A command line, or terminal, is a text based interface to the system. You are able to enter commands by typing them on the keyboard and feedback will be given to you similarly as text.

Within a terminal you have what is known as a shell. This is a part of the operating system that defines how the terminal will behave and looks after running (or executing) commands for you. There are various shells available but the most common one is called bash which stands for Bourne again shell.

This course will assume you are using bash as your shell.
If you would like to know which shell you are using you may use a command called **echo** to display a **system variable** stating your current shell.

**echo** is a command which is used to display messages.

**syntax**: `echo <message to display>` or `$<system variable name>`
An environment variable is a named object that contains data used by one or more applications.

In simple terms, it is a variable with a name and a value. The value of an environmental variable can for example be the location of all executable files in the file system, the default editor that should be used, or the system local settings.

Users new to Linux may often find this way of managing settings a bit unmanageable. However, environment variables provide a simple way to share configuration settings between multiple applications and processes in Linux.
The coreutils package contains the programs printenv and env.

To list the current environmental variables with values:
DE indicates the Desktop Environment being used. Recognised values of DE variable are: gnome, kde, xfce, lxde and mate.

PATH contains a colon-separated list of directories in which your system looks for executable files. When a regular command (e.g., ls, rc-update or ic\emerge) is interpreted by the shell (e.g., bash or zsh), the shell looks for an executable file with the same name as your command in the listed directories, and executes it. To run executables that are not listed in PATH, the absolute path to the executable must be given: /bin/ls.

HOME contains the path to the home directory of the current user. This variable can be used by applications to associate configuration files and such like with the user running it.

PWD contains the path to your working directory.
Globally: The following files should be used for defining global environment variables on your system: `/etc/profile`, `/etc/bash.bashrc` and `/etc/environment`. Each of these files has different limitations, so you should carefully select the appropriate one for your purposes.

- `/etc/profile` initializes variables for login shells only. It does, however, run scripts and can be used by all Bourne shell compatible shells.

- `/etc/bash.bashrc` initializes variables for interactive shells only. It also runs scripts but (as its name implies) is Bash specific.

- `/etc/environment` is used by the PAM-env module and is agnostic to login/non-login, interactive/non-interactive and also Bash/non-Bash, so scripting or glob expansion cannot be used. The file only accepts `variable=value` pairs.
**Per User:** You do not always want to define an environment variable globally. You have to use `export` command.
Linux Permissions

- Linux permissions dictate 3 things you may do with a file, **read**, **write** and **execute**. They are referred to in Linux by a single letter each.
  - **r** read - you may view the contents of the file.
  - **w** write - you may change the contents of the file.
  - **x** execute - you may execute or run the file if it is a program or script.

- For every file we define 3 sets of people for whom we may specify permissions.
  - **owner** - a single person who owns the file. (typically the person who created the file but ownership may be granted to someone else by certain users)
  - **group** - every file belongs to a single group.
  - **others** - everyone else who is not in the group or the owner.

- Three permissions and three groups of people. That's about all there is to permissions really. Now let's see how we can view and change them.
View Linux Permissions

*To view Linux Permissions, digit `ls -l [file]`

In the above example the first **10 characters** of the output are what we look at to identify permissions.

- The first character identifies the **file type**. If it is a dash (`-`) then it is a normal file. If it is a `d` then it is a directory.
- The following 3 characters represent the permissions for the **owner**. A letter represents the presence of a permission and a dash (`-`) represents the absence of a permission. In this example the owner has all permissions (read, write and execute).
- The following 3 characters represent the permissions for the **group**. In this example the group has the ability to read but not write or execute. **Note that the order of permissions is always read, then write then execute.**
- Finally the last 3 characters represent the permissions for others (or everyone else). In this example they have the read permission and nothing else.
To view Linux Permissions, digit `ls -l [file]`
To change Linux Permissions, digit `chmod [permissions] [path]`

Grant the execute permission to the group. Then remove the write permission for the owner.

```
1. user@bash: ls -l frog.png
2. -rwxr-----x 1 harry users 2.7K Jan 4 07:32 frog.png
3. user@bash:
4. user@bash: chmod g+x frog.png
5. user@bash: ls -l frog.png
6. -rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
7. user@bash:
8. user@bash: chmod u-w frog.png
9. user@bash: ls -l frog.png
10. -r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```
To change Linux Permissions, digit `chmod [permissions] [path]`

Don't want to assign permissions individually? We can assign multiple permissions at once.

```
1. user@bash: ls -l frog.png
2. -rwxr--r-- 1 harry users 2.7K Jan 4 07:32 frog.png
3. user@bash:
4. user@bash: chmod g+wx frog.png
5. user@bash: ls -l frog.png
6. -rwxrwx--x 1 harry users 2.7K Jan 4 07:32 frog.png
7. user@bash:
8. user@bash: chmod go-x frog.png
9. user@bash: ls -l frog.png
10. -rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```
The method outlined above is not too hard for setting permissions but it can be a little tedious if we have a specific set of permissions we should like to apply regularly to certain files. Luckily, there is a shorthand way to specify permissions that makes this easy.

<table>
<thead>
<tr>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>
The method outlined above is not too hard for setting permissions but it can be a little tedious if we have a specific set of permissions we should like to apply regularly to certain files. Luckily, there is a shorthand way to specify permissions that makes this easy.

<table>
<thead>
<tr>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>

Terminal

1. user@bash: ls -l frog.png
2. -rw-r--x- 1 harry users 2.7K Jan 4 07:32 frog.png
3. user@bash:
4. user@bash: chmod 751 frog.png
5. user@bash: ls -l frog.png
6. -rw-r--x- 1 harry users 2.7K Jan 4 07:32 frog.png
7. user@bash:
8. user@bash: chmod 240 frog.png
9. user@bash: ls -l frog.png
10. --w-r----- 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
The root user is a superuser who is allowed to do anything and everything on the system. Typically the administrators of a system would be the only ones who have access to the root account and would use it to maintain the system.

Typically normal users would mostly only have access to files and directories in their home directory and maybe a few others for the purposes of sharing and collaborating on work and this helps to maintain the security and stability of the system.
Becoming Superuser

- However a normal user can become a superuser or acquire superuser privileges.

- The command `su` is used to become another user, inheriting identifier, group identifier, privileges, execute permissions and navigation and in some cases, environment variables, aliases, path defaults and so on - in a shell or a virtual terminal.

  `su <parameters> username`

- In this way you can became root by composing “su root” or simply “su”.

- Then you can go back to the normal user session by using the command “exit”
However a normal user can become a superuser or acquire superuser privileges.

The command `sudo` is used to acquire privileges of another user (no to become another user).

```
sudo <parameters> command
```

In this way you can execute the same processes as you were root.
Comandi di uso frequente

- **find percorso -name nome_file [INVIO]**: visualizza tutti i file che si trovano sotto “percorso” aventi “nome_file”.

- **find /tmp -size +10000k [INVIO]**: visualizza i file memorizzati sotto /tmp aventi dimensione maggiore di 10000k

- **find /home -user topolino [INVIO]**: visualizza i file memorizzati sotto /home di proprietà di topolino.

- **which programma [INVIO]**: visualizza il percorso completo di dove si trova il comando

- **grep nome [INVIO]**: filtra le righe che contengono la parola “nome”

- **more programma [INVIO]**: interrompe la visualizzazione quando si riempie lo schermo ed attende la pressione di un tasto per proseguire

- **clear [INVIO]**: ripulisce il terminale
Comandi di uso frequente

- **cat filename**: visualizza il contenuto del file di nome “filename”.

- **cat filename1 filename2 filename3 [INVIO]**: visualizza la concatenazione dei tre file con nomi “filename1”, “filename2”, “filename3”

- **cat filename > newfile [INVIO]**: copia il contenuto del file con nome “filename” nel file con nome “newfile” (sovrascrivendolo).

- **cat filename >> newfile [INVIO]**: “appende” il contenuto del file con nome “filename” alla fine del file con nome “newfile”

- **cat - > newfile [INVIO]**: tutto quello che viene digitato con la tastiera viene memorizzato nel file con nome “newfile”

- **cat -n filename [INVIO]**: nello stampare il contenuto di “filename” enumera le righe

- **clear [INVIO]**: ripulisce il terminale
Comandi di uso frequente

- **head**: prints the first so many lines of it's input. By default it will print the first 10 lines but we may modify this with a command line argument.

- **tail**: prints the last so many lines of it's input. By default it will print the last 10 lines but we may modify this with a command line argument.

- **sort**: will sort it's input, nice and simple.

- **nl**: stands for number lines and it does just that

- **nl -s '. ' -w 10 sample.txt**: The first one -s specifies what should be printed after the number while the second one -w specifies how much padding to put before the numbers

- **wc**: stands for word count and it does just that (as well as characters and lines. By default it will give a count of all 3 but using command line options we may limit it to just what we are after

- **cut**: is a nice little program to use if your content is separated into fields (columns) and you only want certain fields
Comandi di uso frequente

- **uniq**: stands for unique and its job is to remove duplicate lines from the data.

- **tac**: Linux guys are known for having a funny sense of humor. The program tac is actually cat in reverse

- **less**: a program similar to more, but which allows backward movement in the file as well as forward movement
We'll take a look at a mechanism for sending data from one program to another. It's called **piping** and the operator we use is `(|)`. What this operator does is feed the output from the program on the left as input to the program on the right. In the example below we will list only the first 3 files in the directory.

```
1. user@bash: ls
2. barry.txt bob example.png firstfile foo1 myoutput video.mpeg
3. user@bash: ls | head -3
4. barry.txt
5. bob
6. example.png
7. user@bash:
```
We may pipe as many programs together as we like. In the below example we have then piped the output to tail so as to get only the third file.

You may combine pipes and redirection too.
In this example we are sorting the listing of a directory so that all the directories are listed first.

```
user@bash:  ls -l /etc | tail -n +2 | sort
```

<table>
<thead>
<tr>
<th></th>
<th>Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>user@bash:  ls -l /etc</td>
</tr>
<tr>
<td>2</td>
<td>drwxrwxr-x 3 nagios nagcmd 4096 Mar 29 08:52 nagios</td>
</tr>
<tr>
<td>3</td>
<td>drwxr-x--- 2 news news 4096 Jan 27 02:22 news</td>
</tr>
<tr>
<td>4</td>
<td>drwxr-x--- 2 root mysql 4096 Mar 6 22:39 mysql</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>user@bash:</td>
</tr>
</tbody>
</table>
Lesson Learned!

THANK YOU

Current conditions at Pescara, Italy (UBP) 42.26N 014.12E (UBP)
Last updated Feb 10, 2012 - 02:50 PM EST / 2012.02.10 1950 UTC
Temperature: 1 C
Relative Humidity: 80%
Wind: from the W (270 degrees) at 15 MPH (2.7 KIT) gusting to 45 KPH
Weather: light snow grains
Sky conditions: overcast

 Formats available for this Movie: flv
 Selected format: flv
 Progress: 01.37%  13.4M of 25.1M  693.6k/s