The Linux Terminal: A Crash Introduction

The Linux Terminal: A Crash Introduction

Revision 1-2019

Typographical conventions

- Text in `Courier new` font indicates Linux commands that should be typed literally by the user in the terminal.
- Text in **`Courier new bold`** font indicates directories.
- Text in *`Courier new italic`* font indicates human readable files or ascii files.
- Text in **Arial bold font** indicates program elements such as variables, function names, classes, statements and so on. It also indicate environment variables, and keywords. They also highlight important information.
- Text in Arial underline in blue font indicates URLs and email addresses.
- This icon indicates a warning or a caution ⚠
- This icon indicates a tip, suggestion, or a general note 👆
- This symbol indicates that a Linux command should be typed literally by the user in the terminal $>
- This symbol indicates that you must press the enter key ↵

Typographical conventions

- Large code listing, ascii files listing, and screen outputs can be written in a square box, as follows:

```
1     #include <iostream>
2     using namespace std;
3
4     // main() is where program execution begins.  It is the main function.
5     // Every program in c++ must have this main function declared
6
7     int main ()
8     {
9           cout << "Hello world";                    //prints Hello world
10          return 0;                                 //returns nothing
11    }
```

- To improve readability, the text might be colored.
- The font can be `Courier new` or **Arial bold**.
- And when required, the line number will be shown.

On the training material

- In the directory **`101LINUX`** of the training material, you will find the files that we will use during this training.
- Take your time and try to understand the concepts implemented in these examples.
- Also, try to get familiar with the syntax.

The Linux Terminal: A Crash Introduction

- Before continuing, we want to remind you that this is an introductory Linux terminal (or the command line) course.
- Our goal is to help you familiarize with the basic concepts of the Linux operating system and the Linux shell.
- So keep calm, try to follow all the examples, and ask questions.

What is Linux?

- It is a UNIX-like operating system (OS).

So, now you might be wondering what is UNIX?

- UNIX is a family of multitasking, multiuser, stable, and portable computer operating systems that derive from the original AT&T UNIX, developed in the 1970s at the Bell Labs.
- You will find UNIX/Linux in laptops, desktop PCs, servers, and super computers.
- UNIX/Linux systems are made of three components; the kernel, the shell, and the programs.
  - The kernel is the hub of the OS. It allocates time and resources to programs and handles scheduling and system management (files, processes, devices, networking, memory and so on).
  - The shell (or the terminal), acts as an interface between the user and the kernel.
  - Programs are the different applications that the user uses. Most of the programs in Linux are Open Source and free, but you can also find commercial ones. Some programs may have a GUI interface or they can only be accessed via the terminal.

The Linux Terminal: A Crash Introduction

Some Linux facts

- Originally developed by Linus Torvalds.
- The development started in 1991.
- Supported by a world wide community (volunteers and commercial entities).
- Free and Open Source (the kernel).
- A wide range of Linux applications are free and Open Source.
- It can be installed in a great variety of hardware (laptops, desktop PCs, servers, super computers, mobile phones, embedded microchips, and so on).
- Linux is virus free and extremely secure.
- It is stable, fast-performing, and highly configurable.
- It is the ideal OS for developing applications. It comes with many compilers, interpreters, and libraries.
- Most modern Linux versions come with a GUI (similar to Windows or MacOSX), which provides an easy way to interact with the OS.
- Knowledge of the terminal is required for operations which aren't covered by a program with a GUI, or for when there is no graphical interface available.

Linux distributions

- There are many Linux distributions available (free and commercial), just to name a few:
    - Red Hat.
    - CentOS.
    - Fedora.
    - OpenSUSE.
    - Slackware.
    - Debian.
    - Ubuntu.
    - Mint.
    - Arch.

- Finding the right version that fit your needs requires some experimentation.
- **During this course we will use OpenSUSE.**

What is Open Source Software?

- Open Source refers to something people can modify and share because its design is publicly accessible.
- Open Source Software is software with source code that anyone can inspect, modify, and enhance
- Open Source Software is usually developed as a public collaboration and made freely available.
- There are many Open Source Software licenses, just to name a few:
    - GNU General Public License (GPL).
    - Apache License.
    - BSD License.
    - Creative Commons.
    - European Commission License (EUPL).
    - MIT License.

GNU General Public License (GPL)

- Under the GNU General Public License (GPL), you have legal permission to copy, distribute and/or modify the source code.
- Software that includes source code licensed under the GPL inherits the GPL license.
- If compiled binaries of GPL software are redistributed (free or for a fee), the source code must be made available by the distributor.
- The license is designed to offer freedom, in particular it does not force users of the software to make modifications or developments publicly available. That means that GPL software can be used as the basis of in-house, proprietary software.
- When we speak of free software, we are referring to freedom, not price.
- The General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

The Linux Terminal: A Crash Introduction

What is the Linux terminal or shell?

- When we talk about the Linux terminal, we talk about a piece of software that emulates a physical terminal VT220 or its predecessor the VT102.

DEC VT220 connected to the serial port of a modern computer.
https://en.wikipedia.org/wiki/VT220#/media/File:VT220_Irssi.jpg

- The terminal emulator or the shell acts as an interface between the user and the OS.
- The shell, is a program that reads commands in the terminal (from the keyboard) and send results back to the terminal (to the screen).
- It acts as a command line interpreter or CLI.

The Linux Terminal: A Crash Introduction

What shell are we going to use?

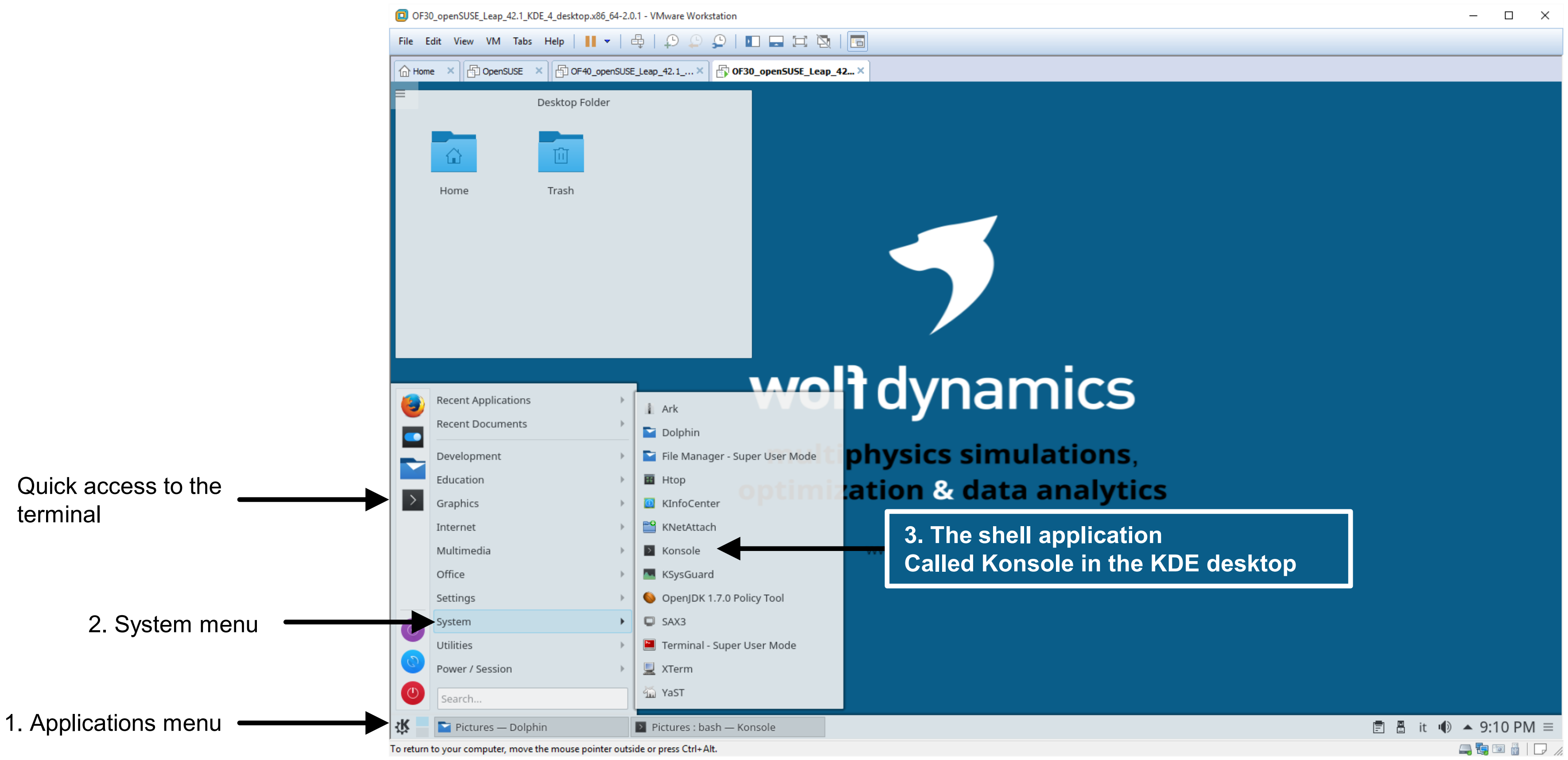
- There are many command shells, just to name a few,
    - Bourne shell (sh)
    - Korn shell (ksh)
    - C shell (csh)
    - Tee shell (tcsh)
    - Bourne-again shell (bash)

- In this course, we will use the **bash** shell.
- The bash shell is also a programming language, programs written in bash shell are called scripts.
- Linux has many scripts, and you can program your own scripts.
- Always remember, after inserting a command in the shell you will need to press the enter key

The Linux Terminal: A Crash Introduction

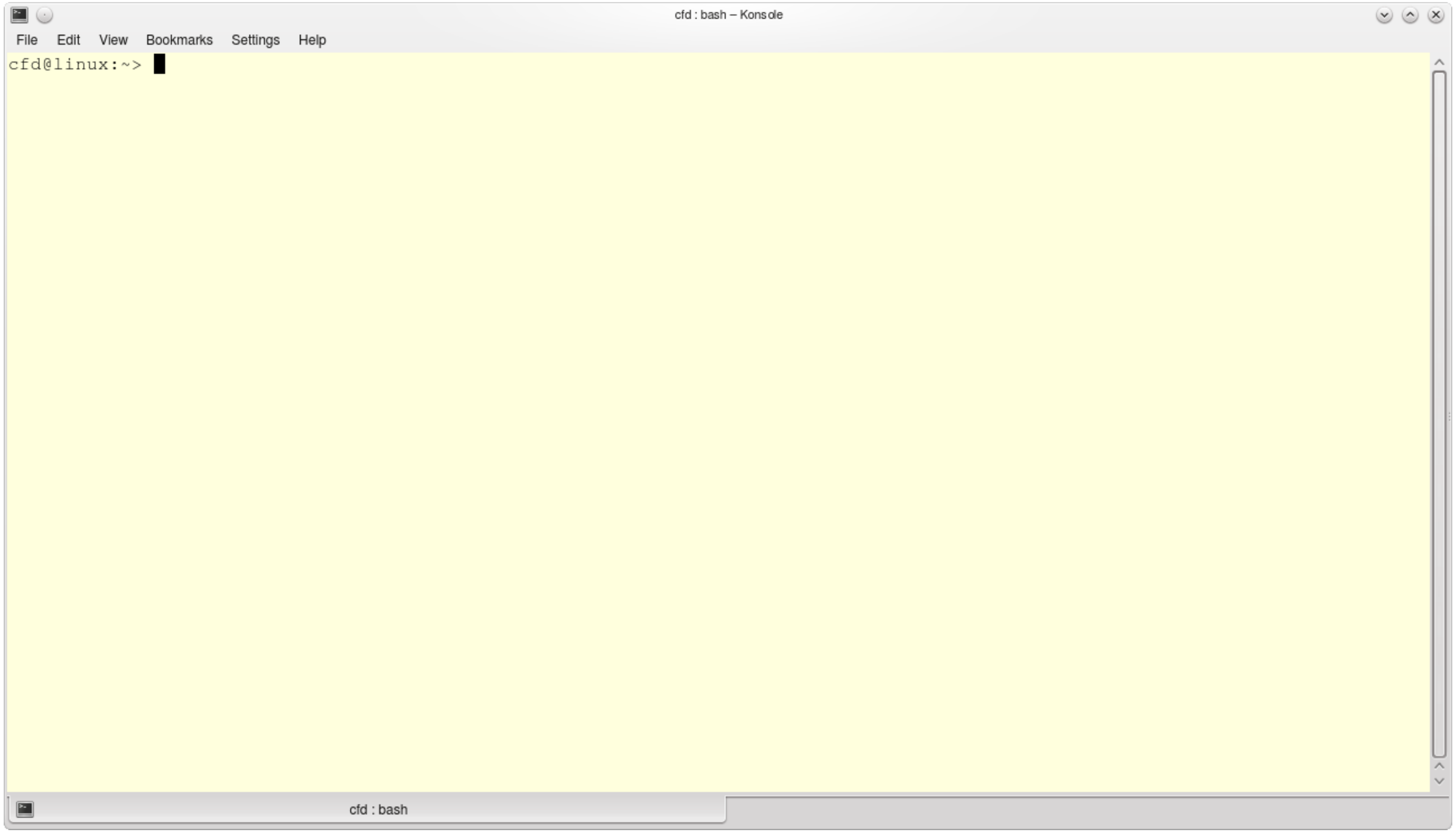
Where is the terminal in my Linux distribution?

- In most Linux distributions, finding the shell is pretty easy. Just look for the terminal or console application in the applications menu or desktop.
- If you are using OpenSUSE 42.1, you will find the shell in the system applications menu, as illustrated in the figure below,


OF30_openSUSE_Leap_42.1_KDE_4_desktop.x86_64-2.0.1 - VMware Workstation
File Edit View VM Tabs Help
Home
OpenSUSE
OF40_openSUSE_Leap_42.1_...
OF30_openSUSE_Leap_42...
Desktop Folder
Home
Trash
wolf dynamics
physics simulations,
optimization & data analytics
Recent Applications
Recent Documents
Development
Education
Graphics
Internet
Multimedia
Office
Settings
System
Utilities
Power / Session
Search...
Ark
Dolphin
File Manager - Super User Mode
Htop
KInfoCenter
KNetAttach
Konsole
KSysGuard
OpenJDK 1.7.0 Policy Tool
SAX3
Terminal - Super User Mode
XTerm
YaST
Pictures — Dolphin
Pictures : bash — Konsole
it
9:10 PM
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.
Quick access to the terminal
2. System menu
1. Applications menu
3. The shell application
Called Konsole in the KDE desktop

The Linux Terminal: A Crash Introduction

You are in Linux, you see this nice window that everybody calls terminal. Now what?

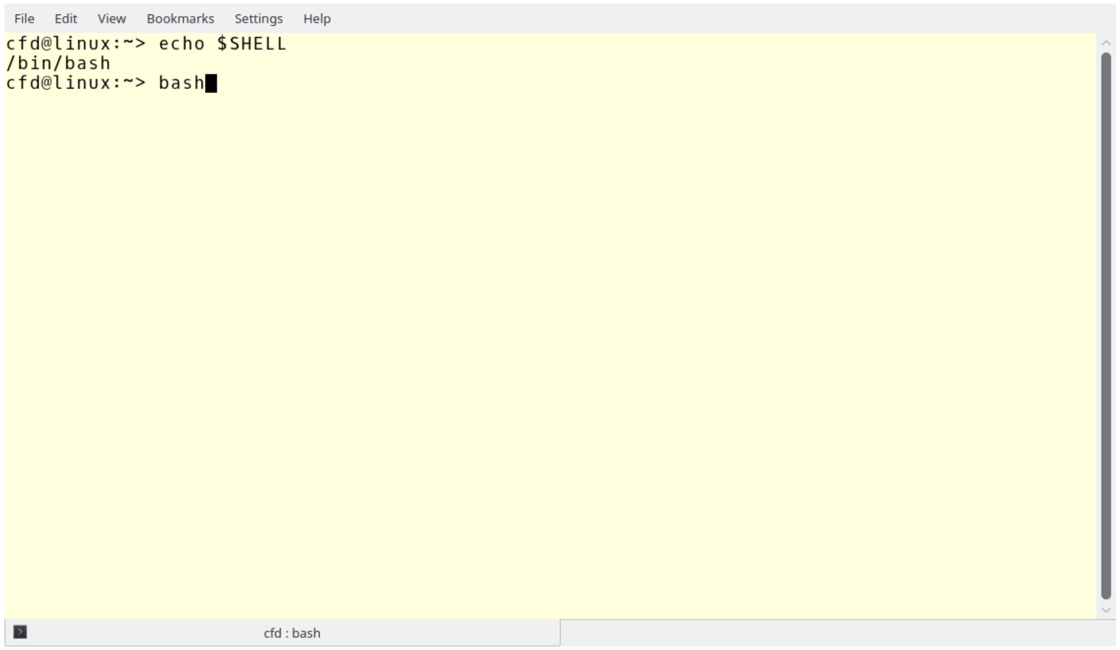


```
cfd@linux:~>
```

How do I know what shell am I using?

- To know what shell you are using, type in the terminal:
    - `$> echo $SHELL ↵`
- If the output is **/bin/bash**, you are using bash shell.
- If you are not using bash shell, type in the terminal
    - `$> bash ↵`

  to start using the bash shell.
- At this point, we are ready to start exploring the Linux terminal.



```
cfd@linux:~> echo $SHELL
/bin/bash
cfd@linux:~> bash
```

The Linux Terminal: A Crash Introduction

Basic shell interaction

- When you open the shell, it is waiting for you to do something.
- Let's use the command `date` to show how to interact with the shell. This command will display on the screen the system's date and time. Type in the terminal,
  - `$> date ↵`
- Commands can take arguments, for example,
  - `$> date +%Y ↵`

  will display the year.
- If you need help on how to use a command, you can type `command --help` or `man command`. If the help information exist, you will see the information on the terminal. For example,
  - `$> date --help ↵`
  - `$> man date ↵`
- Most of the times the command `man` gives more information than the argument `--help`.
- Type in the terminal,
  - `$> date +%s ↵`

  Use help to find the meaning of the output?

The Linux Terminal: A Crash Introduction

Basic shell interaction

- Hereafter, we list a few commands that we will use during this training:
    - `ls`
    - `cd`
    - `pwd`
    - `rm`
    - `cp`
    - `mv`
    - `df`
    - `chgrp`
    - `ps`
    - `bg`
    - `ln`
- Commands can give you very different outputs depending on the arguments you use.
- Commands in UNIX/Linux do one single thing. However, we can combine commands together to do more complex things.
- When we interact with the command line, we type commands and we get results back. As simple as that.

The Linux Terminal: A Crash Introduction

Getting around the directory structure

- Hereafter, we will talk about the Linux directory structure and how to get around it.
- We will introduce the following commands:
    - `pwd`
    - `ls`
    - `cd`
    - `mkdir`
    - `rmdir`
    - `tree`
    - `zypper`
    - `su`
    - `whoami`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Getting around the directory structure

- To know in what directory we are in, use the command `pwd` (print working directory).
  - `$> pwd ↵`
- The output of the `pwd` command should be something like this:
  - `$> /home/user_name`
- By default, when you open a new terminal you are in your home directory. Your home directory, is where you can put your files and programs.
- If you want to list the files inside the current directory use the command `ls` (list).
  - `$> ls ↵`
- To list the files inside the directory **`/usr`** using long listing format, type in the terminal,
  - `$> ls -al /usr ↵`
- To list all files (including hidden files) inside the directory **`/tmp`** using long listing format, type in the terminal,
  - `$> ls -al /tmp ↵`
- In the output of the command `ls –al`, the directory `.` (dot) refers to the current directory, and the directory `..` (dot dot) refers to the parent directory.
- For example, if you want to list the files in the current directory, use the command `ls .`; and if you want to list the files in the parent directory, use the command `ls ..`

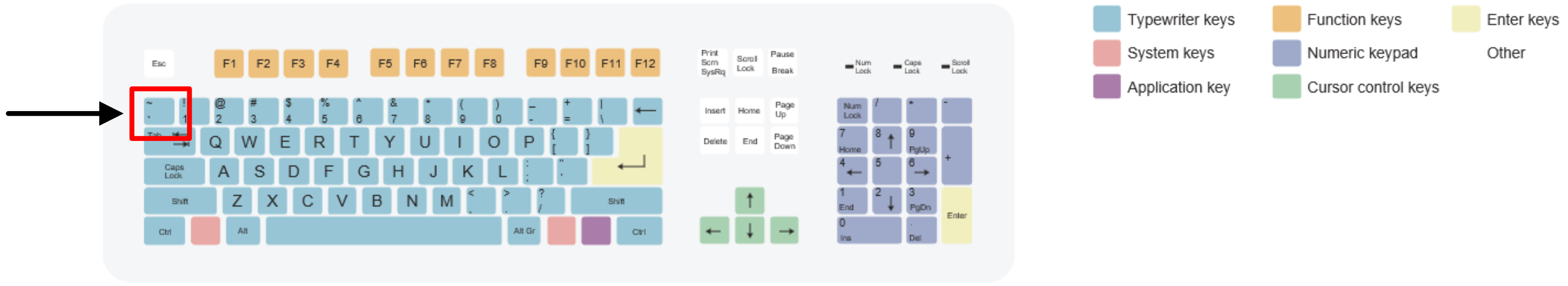
Getting around the directory structure

- If you want to go to a different directory, use the command `cd` (change directory).
    - `$> cd Desktop ↲`
    - `$> cd /usr/local/bin ↲`
- If you want to go back one directory, use `cd` with `..` (dot dot).
    - `$> cd .. ↲`
- If you want to go back many directories, use as many levels of `../` (dot dot forward slash),
    - `$> cd ../../ ↲`
- If you type `cd`, you will go to your home directory no matter where you are in the directory structure.
    - `$> cd ↲`
- To go to your home directory, you can also use the character `~` (tilde) as follows,
    - `$> cd ~ ↲`
- The character `~` (tilde) is a shortcut for your home directory and it can be used with any command.
- You will need to figure out where the character `~` (tilde) is located in your keyboard.

The Linux Terminal: A Crash Introduction

Getting around the directory structure

- If you are using an US QWERTY keyboard, you will find the character ~ (tilde) in the upper left corner.



A 104-key US keyboard layout, largely standardized since the IBM Personal System/2.
https://commons.wikimedia.org/wiki/File:Qwerty.svg

- In Linux, you can work with absolute and relative path. The absolute path is relative to the root directory (`/`) and the relative path is relative to the directory you are in (`.`).
- To go to the user's Documents directory using absolute path,
  - `$> cd /home/user_name/Documents ↵`
- To go to the user's Documents directory using relative path and assuming you are located in your home directory,
  - `$> cd Documents ↵`
- or you can type,
  - `$> cd ./Documents ↵`

Getting around the directory structure

- Remember, to change to a different directory it must exist. If it does not exist, you must create it as explained below.
- If you want to create a new directory, you can use the command `mkdir` (make directory), followed by the name of the new directory.
  - `$> mkdir dirname ↵`
- To go to the newly created directory, type in the terminal,
  - `$> cd dirname ↵`
- Linux is case sensitive, so **dirname** and **Dirname** are different directories.
- Also, when creating directories and files, try to avoid using spaces and unusual characters (?, !, £, $, %, &, ^, -, +, *).
- Underscore (_), is considered a normal character and it should be used as a substitute of a blank space, *e.g.*, **dir_name**.
- But if you really want to use blank spaces or unusual characters, you will need to use the backslash (\) followed by the space or character.
  - `$> mkdir dir\ name ↵`
  - `$> mkdir dir\ name\! ↵`

The Linux Terminal: A Crash Introduction

Getting around the directory structure

- To erase a directory, you can use the command `rmdir` (remove directory), followed by the name of the directory.
    - `$> rmdir dirname ↵`
    - `$> rmdir dir_name ↵`
    - `$> rmdir dir\ name\! ↵`
- When using spaces and unusual characters, you can also use single quotes (‘ ’) or double quotes(“ ”).
    - `$> mkdir ‘dir name’ ↵`
    - `$> rmdir “dir name !” ↵`
- To list the directory structure, you can use the command tree.
    - `$> tree –L 2 –d –C ↵`

This will display the current directory structure only two levels deep, it will show you only the directories (no files included), and it will colorize the output.

- In some Linux distributions, this command is not installed by default.
- We will discuss how to install applications in the next slide.

Installing applications from the terminal

- If you are using OpenSUSE, you can install applications from the command line using the command `zypper`.
    - `$> zypper install tree ↵`
- If you are working as a regular user (a user with no administrative rights), you will get a message similar to this one:
    - **Root privileges are required for installing or uninstalling packages.**

  This is telling you that to install/uninstall programs you need to have administrative privileges.
- To get administrative privileges, you need to become the root user (or super user). To become the root user you need to know the root password. We will talk about this in the next slide.
- In a similar way, to uninstall applications you can proceed as follows,
    - `$> zypper remove tree ↵`
- To search for applications,
    - `$> zypper search tree ↵`
- Take your time and read the help on how to use `zypper`.

Becoming the root user or super user

- To become the root user, use the `su` command (substitute user).
    - `$> su ↵`

  The shell will ask you for the root password.
- As the root user, you have absolute power over the system, so be careful.
- To know the user name, use the command `whoami`.
    - `$> whoami ↵`
- If the output is **root**, it means that currently your are the root user.
- To go back to the previous user after becoming the root user (or any other users), type in the terminal,
    - `$> exit ↵`
- You can also use the command `exit` to close the terminal window.

The Linux Terminal: A Crash Introduction

Linux typical directory structure

- The UNIX/Linux filesystem is laid out as a hierarchical tree structure which is anchored at a special top-level directory know as the root, designated by a forward slash (/).
- A typical UNIX/Linux filesystem is laid out as follows (not all directories are shown),

`/`	Root directory
`— /bin`	In this directory you will find essential low-level system utilities
`— /dev`	Hardware devices
`— /etc`	UNIX/Linux system configuration files
`— /home`	User home directories containing personal file space for each user
`    — /username1`	
`    — /username2`	
`— /lib`	Program libraries for low level system utilities
`— /proc`	Interface to the kernel
`— /sbin`	Superuser system utilities
`— /tmp`	Temporary file storage space
`— /usr`	
`    — /bin`	Higher level system utilities and applications
`    — /lib`	Program libraries for higher level programs

Linux typical directory structure

- As a regular user you only have write permission in your home directory.
- You can explore the system directory structure, as you have reading permission for most of the directory structure.
- You don’t have write and read permission to other users’ home directories or the root’s home.
- If you know the root password, you can become the super user at any time, and do whatever you like.
- The root or super user has read and write permission over the whole directory structure.
- To list the root directory structure, type in the terminal.
    - `$> cd / ↵`
    - `$> tree –L 1 ↵`

Basic file management and file inspection

- Hereafter, we will talk about file management and file inspection (displaying file content).
- We will introduce the following commands:
  - `cat`
  - `touch`
  - `more`
  - `head`
  - `tail`
  - `less`
  - `rm`
  - `cp`
  - `mv`
  - `file`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Basic file management and file inspection

- First, let’s create a new directory in your home.
    - `$> cd ↵`
    - `$> mkdir task ↵`
    - `$> cd task`
    - `$> mkdir dir1 ↵`
- To create, combine and/or display a file, we can use the command `cat`.
- Let’s use `cat` to create an empty file and write some text in it,
    - `$> cat > file1.txt ↵`

  You will notice that the shell is waiting for your input, write something in the terminal. Whatever you write will be saved in *`file1.txt`*. Have in mind that `cat` is not a text editor per se, we will talk about text editors later on.
- By using `>`, we are redirecting the output of `cat` to the file *`file1.txt`*. After you finish writing, press `ctrl-c` to exit.
- You can use redirection (`>`), with any command.
- When working on the terminal, you can use `ctrl-c` to exit any application.
- Alternatively, if you want to create an empty file, you can use the command `touch`.
    - `$> touch file2.txt ↵`

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- If you want to display the content of the *file1.txt* using `cat` (catenation), type in the terminal,
  - `$> cat file1.txt ↲`
- To display the content of a file in a scrollable manner, we can use the command `more`.
  - `$> more file1.txt ↲`
- To display the beginning of the *file1.txt*, you can use the command `head`.
  - `$> head file1.txt ↲`
- To display the end of the *file1.txt*, you can use the command `tail`.
  - `$> tail file1.txt ↲`
- The commands `head` and `tail` are very useful when dealing with big files, we are going to address this later.
- To erase files, we use the command `rm` (remove).
  - `$> rm file2.txt ↲`
- To copy a file into another file, we can use the command `cp` (copy).
  - `$> cp file1.txt file2.txt ↲`

If *file2.txt* doesn't exist it will be created.

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- To copy a file into another directory, we can proceed as follows,
  - `$> cp file1.txt dir1/file1.txt ↵`
- To move a file into another directory or file (rename), we can use the command `mv` (move).
  - `$> mv file1.txt file2.txt ↵`
  - `$> mv file2.txt dir1 ↵`
- To copy a file from a given directory into the current directory.
  - `$> cp dir1/file2.txt . ↵`

`.` (dot) means into the current directory and it can be used with any command.

- To copy a file into your home directory,
  - `$> cp dir1/file2.txt ~ ↵`

`~` (tilde) is a shortcut for your home directory and it can be used with any command.

- To append information to an existing file,
  - `$> cat file2.txt >> file1.txt ↵`

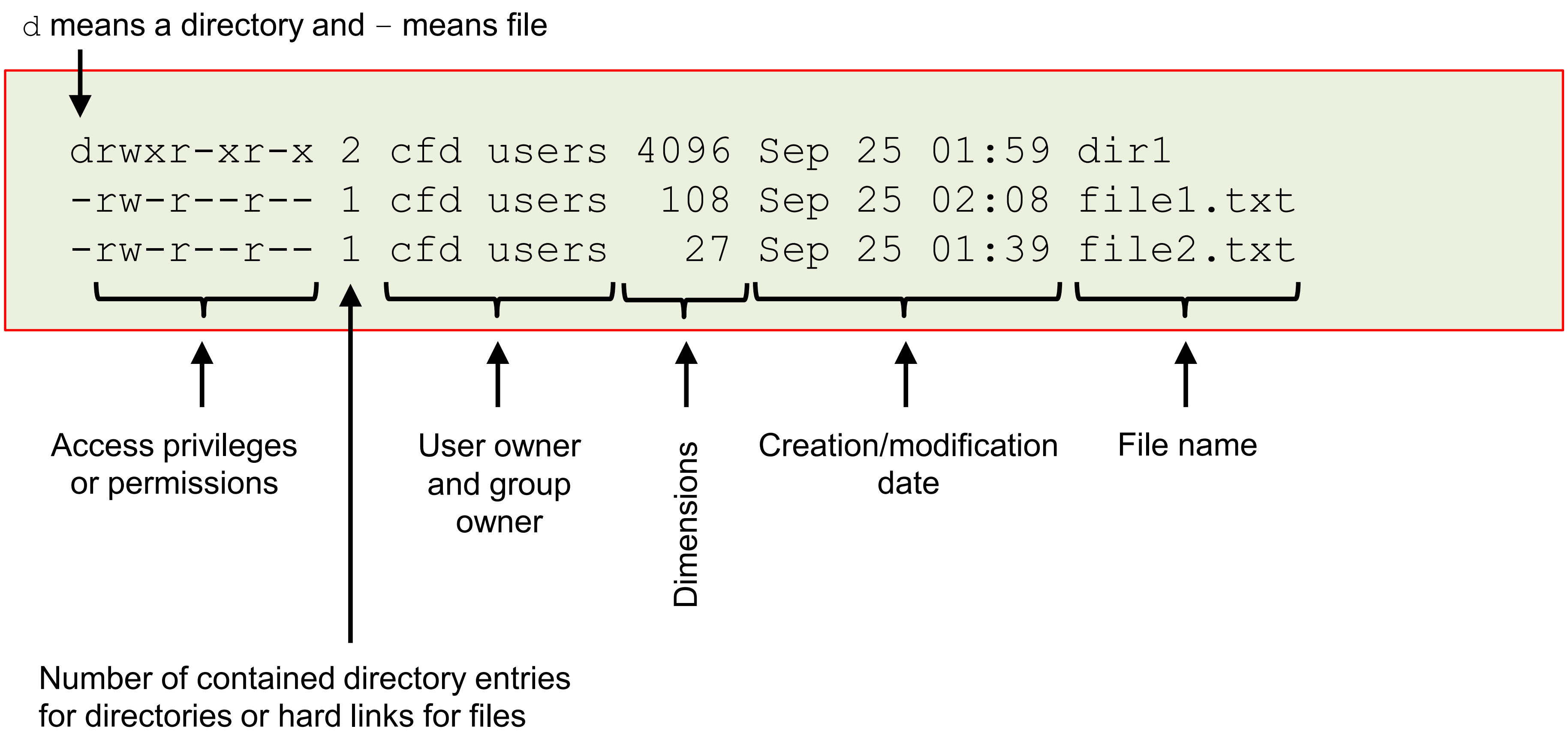
`>>` will redirect the output of `cat` to the file *`file1.txt`* and it will appended at the end of the file. If you use a single `>`, it will overwrite the file.

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- To know the properties of files and directories.
  - `$> ls -l ↲`

- You will get an output similar to this one:



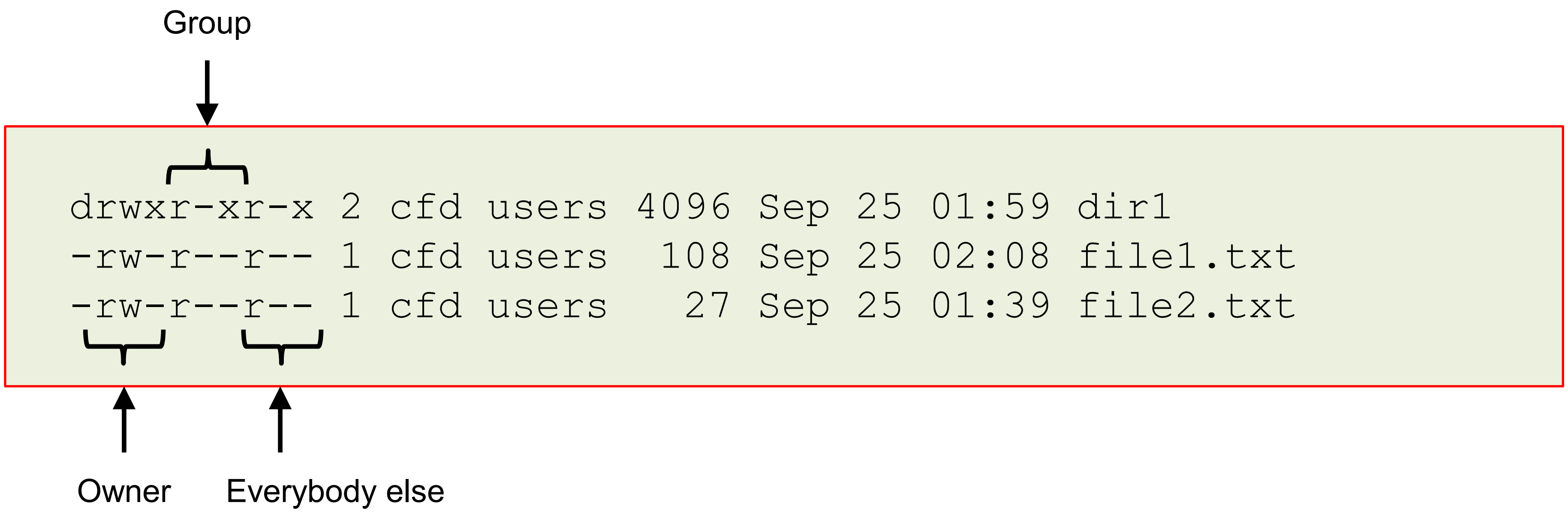
```
drwxr-xr-x 2 cfd users 4096 Sep 25 01:59 dir1
-rw-r--r-- 1 cfd users  108 Sep 25 02:08 file1.txt
-rw-r--r-- 1 cfd users   27 Sep 25 01:39 file2.txt
```

d means a directory and – means file
Access privileges or permissions
User owner and group owner
Dimensions
Creation/modification date
File name
Number of contained directory entries for directories or hard links for files

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- Let's talk about permissions details.
    - The letter `r` refers to read permission.
    - The letter `w` refers to write permission.
    - The write `x` refers to execute permission.
- In this example, the owner has read and write permission (the first three letters or `rw-`) for the files, and read, write, and execute permission (the first three letters or `rwx`) for the directories.
- The group owner has read permission (the next three letters or `r--`).
- Everybody else has read permission (the last three letters or `r--`).

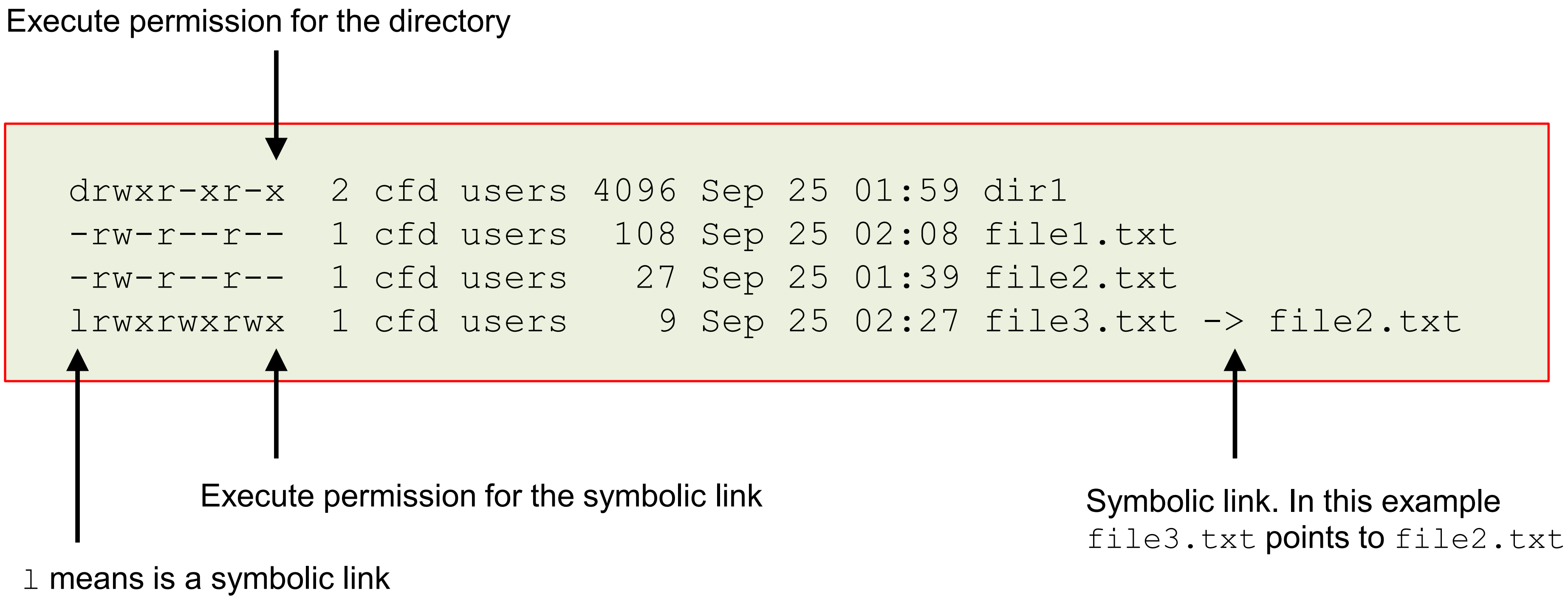


```
drwxr-xr-x 2 cfd users 4096 Sep 25 01:59 dir1
-rw-r--r-- 1 cfd users  108 Sep 25 02:08 file1.txt
-rw-r--r-- 1 cfd users   27 Sep 25 01:39 file2.txt
```

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- To create symbolic links (or shortcuts) to files and directories, we use the command `ln` (links).
    - `$> ln -s file2.txt file3.txt ↵`
- We just created a link to `file2.txt` (source) using `file3.txt` (target or link name).
- If you type `ls -l`, you should get an output similar to the following one,



```
drwxr-xr-x  2 cfd users 4096 Sep 25 01:59 dir1
-rw-r--r--  1 cfd users  108 Sep 25 02:08 file1.txt
-rw-r--r--  1 cfd users   27 Sep 25 01:39 file2.txt
lrwxrwxrwx  1 cfd users    9 Sep 25 02:27 file3.txt -> file2.txt
```

Execute permission for the directory
Execute permission for the symbolic link
Symbolic link. In this example file3.txt points to file2.txt
l means is a symbolic link

- Remember, to access files in a directory you need execute permission on that directory.

The Linux Terminal: A Crash Introduction

Basic file management and file inspection

- Let’s revisit the command `touch`. The basic function of this command is to update the timestamp of the file. If the file doesn’t exist, `touch` will create an empty file.
- Let’s say that the file *`file1.txt`* already exists, if you use touch you will update the timestamp to the current time. However, if you want to use a different timestamp, you can proceed as follows,

```
$> touch –t 201501211059 file1.txt ↵
```

- The timestamp is given as follows: year (2015), month (01), day (21), hour (10), minute (59).
- Let’s go back to the user home, and let’s create a new directory containing a sub-directory.

```
$> cd ↵
$> mkdir –p stuff/dir1 ↵
```

The `–p` argument let’s create the directory structure at once, if you don’t use this argument you will need to create each directory one at a time.

- Similar, if you want to erase the whole directory structure, you can proceed as follows

```
$> rmdir –p stuff/dir1 ↵
```

Have in mind that `rmdir` only works with empty directories.

Basic file management and file inspection

- Let’s create some empty files inside the directory **`~/stuff/dir2`**.
  - `$> mkdir ~/stuff/dir2 ↵`
  - `$> touch ~/stuff/dir2/file1.txt ↵`
  - `$> touch ~/stuff/dir2/file2.txt ↵`
- Notice that we are using the absolute path to create the files and directories.
- Let’s go to the home directory and erase the directory **`./stuff`**,
  - `$> cd ↵`
  - `$> rmdir -p stuff ↵`
- Notice that we are using the relative path of he directory **`./stuff`**.
- If the directories are not empty, using `rmdir -p` will give the following error.
  - **rmdir: failed to remove ‘stuff/’: Directory not empty**
- The command `rmdir` only works with empty directories. A solution to this problem is to use the command `rm` with the argument `-r`, which means erase recursively (everything inside the given directory).
  - `$> rm -r stuff ↵`
- Be careful when using `rm`, it erases files and directories for good. There is no simple way to recover what you just erased.

Basic file management and file inspection

- Let’s create a non-empty file, type in the terminal,
  - `$> ls -lR /usr/lib64 > file.txt ↲`
- If you use the command cat, you will realize that there is a lot information in this file.
  - `$> cat file.txt ↲`
- Here is where `head` and `tail` comes in handy. To display the first 5 lines of the file,
  - `$> head -5 file.txt ↲`
- And to display the last 5 lines of the file,
  - `$> tail -5 file.txt ↲`
- The commands `head` and `tail` will display 10 lines by default.
- To read the file *`file.txt`* in an interactive way in the terminal, we can use the command `less`.
  - `$> less file.txt ↲`

As soon as you are inside `less`, press the key `h` to get basic help, use the keyboard arrows to navigate the text, or press `q` to exit `less`.

- To determine the file type, you can use the command `file`.
  - `$> file filename_or_dirname ↲`

`file` will let you know if you are dealing with a file or a directory

Wildcards

- A wildcard is a character that can be used as a substitute for any class of character in a search, thereby greatly increasing the flexibility and efficiency of searches.
- All shell command can take wildcards.
- These are the basic set of wildcards:
    - `*` represents zero or more characters
    - `?` represents a single character
    - `[ ]` represents a range of characters
- To list all the files that start with the letter **z** in the directory **/usr/bin**,
    - `$> ls /usr/bin/z* ↵`
- To list all the files with two letters in the directory **/usr/bin**,
    - `$> ls /usr/bin/?? ↵`
- To list all the files that start with two vowels in the directory **/usr/bin**,
    - `$> ls /usr/bin/[aeiou][aeiou]*`
- To list all the files with three letters that start with the letter **z**, have a vowel in the middle, and end with any character in the directory **/usr/bin**,
    - `$> ls /usr/bin/z[aeiou]? ↵`

The Linux Terminal: A Crash Introduction

Redirecting IO

- To redirect the output of a command to a file, we can use `>` or `>>`.
- The character `>` will overwrite the file if already exists, and the character `>>` will append the information at the end of the file.
- To redirect the output of the command `ls` to a file, we can proceed as follows,
  - `$> ls -al > dir.txt ↵`
- To append more information to the file *`dir.txt`*, we can proceed as follows,
  - `$> ls -R >> dir.txt ↵`
- To redirect the standard output and the standard error to different files,
  - `$> ls -al > dir.txt 2> error.txt ↵`
  - `$> cd; grep Desktop * > out.txt 2> error.txt ↵`

  Semicolons (;) allows us to execute multiple commands on a single line (command separator).
- To redirect the standard output and the standard error to the same file,
  - `$> ls -al > dir.txt 2>&1 ↵`
- To redirect the standard error to an empty file,
  - `$> cd; grep Desktop * > out.txt 2>/dev/null ↵`

Directories and files permissions

- Hereafter, we will talk about changing directories and files permissions.
- We will introduce the following commands:
    - `chmod`
    - `chgrp`
    - `chown`
    - `id`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Directories and files permissions

- So far we have seen that in order to display files and directories permissions we can use the command `ls -l`.
- But what if we want to change files and directories permissions? To do so, we can use the command `chmod`.
- Let’s create and empty file using `touch`.

  -
    ```
    $> touch file1.txt ↵
    ```
- To list the permissions of the file, you can type in the terminal,

  -
    ```
    $> ls -l file1.txt ↵
    ```
- The output should be similar to this one,

  -
    ```
    -rw-r--r-- 1 cfd users 0 Sep 25 14:38 file1.txt
    ```
- To change the permissions using `chmod`, you can proceed as follows,

  -
    ```
    $> chmod u=rwx, g=r, o=- file1.txt ↵
    ```

  we just gave read, write, and execute permissions to the owner of the file (`u`), read permissions to the group (`g`), and no permissions to everybody else (`o`).
- These are the new permissions of the file if you use `ls -l`,

  -
    ```
    -rwxr----- 1 cfd users 0 Sep 25 14:38 file1.txt
    ```
- Remember, to access files in a directory you need execute permission on that directory.

Directories and files permissions

- To change the group of a file, you can use the command `chgrp`.
  - `$> chgrp users new_group file1.txt ↵`

  Have in mind that you need to be a member of the group new_group.
- Changing groups is something that only the owner of the file (or the root) can do.
- To list the groups you are a member of, you can use the command `id`.
- To change the owner of a file, you can use the command `chown`. Have in mind that only the root user can change the owner of a file (for security reasons).
- Switch to the super user and change the file owner as follows,

```
$> chown nobody file1.txt ↵
```

- The command `chown` can also be used to change the user and group of files and directories as follows,

```
$> chown user_name:group_name file_dir_name ↵
```

- At this point, visualize the new permissions using `ls -l`.

Managing users and groups

- Hereafter, we will talk about managing users and groups.
- We will introduce the following commands:
    - `useradd`
    - `usermod`
    - `userdel`
    - `passwd`
    - `groupadd`
    - `groupdel`
    - `su`
    - `finger`
    - `groups`
    - `who`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Managing users and groups

- First at all, to manage users and groups you need to be the root user. To become the root user use the command `su` (substitute user) with no arguments.
    - `$> su ↲`
- To create a user with the name **user1**, you can use the command `useradd`.
    - `$> useradd –m user1 ↲`

  The argument `–m` will create the user's home directory.
- To create (or change) the password of a user, we can use the command `passwd`.
    - `$> passwd user1 ↲`

  At this point, you can type in the desired password.
- To change users or login as a different user in the terminal we can use the command `su`, which stands for substitute user.
    - `$> su - user1 ↲`

  The argument `-` is telling the shell to go to the user's home.
- If you are working as the root user, `su` won't ask you for a password. Otherwise, you will need to give the user's password.

The Linux Terminal: A Crash Introduction

Managing users and groups

- If a user has no password, only the root user can create a password for that user.
- Similarly, only the root user can login using the credentials of the given user.
- To know which users are logged in, use the command `who`.
- To know the groups a user is in, use the command `groups`.
- To exit from a user account, use the command `exit`.
- To get information about a user, you use the command `finger`,
  - `$> finger user_name ↵`
- To delete a user with the name **user1**, you can use the command `userdel`.
  - `$> userdel -r user1 ↵`

  The argument `-r` will remove the user's home directory.
- To create a new group with the name **newgroup**, you can use the command `groupadd`.
  - `$> groupadd newgroup ↵`
- To delete a group, you can use the command `groupdel`.
  - `$> groupdel newgroup ↵`

The Linux Terminal: A Crash Introduction

Managing users and groups

- To modify user and group information, you can use the command `usermod`.
- To change the group of a user,
    - `$> usermod –G group_name user_name ↵`
- If at any point you want to know the users and groups defined in your system, type in the terminal,
    - `$> cat /etc/passwd ↵`
    - `$> cat /etc/group ↵`
- As a regular user you can read these files but you are not allow to modify them.
- The users password information is stored in the file `/etc/shadow`, which can be only accessed by the root.
- Do not worry, the information stored in the file `/etc/shadow` is not human readable. Not even the root user can decipher it.

Dealing with processes

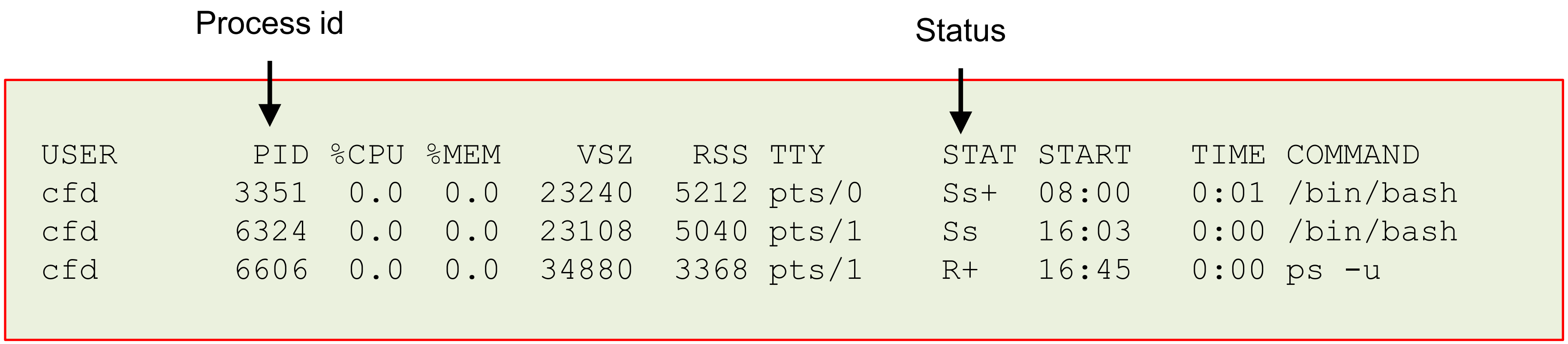
- Hereafter, we will talk about managing processes.
- We will introduce the following commands:
  - `ps`
  - `top`
  - `kill`
  - `jobs`
  - `bg`
  - `fg`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

The Linux Terminal: A Crash Introduction

Dealing with processes

- To list the processes ruining in your system, you can use the command `ps` (processes).
    - `$> ps ↲`
- This command can take many arguments, we recommend you to take a look at the help information.
- To display the processes in full listing format and only for a given user,
    - `$> ps -fu username ↲`
- To display all the running processes in full listing format,
    - `$> ps -fe ↲`
- To display processes using user-oriented format (most of the time we use this option),
    - `$> ps u  ↲`



```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
cfd         3351  0.0  0.0  23240  5212 pts/0    Ss+  08:00   0:01 /bin/bash
cfd         6324  0.0  0.0  23108  5040 pts/1    Ss   16:03   0:00 /bin/bash
cfd         6606  0.0  0.0  34880  3368 pts/1    R+   16:45   0:00 ps -u
```

The Linux Terminal: A Crash Introduction

Dealing with processes

- To terminate a running process, you can use the command `kill`. However, you will first need to identify the process id (PID). For instance, to terminate the process PID 6606,
  - `$> kill 6606 ↵`
- If you launch a program from the terminal, you will notice that the shell will be blocked while the program is running. For example, let’s launch the program gedit (which is a GUI text editor),
  - `$> gedit ↵`

If you try to run a program from the same terminal, you will notice that is not possible.

- To be able to use the same terminal, we can stop the current job by pressing the keys **ctrl** and **z**.
- At this point, you can send the job to background and have the terminal free, or you can run a new command(s) and then bring the previous job back to the foreground.
- To identify the job id, use the command `jobs`,
  - `$> jobs ↵`

Job id

```
[1]-  Running                 gedit &
[2]+  Running                 leafpad &
```

Dealing with processes

- To send the job to background, you can use the command `bg` (background),
  - `$> bg %job_id ↵`

- In a similar way, you can send a job back to the foreground by using the command `fg` (foreground),
  - `$> fg %job_id ↵`

  Have in mind that this will block the terminal again.

- If you do not want to deal with jobs ids, you can send the program right away to background by using `&` (ampersand),
  - `$> gedit & ↵`

  This is the recommended way to proceed.

Getting system information

- Hereafter, we will talk about getting system information (hardware and software).
- We will introduce the following commands:
    - `df`
    - `uptime`
    - `hostname`
    - `uname`
    - `lsb_release`
    - `lscpu`
    - `lsblk`
    - `lslogins`
    - `lsscsi`
    - `du`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Getting system information

- To display free disk space, use the command `df` (disk filesystem).
    - `$> df -h ↵`

  the `–h` argument will show the information in human readable format.
- To know how long the system has been running, use the command `uptime`.
    - `$> uptime ↵`

  It will also display users login and how busy the system is.
- To display the name of the host system, use the command `hostname`.
    - `$> hostname ↵`
- To print the operating system name, use the command `uname`.
    - `$> uname -a ↵`
- You can also use the command `lsb_release`, which will give you more information about the Linux version you are using,
    - `$> lsb_release -a ↵`
- The information of the Linux distribution can also be found in the following file,
    - `$> cat /etc/os-release ↵`

Getting system information

- To display information about the CPU architecture, use the command `lscpu`.
    - `$> lscpu ↵`
- To display the block devices (attached devices), use the command `lsblk`.
    - `$> lsblks ↵`
- To display information about known users in the system, use the command `lslogins`.
    - `$> lslogins ↵`
- To list SCSI devices (or hosts) and their attributes, use the command `lsscsi`.
    - `$> lsscsi ↵`
- To display the dimensions of a file or directory, use the command `du` (disk usage).
    - `$> du -hs  /usr/lib64 ↵`

the `-h` argument will show the information in human readable format, the argument `-s` will summarize the information.

The Linux Terminal: A Crash Introduction

Looking for information in the directory structure

- Hereafter, we will talk about looking for information in files and in the directory structure.
- We will introduce the following commands:
    - `grep`
    - `find`
    - `which`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Looking for information in the directory structure

- To locate files you can use the `find` command. For example, if you want to locate the file `hostname` in the root directory **/**,
  - `$> find / -name “hostname” 2>/dev/null ↵`

  the option `2>/dev/null` will ignore any error by redirecting the errors to the empty file */dev/null*.
- If you want to find a string inside a file, you can use the command `grep`,
  - `$> grep –n “user” /etc/passwd ↵`

  This will look for the string user in the file */etc/passwd*. The argument `–r` means look inside all the sub-directories, and the argument `–n` means show the line number.
- If you want to find a string inside all the files inside a directory, you can proceed as follows,
  - `$> grep –r –n “user” /etc/ ↵`

  This will look for the string user in the directory */etc*. The argument `–r` means look inside all the sub-directories, and the argument `–n` means show the line number.

- If you want to display the full path of shell commands, use the command `which`.
  - `$> which find ↵`

Combining commands (pipes and pipelining)

- Hereafter, we will talk about how to combine commands to do complex tasks.
- We will introduce the following commands:
  - `|` (pipe)
  - `xargs`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

Combining commands (pipes and pipelining)

- To find the files that contain the string wc in their name, and inside the directory `/usr/bin`, we can proceed as follows,

  -
    ```
    $> ls -lR /usr/bin | grep “wc” ↵
    ```

  the character | (pipe) is used to concatenate commands. You can concatenate as many command as you like.

- Let’s create four empty files,

  -
    ```
    $> touch f1.txt f2.txt f3.txt f101.txt ↵
    ```
- To erase these files using wildcards, and by combining the commands `ls` and `rm`, you can proceed as follows,

  -
    ```
    $> ls f[1_5].txt | xargs rm ↵
    ```

  `xargs` is used to redirect the output of `ls` to `rm`. This command will only erase the files *f1.txt*, *f2.txt*, and *f3.txt*.
- If you want to erase all files with the extension `txt`, and asking for confirmation,

  -
    ```
    $> rm –i *.txt ↵
    ```

The Linux Terminal: A Crash Introduction

Compressing and uncompressing files and directories

- Hereafter, we will talk about how to compress/uncompress files and directories.
- We will introduce the following commands:
    - `tar`
    - `zip`
    - `unzip`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

The Linux Terminal: A Crash Introduction

Compressing and uncompressing files and directories

- First, let’s create a file containing some information,
  - `$> ls -lR > dir.txt ↵`
- Let’s now create a new directory and copy the newly created file into this directory,
  - `$> mkdir dir1 ↵`
  - `$> cp dir.txt dir1 ↵`
- To compress a directory using the command `tar`,
  - `$> tar –czvf filename.tar.gz dir1/ ↵`
- To compress a file using the command `tar`,
  - `$> tar –czvf filename.tar.gz dir.txt ↵`
- To uncompress a `*.tar.gz` file using the command `tar`,
  - `tar –xzvf filename.tar.gz  ↵`
- To compress a file/directory using the command `zip`,
  - `zip filename.zip dir1/dir.txt ↵`
- To uncompress a `*.zip` file using the command `unzip`,
  - `unzip filename.zip  ↵`

The Linux Terminal: A Crash Introduction

User environment

- Hereafter, we will talk about the user environment.
- We will introduce the following commands:
    - `env`
    - `echo`
    - `source`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

User environment

- The user environment refers to where to look for programs and libraries, and how to display the information on the shell (not only limited to this).
- To display your user environment, use the command `env` (environment).
  - `$> env ↵`
- The `env` command will display a lot of information, for the moment, the most important variables are **PATH** and **LD_LIBRARY_PATH**.
- The variable **PATH** refers to where the shell will look for programs. To know the definition of the **PATH** variable, type in the terminal,
  - `$> echo $PATH ↵`
- When you type a command in the terminal and it is defined in your **PATH**, the shell will execute it. Otherwise, the shell will give you the error command not found.
- To add a new directory to your **PATH**, type in the terminal,
  - `$> PATH=$PATH:/home/user/bin ↵`
  - `$> export PATH ↵`

This will add **`/home/use/bin`** to your current **PATH**. The shell will look for programs in this directory.

The Linux Terminal: A Crash Introduction

User environment

- The variable **LD_LIBRARY_PATH** refers to where the shell will look for libraries. To know the definition of the **LD_LIBRARY_PATH** variable, type in the terminal,
    - `$> echo $LD_LIBRARY_PATH ↵`

- To add a new directory to your **LD_LIBRARY_PATH**, type in the terminal,
    - `$> LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user/lib ↵`
    - `$> export LD_LIBRARY_PATH ↵`

This will add **`/home/use/lib`** to your current **LD_LIBRARY_PATH**. The shell will look for libraries in this directory.

- If you do not want to type the new **PATH** and **LD_LIBRARY_PATH** variables every time you open a terminal or login, you can add these variables to your *`.bashrc`* file.
- The *`.bashrc`* file is a shell script that bash runs whenever it is started. You can put any command that you want to execute automatically in this file. It is kind of a configuration file.
- The *`.bashrc`* file is located in the user home, and it's hidden as its name starts with the character `.` (dot).

User environment

- Open the *.bashrc* file using gedit (a GUI based text editor):
    - `$> gedit .bashrc & ↵`

- Add the variables information to the *.bashrc* file (at the end of the file),

```
PATH=$PATH:/home/user_name/bin
export PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user_name/lib
export LD_LIBRARY_PATH
```

- In order to make the modifications active in the current terminal, you will need to source the *.bashrc* file,
    - `$> source .bashrc ↵`

- From now on, every time you login or open a new terminal, the new variables will be defined or loaded automatically.
- When you source *.bashrc* file, you load the configuration to your current environment.

Aliases and shell built-in commands

- Hereafter, we will talk about how to define aliases.
- We will introduce the following commands:
    - `alias`
    - `type`
    - `unalias`

- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

The Linux Terminal: A Crash Introduction

Aliases and shell built-in commands

- Aliases are shortcuts to commands.
- If you are using bash shell, you define aliases in the *.bashrc* file.
- To display the aliases defined, use the command `alias`.
    - `$> alias ↵`
- To define a new alias, you will need to add it to your *.bashrc* file.
- Open the *.bashrc* file using gedit (a GUI based text editor):
    - `$> gedit .bashrc & ↵`

- Add the following alias (at the end of the file),

  alias ll=‘ls –al –color=auto’

- Source the *.bashrc* file.
    - `$> source .bashrc ↵`
- To use the new alias, type in the terminal its name,
    - `$> ll ↵`

Aliases and shell built-in commands

- To know if a command is an alias, a shell built-in command, or a normal program, we ca use the command `type`.
    - `$> type ll ↵`
    - `$> type pwd ↵`
    - `$> type grep ↵`
    - `$> type cowsay ↵`

- Built-in command are programs bundled with the shell.
- To get a list of the built-in commands type `help` in terminal.
- To get help on a built-in command, type `help name_of_the_command`. For example,
    - `$> help type ↵`

- To remove an alias from the current aliases definition,
    - `$> unalias ll ↵`

Using the text editor vi

- Hereafter, we will talk about how to use the text editor vi.
- We will introduce the following commands:
    - `vi`
- Remember, these commands can take arguments.
- If you need help on how to use a command, type in the terminal `command_name --help` or `man command_name`.

The Linux Terminal: A Crash Introduction

Using the text editor vi

- The program vi is a command line text editor, or in other words, it does not use a GUI.
- Everything in vi is done via the keyboard.
- It is the default text editor on most UNIX/Linux systems.
- Have in mind that vi is not the only option. Hereafter we list a few command line based editors alternatives:
    - emacs
    - nano
    - pico
    - joe

- And if you are looking for GUI based text editors, you can try the following ones:
    - gedit
    - leafpad
    - kwrite
    - nedit

Using the text editor vi

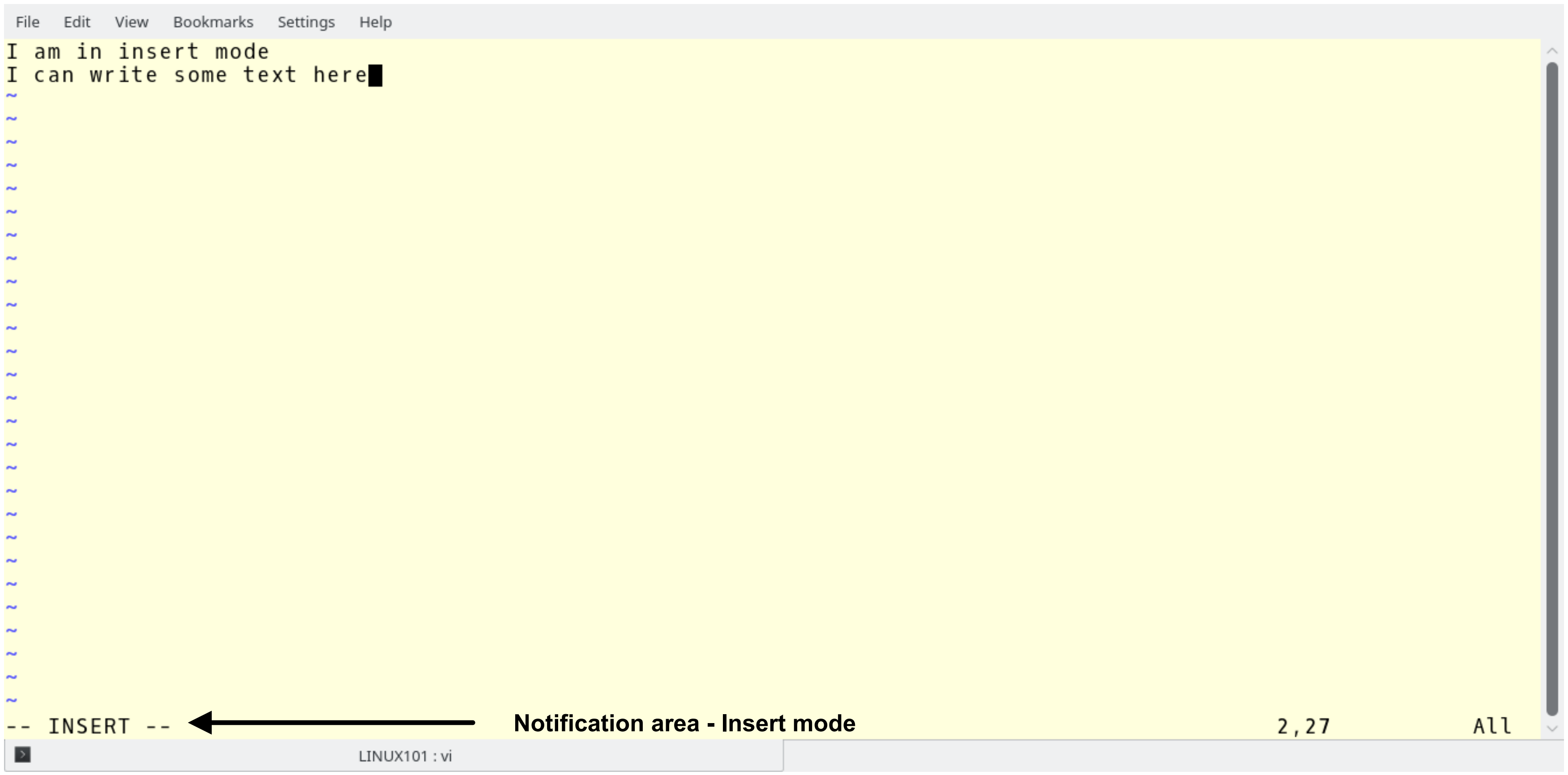
- To start using vi, type in the terminal,
  - `$> vi ↵`
- To start using vi and create a new empty file,
  - `$> vi newfile.txt ↵`
- To edit an existing file,
  - `$> vi existingfile.txt ↵`

- There are two modes in vi. **Insert** mode and **Edit** mode.
- In Insert mode you can input or enter content into the file.
- In edit mode you can move around the file, perform actions such as deleting, copying, search and replace, saving, etc.
- A common mistake is to start entering commands without first going back into edit mode or to start typing input without first going into insert mode.
- At the beginning you will think that vi is terrible, but as you get use to it you will realize that it is very efficient and it will save you a lot time.

The Linux Terminal: A Crash Introduction

Using the text editor vi

- To know if you are in insert mode, look at the bottom of the terminal (notification area). If you see the word INSERT, you are in insert mode.
- If you are in insert mode and you want to switch to edit mode, press the key **esc**.
- If you are in edit mode and you want to switch to insert mode, press the key **i**.

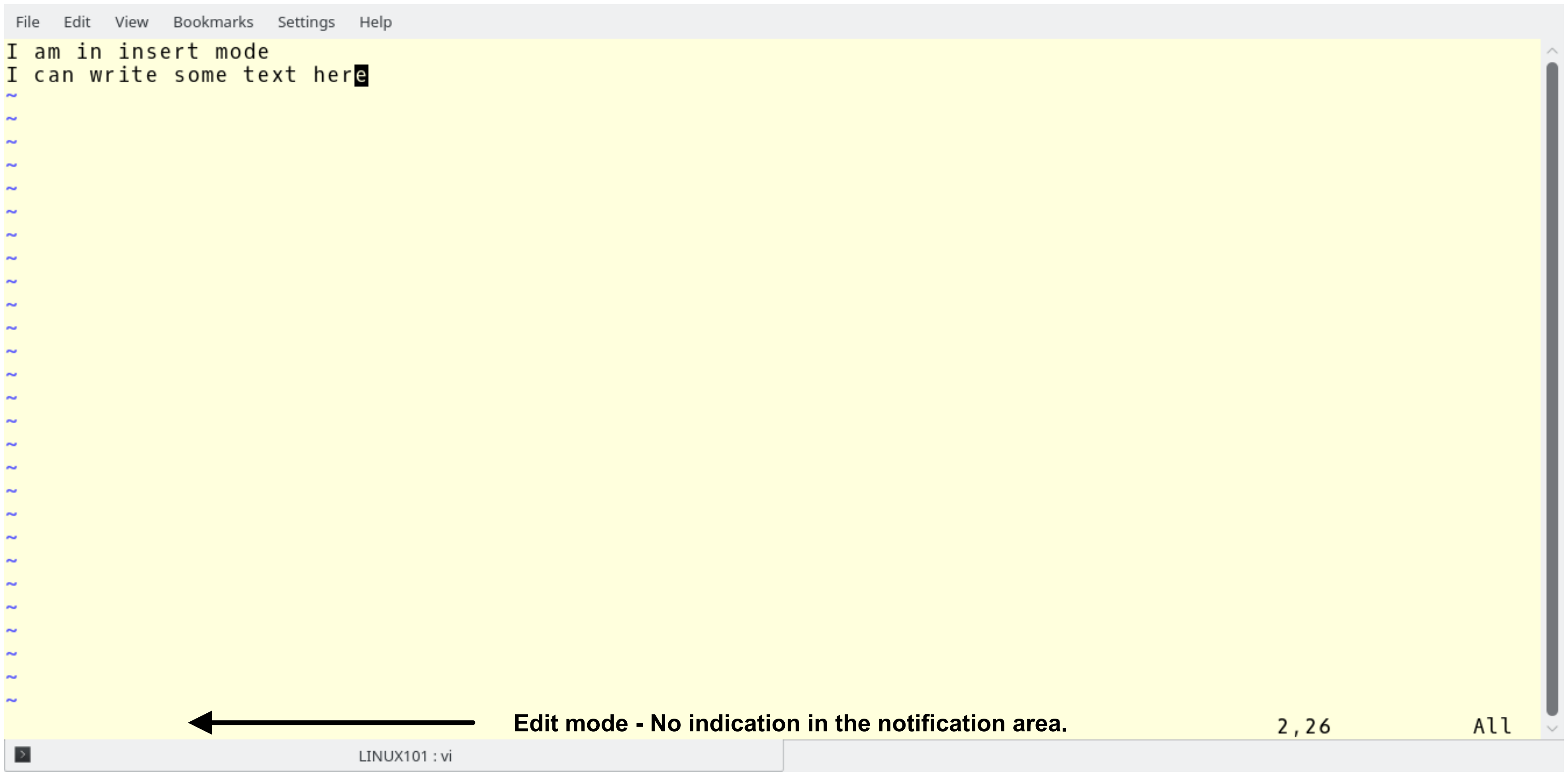


```
I am in insert mode
I can write some text here
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                    2,27          All
```

Notification area - Insert mode

Using the text editor vi

- To know if you are in edit mode, look at the bottom of the terminal (notification area). If you do not see anything there, you are in edit mode.
- If you are in insert mode and you want to switch to edit mode, press the key **esc**.
- If you are in edit mode and you want to switch to insert mode, press the key **i**.



```
I am in insert mode
I can write some text here
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                  2,26          All
```

Edit mode - No indication in the notification area.

Using the text editor vi

- To navigate the file use the arrow keys (← → ↑ ↓).
- While in edit mode, you can use the following key combinations to perform the following operations,

:w	save the file
:w name	save the file with the specified name
:wq	save the file and quit
:q	quit
:q!	quit without saving changes
:n	go to line number **n**
/pattern	search the word pattern (press n/N to find the next/previous occurrence)
?pattern	search the word pattern backward
:s/pattern/replace/	search the word pattern and replace it with the word replace (first occurrence)
:%s/pattern/replace/	search the word pattern and replace it with the word replace (all occurrences)
dd	deletes an entire line
x	deletes the character under the cursor
u	undo last change
U	undo all changes
gg	go to the beginning of the file
G	go to the end of the file
ctrl f	move forward one screen
ctrl b	move backward one screen

- There are many more combinations!

The Linux Terminal: A Crash Introduction

Bonus command

- A bonus command,
  - `$> cowsay "Thank you for your attention and enjoy the Linux terminal" ↵`

```
 ________________________________________
/ Thank you for your attention and enjoy \
\ the Linux terminal                     /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

- You might need to install the command `cowsay`.

The Linux Terminal: A Crash Introduction

A few good Linux references

- **The Linux Command Line: A Complete Introduction**
  W. Shotts, Jr. 2012, No Starch Press.
- **How Linux Works: What Every Superuser Should Know**.
  B. Ward. 2014, No Starch Press.
- **Linux Pocket Guide**.
  D. J. Barret. 2016, O’Reilly Media.
- **Linux Command Line for Beginners**.
  G. Allen. 2015, O’Reilly Media.
- http://www.linuxcommand.org/
- http://www.computerhope.com/unix.htm
- http://www.ryanstutorials.net/
- https://doc.opensuse.org/documentation/leap/startup/html/book.opensuse.startup/part.bash.html