architecture

In Linux, a **desktop environment** (such as KDE Plasma or Gnome) provides the graphical user interface (GUI), while shells facilitate a command line interface.

There are different shells available:

- Korn shell
- Bourne shell
- C shell
- POSIX shell
- Bash shell
- zsh shell

Linux Architecture

In Linux, a **desktop environment** (such as KDE Plasma or Gnome) provides the graphical user interface (GUI), while shells facilitate a command line interface.

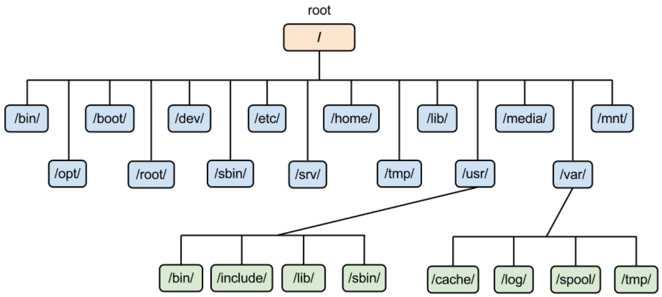
There are different shells available:

- Korn shell
- Bourne shell
- C shell
- POSIX shell
- Bash shell
- zsh shell

We are using here the **Bash shell** (**bash**), which btw. is also available for MacOS.

Linux File System Hierarchy

The entire Linux directory structure starts at the top (/) **root directory**.



Linux File System Hierarchy

The following provides a **summary of top-level Linux directories and their purposes** (I):

/ (root filesystem) The root filesystem is the top-level directory of the filesystem. It must contain all files required to boot the Linux system before other filesystems are mounted. After the system is booted, all other filesystems are mounted on well-defined mount points as subdirectories of the root filesystem.

`/bin` The /bin directory contains user executable files.

`/boot` Contains the static bootloader and kernel executable and configuration files required to boot a Linux computer.

`/dev` Device files for every hardware device attached to the system. These are not device drivers, but represent each device on the computer and facilitate access.

`/etc` Contains the local system configuration files for the host computer.

`/home` Home directory storage for user files. Each user has a subdirectory in /home.

`/root` This is not the root (/) filesystem. It is the home directory for the root user.

`/lib` Contains shared library files that are required to boot the system.

Linux File System Hierarchy

The following provides a **summary of top-level Linux directories and their purposes** (II):

`/media` A place to mount external removable media devices such as USB thumb drives that may be connected to the host.

`/mnt` A temporary mountpoint for regular filesystems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.

`/opt` Optional files such as vendor supplied application programs should be located here.

`/sbin` System binary files. These are executables used for system administration.

`/tmp` Temporary directory, used by the operating system and many programs to store temporary files. Note that files stored here may be deleted at any time without prior notice.

`/usr` These are shareable, read-only files, including executable binaries, libraries, `man` files.

`/var` Variable data files are stored here, include like log files, database files, web server data files, email inboxes.

Linux User Permissions

Linux permissions (rights) are organized in three categories:

- User (u)
- Group (g)
- Other (o)

Each file belongs to a user and to a group.

For each category rights define what one can do:

- Read (r)
- Write (w)
- Execute (x)

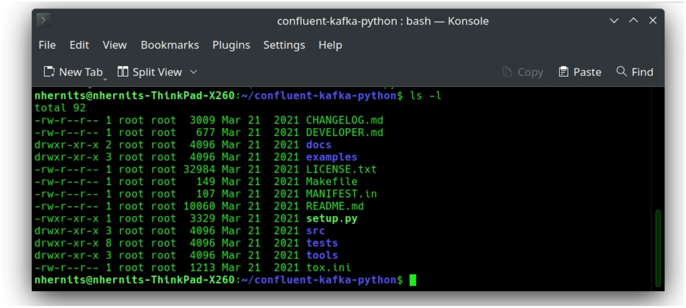
We can display permission, owners, and groups by running the command

```
# ls -l
```

which lists directory contents in long format.

Linux User Permissions

This shows the permissions for each file and directory in the first column:


confluent-kafka-python : bash — Konsole
File Edit View Bookmarks Plugins Settings Help
New Tab
Split View
Copy
Paste
Find

```
nhernits@nhernits-ThinkPad-X260:~/confluent-kafka-python$ ls -l
total 92
-rw-r--r-- 1 root root  3009 Mar 21  2021 CHANGELOG.md
-rw-r--r-- 1 root root   677 Mar 21  2021 DEVELOPER.md
drwxr-xr-x 2 root root  4096 Mar 21  2021 docs
drwxr-xr-x 3 root root  4096 Mar 21  2021 examples
-rw-r--r-- 1 root root 32984 Mar 21  2021 LICENSE.txt
-rw-r--r-- 1 root root   149 Mar 21  2021 Makefile
-rw-r--r-- 1 root root   107 Mar 21  2021 MANIFEST.in
-rw-r--r-- 1 root root 10060 Mar 21  2021 README.md
-rwxr-xr-x 1 root root  3329 Mar 21  2021 setup.py
drwxr-xr-x 3 root root  4096 Mar 21  2021 src
drwxr-xr-x 8 root root  4096 Mar 21  2021 tests
drwxr-xr-x 3 root root  4096 Mar 21  2021 tools
-rw-r--r-- 1 root root  1213 Mar 21  2021 tox.ini
nhernits@nhernits-ThinkPad-X260:~/confluent-kafka-python$
```

These are shown as `-uuugggooo`, where `u` is "user", `g` is "group", and `o` is "other." Directories start with `d`.

The standard permissions for a directory are `drwx-r-xr-x`, or 755, giving the user permission to write, and the user, group, and other permission to read and execute, or in the case of a directory, to access it.

Linux User Permissions

We can **change the rights** on the terminal or from the GUI.

The commands are as follows:

Change file owner: `chown`
Change file rights: `chmod`, e.g, to make a bash script executable:

```
chmod u+x prog.sh
```

Linux User Permissions

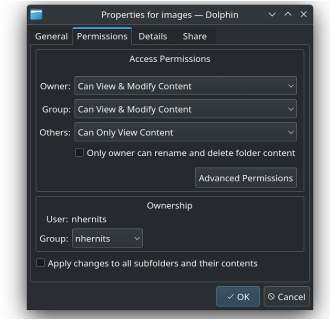
We can **change the rights** on the terminal or from the GUI.

The commands are as follows:

Change file owner: `chown`
Change file rights: `chmod`, e.g, to make a bash script executable:

```
chmod u+x prog.sh
```

This can also be accessed from the GUI:



Access Permissions
Owner: Can View & Modify Content
Group: Can View & Modify Content
Others: Can Only View Content
☐ Only owner can rename and delete folder content

Ownership
User: nhernits
Group: nhernits
☐ Apply changes to all subfolders and their contents

Linux Shell

Nowadays we have a graphical user interface (GUI), but using the command line (shell) is still common for many tasks.

Linux Shell

Nowadays we have a graphical user interface (GUI), but using the command line (shell) is still common for many tasks.

The Linux command line (often referred to as the shell, terminal, console, prompt or various other names) is a text interface to your computer.
Despite using the command line can look intimidating, once being familiar with it, it can be **very efficient and comfortable for many tasks**.

Linux Shell

Nowadays we have a graphical user interface (GUI), but using the command line (shell) is still common for many tasks.

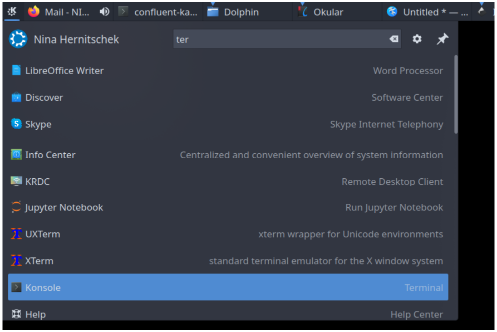
The Linux command line (often referred to as the shell, terminal, console, prompt or various other names) is a text interface to your computer.
Despite using the command line can look intimidating, once being familiar with it, it can be **very efficient and comfortable for many tasks**.

In this lecture (and tutorial session later this week) you will learn how to do so.

Linux Shell

On most Linux systems, you will find the terminal program where you also access other software you use.

For example, on an Ubuntu system you can find a launcher for the terminal by clicking on the Activities item at the top left of the screen, then typing the first few letters of `terminal`, `command`, `prompt` or `shell`.



Mail - NI... confluent-ka... Dolphin Okular Untitled * — ...

Nina Hernitschek

ter

LibreOffice Writer	Word Processor
Discover	Software Center
Skype	Skype Internet Telephony
Info Center	Centralized and convenient overview of system information
KRDC	Remote Desktop Client
Jupyter Notebook	Run Jupyter Notebook
UXTerm	xterm wrapper for Unicode environments
XTerm	standard terminal emulator for the X window system
Konsole	Terminal
Help	Help Center

Linux Shell

Let's run our first command. Type the following command, all in lower case, before pressing the Enter key:

```
pwd
```

Linux Shell

Let's run our first command. Type the following command, all in lower case, before pressing the Enter key:

```
pwd
```

You should see a directory path printed out (probably something like `/home/your_username`), then another copy of that odd bit of text, similar to the following:

```
nhernits@nhernits-ThinkPad-X260:~/confluent-kafka-python$ pwd

/home/nhernits/confluent-kafka-python
```

The Linux shell

Here is a bit more background information:

The shell runs in a terminal.

The shell is one of the primary interfaces to use the computer.

It interactively interprets commands, starts programs.

Full programs can be written for and interpreted by the shell.

The Linux shell

Here is a bit more background information:

The shell runs in a terminal.

The shell is one of the primary interfaces to use the computer.

It interactively interprets commands, starts programs.

Full programs can be written for and interpreted by the shell.

There exist different shells:

The most common shell is the Bourne Again Shell (`bash`).

In the past the (tiny) C Shell (`tcsh`) was popular.

Other shells such as the Korn Shell (`ksh`) or the Z Shell (`zsh`) have a small but active community.

The difference is important as different shells have a different syntax, and so also **shell scripts are shell-dependent**.

Linux Commands

A Linux command is a program or utility that runs on a command line. We have already seen one command: `cmd`.

Many commands have **flags** and **arguments** that can be passed to commands. In many cases they are optional.

Flags can be invoked using hyphens and double hyphens in the system. Flags can usually be combined.

An **argument or parameter** is the input you give to a command. Often, the argument is a file path. Argument execution depends on their order.

Linux Commands

Some examples:

```
ls
```

List information about the files in the current directory. Entries are sorted alphabetically.

Linux Commands

Some examples:

```
ls
```

List information about the files in the current directory. Entries are sorted alphabetically.

```
ls -lh
```

The `ls` command with two flags.
`-l`: use a long listing format
`-h`, `--human-readable`: print sizes like 1K, 234M, 2G etc.

Linux Commands

Some examples:

```
ls
```

List information about the files in the current directory. Entries are sorted alphabetically.

```
ls -lh
```

The `ls` command with two flags.
`-l`: use a long listing format
`-h`, `--human-readable`: print sizes like 1K, 234M, 2G etc.

```
python3 program.py
```

Invoking Python3.x with an argument, the name of the script to be executed.

Linux Commands

The importance of case

Be extra careful with case when typing in the command line. Typing `PWD` instead of `pwd` will produce an error, but sometimes the wrong case can result in a command appearing to run, but not doing what you expected. We'll look at case a little more on the next page but, for now, just make sure to type all the following lines in exactly the case that's shown.

Linux Commands

Most Linux command line tools include a `man` page (manual). Try to take a brief look at the pages for some of the commands you've already encountered:

```
man ls
```

There's even a `man` page for the `man` program itself:

```
man man
```

Navigating the File System

In the following we will see commands to navigate the file system.

We start by typing

```
pwd
```

`pwd` is an abbreviation of *print working directory*. All it does is print out the shell's current working directory. But what is a working directory?

The working directory is basically *where you are currently* in the file system. If you try to create new files or directories, view existing files, or even delete them, the shell will assume you are looking for them in the current working directory unless you take steps to specify otherwise.

If you're ever in any doubt, the `pwd` command will tell you exactly what the current working directory is.

Navigating the File System

You can change the working directory using the `cd` command, an abbreviation for *change directory*. Try the following:

```
cd /
pwd
```

Navigating the File System

You can change the working directory using the `cd` command, an abbreviation for *change directory*. Try the following:

```
cd /
pwd
```

Often it is necessary to go to the `home` direcory (which is an immediate subdirectory of /):

```
cd home
```

Navigating the File System

You can change the working directory using the cd command, an abbreviation for *change directory*. Try the following:

```
cd /
pwd
```

Often it is necessary to go to the home direcory (which is an immediate subdirectory of /):

```
cd home
```

Also typing cd on its own is a quick shortcut to get back to your home directory:

```
cd
```

Navigating the File System

You can change the working directory using the `cd` command, an abbreviation for *change directory*. Try the following:

```
cd /
pwd
```

Often it is necessary to go to the `home` direcory (which is an immediate subdirectory of /):

```
cd home
```

Also typing `cd` on its own is a quick shortcut to get back to your home directory:

```
cd
```

To go up to the parent directory, in this case back to /, use the special syntax of two dots `..` when changing directory:

```
cd ..
```

Navigating the File System

Most of the examples we have looked at so far use **relative paths**.

Consider trying to `cd` into the `etc` folder. If you're already in the root directory that will work fine:

```
cd /
pwd
cd etc
pwd
```

Navigating the File System

Most of the examples we have looked at so far use **relative paths**.

Consider trying to `cd` into the `etc` folder. If you're already in the root directory that will work fine:

```
cd /
pwd
cd etc
pwd
```

But what if you are in your home directory, doing the above produce an error saying `No such file or directory` before you even get to run the last `pwd`. Changing directory by specifying the directory name, or using `..` will have different effects depending on where you start from. The path only makes sense relative to your working directory.

Navigating the File System

Most of the examples we have looked at so far use **relative paths**.

Consider trying to `cd` into the `etc` folder. If you're already in the root directory that will work fine:

```
cd /
pwd
cd etc
pwd
```

But what if you are in your home directory, doing the above produce an error saying `No such file or directory` before you even get to run the last `pwd`. Changing directory by specifying the directory name, or using `..` will have different effects depending on where you start from. The path only makes sense relative to your working directory.

Absolute paths start from your home directory:

```
/home/yourusername/Documents
```

Working with Folders and Files

Knowing how to navigate between folders, we now continue with **creating folders and files**.

Despite folders and files are often created from within programs, in many cases it can be convenient or necessary to create them from the shell.

Working with Folders and Files

Knowing how to navigate between folders, we now continue with **creating folders and files**.

Despite folders and files are often created from within programs, in many cases it can be convenient or necessary to create them from the shell.

To avoid accidentally destroying any of your real files, we're going to start by creating a new directory, well away from your home folder, which will serve as a safer environment in which to experiment:

```
mkdir /tmp/tutorial
cd /tmp/tutorial
```

Working with Folders and Files

Knowing how to navigate between folders, we now continue with **creating folders and files**.

Despite folders and files are often created from within programs, in many cases it can be convenient or necessary to create them from the shell.

To avoid accidentally destroying any of your real files, we're going to start by creating a new directory, well away from your home folder, which will serve as a safer environment in which to experiment:

```
mkdir /tmp/tutorial
cd /tmp/tutorial
```

Notice the use of an absolute path, to make sure that we create the tutorial directory inside /tmp. Without the forward slash at the start the `mkdir` command would try to find a `tmp` directory inside the current working directory, then try to create a tutorial directory inside that.

Working with Folders and Files

Create, copy, delete, and move files

We can create a directory using **mkdir**:

```
mkdir /tmp/tutorial
cd /tmp/tutorial
mkdir testdir
```

Working with Folders and Files

Create, copy, delete, and move files

We can create a directory using **mkdir**:

```
mkdir /tmp/tutorial
cd /tmp/tutorial
mkdir testdir
```

We can create an empty file with `touch`:

```
touch newfile
```

Working with Folders and Files

Create, copy, delete, and move files

We can create a directory using **mkdir**:

```
mkdir /tmp/tutorial
cd /tmp/tutorial
mkdir testdir
```

We can create an empty file with `touch`:

```
touch newfile
```

We create a copy with cp, stating the source and destination:

```
cp oldfilename newfilename
```

We can move (rename) a file with `mv`:

```
mv oldfilename newfilename
```

Working with Folders and Files

Create, copy, delete, and move files

We delete files with `rm`:

```
rm newfilename
```

For deleting files, there exist flags such as `-r` for recursively deleting files and subdirectories in a directory and `-f` for force.

Working with Folders and Files

Create, copy, delete, and move files

We delete files with `rm`:

```
rm newfilename
```

For deleting files, there exist flags such as `-r` for recursively deleting files and subdirectories in a directory and `-f` for force.

Caution

Be extremely careful with those. Although `rm -r` is quick and convenient, it is also dangerous. It is safest to explicitly delete files to clear out a directory, then `cd ..` to the parent before using `rmdir` to remove it. Unlike with graphical interfaces, `rm` doesn't move files to a folder called "trash" or similar. There is often no way to recover files.

Use `-i` (interactive) to mitigate errors: It will prompt you to confirm the deletion of each file; enter `Y` to delete it, `N` to keep it, and press Ctrl C to stop the operation entirely.

Working with Folders and Files

Creating files using redirection

We have seen that certain commands, like `ls`, will lead to text output on the shell.
Instead of copy-paste this output to a text editor, in many cases it is more convenient to directly write into a file. We can do so like in the following example:

```
ls > output.txt
```

This time there is nothing printed to the screen, because the output is being redirected to a file instead.

Working with Folders and Files

Creating files using redirection

We have seen that certain commands, like `ls`, will lead to text output on the shell.
Instead of copy-paste this output to a text editor, in many cases it is more convenient to directly write into a file. We can do so like in the following example:

```
ls > output.txt
```

This time there is nothing printed to the screen, because the output is being redirected to a file instead.

Working with Folders and Files

Creating files using redirection

We have seen that certain commands, like `ls`, will lead to text output on the shell.
Instead of copy-paste this output to a text editor, in many cases it is more convenient to directly write into a file. We can do so like in the following example:

```
ls > output.txt
```

This time there is nothing printed to the screen, because the output is being redirected to a file instead.

You can open the created file `output.txt` either with a text editor, or directly on the shell using

```
cat output.txt
```

Working with Folders and Files

Creating files using redirection

Let's look at another command, `echo`:

```
echo "This is a test"
```

Working with Folders and Files

Creating files using redirection

Let's look at another command, `echo`:

```
echo "This is a test"
```

It doesn't look very useful as `echo` just prints its arguments back out again (hence the name). But combine it with a redirect, and you've got a way to easily create small test files:

```
echo "This is a test" > test_1.txt
echo "This is a second test" > test_2.txt
echo "This is a third test" > test_3.txt
```

Working with Folders and Files

Good naming practice

When you consider both case sensitivity and escaping, a good rule of thumb is to keep your file names all lower case, with only letters, numbers, underscores and hyphens. For files there's usually also a dot and a few characters on the end to indicate the type of file it is (referred to as the **file extension**). This guideline may seem restrictive, but if you end up using the command line with any frequency you'll be glad you stuck to this pattern.

Working with Folders and Files

Display, navigate and search text files

We can display the contents of a text file using `cat`. This command is actually used to concatenate files, e.g.,

```
cat file1 file2 > file3
```

Remember here what the > operator does:
Here we actually output two files and redirect the output into a third file.

To display only the first or last few lines of a file, you can use `head` or `tail`, respectively.

Working with Folders and Files

Display, navigate and search text files

We can display the contents of a text file using `cat`. This command is actually used to concatenate files, e.g.,

```
cat file1 file2 > file3
```

Remember here what the > operator does:
Here we actually output two files and redirect the output into a third file.

To display only the first or last few lines of a file, you can use `head` or `tail`, respectively.

Use `-n` to specify the number of lines, e.g.,

```
head -n 42 myfile.txt
```

Working with Folders and Files

Display, navigate and search text files

A more convenient way to inspect a text file is with `less`. It allows to go forward/backward through a file. In addition, it e.g. allows to search a string with `/somestring`, one can show line numbers with the `-N` flag.

Working with Folders and Files

Display, navigate and search text files

A more convenient way to inspect a text file is with `less`. It allows to go forward/backward through a file. In addition, it e.g. allows to search a string with `/somestring`, one can show line numbers with the `-N` flag.

More advanced ways of searching for strings in files are `grep` and `awk`. An example:

```
grep -i "upper limit" *.txt
```

Working with Folders and Files

Display, navigate and search text files

A more convenient way to inspect a text file is with `less`. It allows to go forward/backward through a file. In addition, it e.g. allows to search a string with `/somestring`, one can show line numbers with the `-N` flag.

More advanced ways of searching for strings in files are `grep` and `awk`. An example:

```
grep -i "upper limit" *.txt
```

`grep` is often combined with cat to form more complex expressions, e.g.,

```
cat [AB]*{.txt,.dat} | grep -i astronomy
```

Working with Folders and Files

Display, navigate and search text files

A more convenient way to inspect a text file is with `less`. It allows to go forward/backward through a file. In addition, it e.g. allows to search a string with `/somestring`, one can show line numbers with the `-N` flag.

More advanced ways of searching for strings in files are `grep` and `awk`. An example:

```
grep -i "upper limit" *.txt
```

`grep` is often combined with cat to form more complex expressions, e.g.,

```
cat [AB]*{.txt,.dat} | grep -i astronomy
```

Such expressions are often used as parts of **shell scripts** . We will see later on how to write them. Shell scripts are a convenient way for manipulating and reformatting large tables in text format.

Summary of the main shell commands

Some basic commands:

Change directory: `cd`
Create a directory: `mkdir`
Remove a directory: `rmdir`
List files: `ls`
Copy files: `cp`
Move files: `mv`
Delete files: `rm`
Find a file: `find`
Display a text file: `cat`
Navigate in a text file: `less`
First lines of a file: `head`
Last lines of a file: `tail`
Find in a text file: `grep`

For all shell commands: look at the documentation

- For a brief summary of options: `command --help`
- For a more complete manual: `man command`

Summary of the main shell commands

Miscellaneous typical application for commands (I):

If you want to sort anything, use `sort`. This sorts a file by its 2nd column:

```
cat mydata.dat | sort -k 2
```

Summary of the main shell commands

Miscellaneous typical application for commands (I):

If you want to sort anything, use `sort`. This sorts a file by its 2nd column:

```
cat mydata.dat | sort -k 2
```

To find duplicates for a list, use `uniq`. This outputs only the unique lines:

```
cat mydata.dat | uniq -u
```

This outputs only the duplicated lines:

```
cat mydata.dat | uniq -d
```

Summary of the main shell commands

Miscellaneous typical application for commands (I):

If you want to sort anything, use `sort`. This sorts a file by its 2nd column:

```
cat mydata.dat | sort -k 2
```

To find duplicates for a list, use `uniq`. This outputs only the unique lines:

```
cat mydata.dat | uniq -u
```

This outputs only the duplicated lines:

```
cat mydata.dat | uniq -d
```

If you want to count the number of characters/words/lines, use `wc`:

```
cat mydata.dat | wc -l
```

Summary of the main shell commands

Miscellaneous typical application for commands (II):

If you want to extract a column from a file, use `awk`. Here we output colums 2, 3, 5 from a comma-separated file (`-F,`):

```
cat mydata.dat | awk -F, '{print $2, $3, $5}'
```

Summary of the main shell commands

Miscellaneous typical application for commands (II):

If you want to extract a column from a file, use `awk`. Here we output colums 2, 3, 5 from a comma-separated file (`-F,`):

```
cat mydata.dat | awk -F, '{print $2, $3, $5}'
```

`awk` is actually a powerful scripting language. If you regularly work with (large) text files such as log files and tables, it is worth checking it out.

Managing Processes

In addition to manipulating files and folders, one of the most important tasks carried out from the shell is **process management**, which involves controlling the various programs and services running on the system.

What is a Process in Linux?
A Linux process is a program which is currently under execution. Whenever we send a command to the Linux system, it initiates a new process. More than one process can also be initiated for a single program such as multiple windows or terminal.

Managing Processes

We can divide the Linux processes in two categories:

Foreground Processes: These processes are real time and run on the system screen. These processes are also known as interactive processes. Foreground processes can be started using the GUI or terminal. If we start a certain foreground process from the terminal, then we have to wait for the terminal until the process begins. Examples: Office programs, astronomical software such as topcat.

Managing Processes

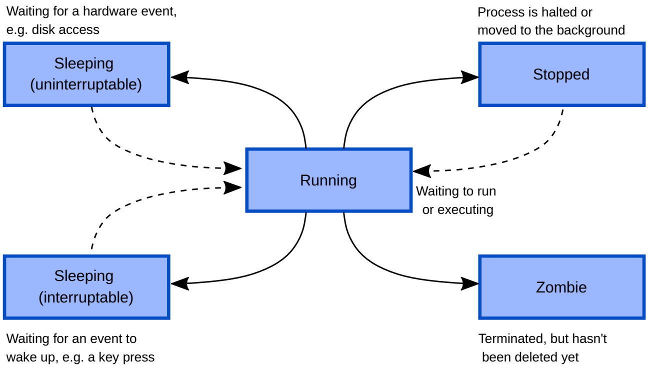
We can divide the Linux processes in two categories:

Foreground Processes: These processes are real time and run on the system screen. These processes are also known as interactive processes. Foreground processes can be started using the GUI or terminal. If we start a certain foreground process from the terminal, then we have to wait for the terminal until the process begins. Examples: Office programs, astronomical software such as topcat.

Background Processes: These processes run in background and do not need user interference or input. These processes are also known as non-interactive processes. Examples: backup programs.

Managing Processes

Processes in Linux can move between different states:


Waiting for a hardware event, e.g. disk access
Sleeping (uninterruptable)
Process is halted or moved to the background
Stopped
Running
Waiting to run or executing
Sleeping (interruptable)
Zombie
Waiting for an event to wake up, e.g. a key press
Terminated, but hasn't been deleted yet

Managing Processes

List running processes using `top`

The `top` command displays real-time information about running processes, such as resource usage and CPU time. Processes are displayed in order of their resource usage.

To track the running processes run `top`

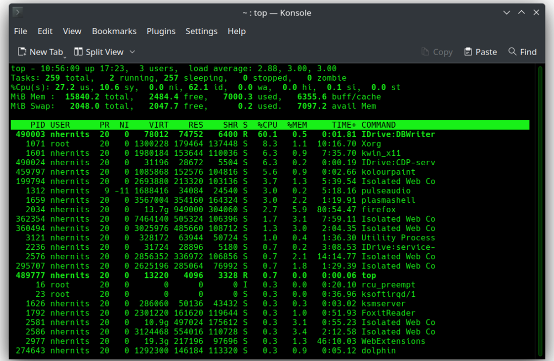
Managing Processes

List running processes using `top`

The `top` command displays real-time information about running processes, such as resource usage and CPU time. Processes are displayed in order of their resource usage.

To track the running processes run `top`

A list of processes that are running on the system is displayed:



```
top - 10:56:09 up 17:23,  3 users,  load average: 2.88, 3.00, 3.00
Tasks: 259 total,   2 running, 257 sleeping,   0 stopped,   0 zombie
%Cpu(s): 27.2 us, 10.6 sy,  0.0 ni, 62.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  15840.2 total,   2484.4 free,   7000.3 used,   6355.6 buff/cache
MiB Swap:   2048.0 total,   2047.7 free,      0.2 used.   7097.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 490003 nhernits  20   0   78012  74752   6400 R  60.1   0.5   0:01.81 IDrive:DBWriter
   1071 root      20   0 1300228 179464 137448 S   8.3   1.1  10:16.70 Xorg
   1601 nhernits  20   0 1980184 153644 110036 S   6.3   0.9   7:35.70 kwin_x11
 490024 nhernits  20   0   31196  28672   5504 S   6.3   0.2   0:00.19 IDrive:CDP-serv
 459797 nhernits  20   0 1085868 152576 104816 S   5.6   0.9   0:02.66 kolourpaint
 199794 nhernits  20   0 2693880 213320 103136 S   3.7   1.3   5:39.54 Isolated Web Co
   1312 nhernits   9 -11 1688416  34084  24540 S   3.0   0.2   5:18.16 pulseaudio
   1659 nhernits  20   0 3567004 354160 164324 S   3.0   2.2   1:19.91 plasmashell
   2034 nhernits  20   0   13.7g 949000 304060 S   2.7   5.9  80:54.47 firefox
 362354 nhernits  20   0 7464140 505324 106396 S   1.7   3.1   7:59.11 Isolated Web Co
 360494 nhernits  20   0 3025976 485660 108712 S   1.3   3.0   2:04.35 Isolated Web Co
   3121 nhernits  20   0  328172  63944  50724 S   1.0   0.4   1:36.30 Utility Process
   2236 nhernits  20   0   31724  28896   5180 S   0.7   0.2   3:08.53 IDrive:service-
   2576 nhernits  20   0 2856352 336972 106856 S   0.7   2.1  14:14.77 Isolated Web Co
 295707 nhernits  20   0 2625196 285064  76992 S   0.7   1.8   1:29.39 Isolated Web Co
 489777 nhernits  20   0   13220   4096   3328 R   0.7   0.0   0:00.06 top
     16 root      20   0       0      0      0 I   0.3   0.0   0:20.10 rcu_preempt
     23 root      20   0       0      0      0 S   0.3   0.0   0:36.96 ksoftirqd/1
   1626 nhernits  20   0  286060  50136  43432 S   0.3   0.3   0:03.02 ksmserver
   1792 nhernits  20   0 2301220 161620 119644 S   0.3   1.0   0:51.93 FoxitReader
   2581 nhernits  20   0   10.9g 497024 175612 S   0.3   3.1   0:55.23 Isolated Web Co
   2586 nhernits  20   0 3124468 554016 110728 S   0.3   3.4   2:12.58 Isolated Web Co
   2977 nhernits  20   0   19.3g 217196  97696 S   0.3   1.3  46:10.03 WebExtensions
 274643 nhernits  20   0 1292300 146184 113320 S   0.3   0.9   0:05.12 dolphin
```

Managing Processes

List running processes using `top`

Below is the detail of all fields shown by `top`:

`PID`: Every process is assigned a unique identifier called the PID.

`User`: Process owner username (system name).

`PR`: This indicates the priority given to a process during scheduling.

`%NI`: This field displays a nice value.

`VIRT`: Virtual memory used by a certain process.

`RES`: Physical memory used by a certain process.

`SHR`: Shared memory with other processes.

`S`: This field displays the state of the process, which can be
`D` = uninterrupt. sleep, `R` = running, `S` = sleeping, `T` = stopped, `Z` = zombie

`%CPU`: CPU percentage used by a certain process.

`%MEM`: It gives us the percentage of RAM a process is utilizing.

`TIME+`: This gives information about the total CPU time used by a process.

`Command`: Command used to activate the process.

Managing Processes

List running processes using `top`

Below is the detail of all fields shown by `top`:

`PID`: Every process is assigned a unique identifier called the PID.

`User`: Process owner username (system name).

`PR`: This indicates the priority given to a process during scheduling.

`%NI`: This field displays a nice value.

`VIRT`: Virtual memory used by a certain process.

`RES`: Physical memory used by a certain process.

`SHR`: Shared memory with other processes.

`S`: This field displays the state of the process, which can be
`D` = uninterrupt. sleep, `R` = running, `S` = sleeping, `T` = stopped, `Z` = zombie

`%CPU`: CPU percentage used by a certain process.

`%MEM`: It gives us the percentage of RAM a process is utilizing.

`TIME+`: This gives information about the total CPU time used by a process.

`Command`: Command used to activate the process.

You can exit **top** by pressing Q .

Managing Processes

Display Process Status using `ps`

The `ps` command in Linux stands for *Process Status* and is used to display information about the running processes. It provides us with the current state of the system's processes. Unlike the `top` command, the information displayed by `ps` is not updated in real-time.

Run the `ps` command to get the information of current running process:

```
ps
```

The terminology is as follows:

`PID` (Process ID): A unique numerical identifier given by system to a process.

`TTY` (Terminal Type): The type of terminal or console associated with the process.

`TIME` (Total Time): The amount of time, typically measured in CPU seconds, that the process has been running since it started.

`CMD` (Command): The name of the command or executable that starts a process.

There is a variety of flags. Use `ps -u` to get more info about a system processes:

Managing Processes

Stop a Process using `kill`

The `kill` command sends a signal to the specified process, causing it to stop executing and exit.

By default, the kill command sends a `SIGTERM [-15]` signal, which completely stops and cleans the process before exiting. However, it is also possible to send a `SIGKILL[-9]` signal, which immediately terminates the process without allowing it to clean up.

Managing Processes

Stop a Process using `kill`

The `kill` command sends a signal to the specified process, causing it to stop executing and exit.

By default, the kill command sends a `SIGTERM [-15]` signal, which completely stops and cleans the process before exiting. However, it is also possible to send a `SIGKILL[-9]` signal, which immediately terminates the process without allowing it to clean up.

To first know the process ID we use following command:

```
pidof [process name]
```

For killing a certain process with the help of its process id `[pid]` use:

```
kill [pid]
```

Or we can also send a `SIGKILL[-9]` signal:

```
kill -9 [pid]
```

Managing Processes

Priorize Processes using `nice`

Running too many processes can slow down the performance of high-priority processes.

The `nice` command sets a process priority.
Niceness values range from -20 to 19 and lower the values means more priority will be given to that process. By default, processes have a niceness value of 0.

To change a process's niceness value run below command:

```
nice -n [nice value] [process name]
```

The current value can be checked using the `top` command.

Managing Processes

Priorize Processes using `nice`

Running too many processes can slow down the performance of high-priority processes.

The `nice` command sets a process priority.
Niceness values range from -20 to 19 and lower the values means more priority will be given to that process. By default, processes have a niceness value of 0.

To change a process's niceness value run below command:

```
nice -n [nice value] [process name]
```

The current value can be checked using the `top` command.

Most likely you will encounter setting the niceness value when working on a shared system, such as a **computer cluster**. Often users are adviced by the admins to set a lower priority to their processes.

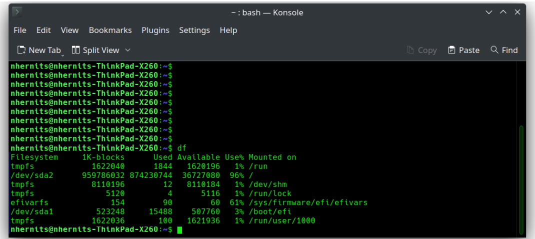
Managing Processes

Check Free Disk Space using `df`

The commands shown before are related to process management. Often, however, we also want to take a look at how the system behaves in general.

The `df` command is used to check free disk space available. It displays information about the total size of the file system and total space used. It also gives space available and used space percentage.

```
df
```



```
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
tmpfs            1622040      1844   1620196   1% /run
/dev/sda2      959786032 874230744  36727080  96% /
tmpfs            8110196        12   8110184   1% /dev/shm
tmpfs               5120         4      5116   1% /run/lock
efivarfs             154        90        60  61% /sys/firmware/efi/efivars
/dev/sda1         523248     15488    507760   3% /boot/efi
tmpfs            1622036       100   1621936   1% /run/user/1000
nhernits@nhernits-ThinkPad-X260:~$
```

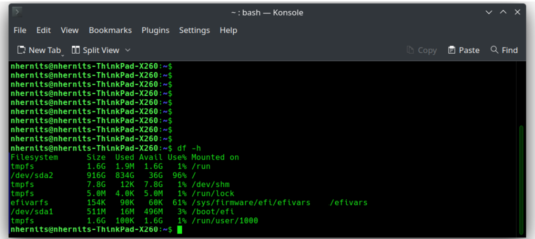
Managing Processes

Check Free Disk Space using `df`

The commands shown before are related to process management. Often, however, we also want to take a look at how the system behaves in general.

The information can also be shown in more simplified way:

```
df -h
```



```
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           1.6G  1.9M  1.6G   1% /run
/dev/sda2       916G  834G   36G  96% /
tmpfs           7.8G   12K  7.8G   1% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
efivarfs        154K   90K   60K  61% /sys/firmware/efi/efivars    /efivars
/dev/sda1       511M   16M  496M   3% /boot/efi
tmpfs           1.6G  100K  1.6G   1% /run/user/1000
nhernits@nhernits-ThinkPad-X260:~$
```

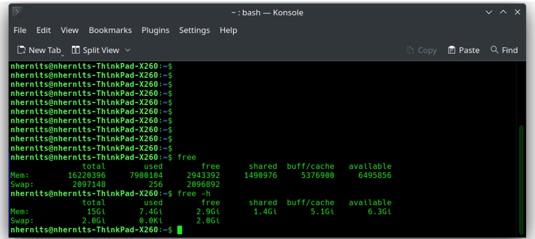
Check Memory Usage using `free`

The `free` command is used to check the memory usage and free space available on a system.

```
free
```

The information can also be shown in more simplified way:

```
free -h
```



```
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$
nhernits@nhernits-ThinkPad-X260:~$ free
               total        used        free      shared  buff/cache   available
Mem:        16220396     7900104     2943392     1490976     5376900     6495856
Swap:        2097148         256     2096892
nhernits@nhernits-ThinkPad-X260:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi       7.4Gi       2.9Gi       1.4Gi       5.1Gi       6.3Gi
Swap:          2.0Gi       0.0Ki       2.0Gi
nhernits@nhernits-ThinkPad-X260:~$
```

Summary of Process Management Commands

list running processes: `ps` (more commonly: `ps -aux`)

interactive list of processes: `top`

kill process by number process number: `kill`

kill process by name: `killall`

change priority of already running process: `renice`

kill foreground process: Ctrl c pause foreground process: Ctrl z

continue paused process and move to background: `bg`

continue paused process in foreground: `fg`

list all processes running in the background of the terminal: `jobs`

memory usage: `free`

disk usage: `df`

disk usage of current directory: `du`

Writing Shell Scripts

So far, we have used the shell **interactive** by typing commands.

In many cases it is useful to create a **shell script**:
A shell script is a computer program designed to be run by a Unix/ Linux shell. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. Commands used in a shell script are basically the same as on the command line.

How to write shell scripts will be a topic of this week's **tutorial**. Here, just an example will be shown as introduction to shell scripts.

Writing Shell Scripts

Create a file named `testscript.sh` with the following content:

```
##!/bin/bash
# This is a comment!
echo Hello World        # This is a comment, too!
```

This will be our first shell script.

The **first line** tells Unix that the file is to be executed by `/bin/bash`. Even if you are using `csh`, `ksh`, or anything else as your interactive shell (on the terminal), that what follows should be interpreted by the Bourne shell.

The **second line** begins with a special symbol: `#`. This marks the line as a comment, and it is ignored completely by the shell. The only exception is when the first line of the file starts with `#!` as ours does.

The **third line** runs a command: `echo`, with two arguments, where the first is `Hello`, the second is **World**. Note that `echo` will automatically put a single space between its parameters.

Writing Shell Scripts

In order to execute this script, we have to make the file **executable** by changing the permissions.

We do so by using the `chmod` command.

Afterwards, we can execute the script with `./` followed by the name of the script. Your screen should then look like this:

```
$ chmod 755 testscript.sh
$ ./testscript.sh
Hello World
```

Summary

We have seen how to use shell commands to create and manipulate folders and files, to manage processes and to logon to remote computers.

In this week's **tutorial session** we will see more on how to write shell scripts, a task you will encounter often in order to process large (astronomical) data files and to start programs.

An Outlook: Astronomical Data Sources

Next week we will continue with how to access **Astronomical Survey Data**.