The UNIX Command Line: A Beginner's Guide

Many people are accustomed to using computers through a graphical user interface (GUI). However, scientists and tech enthusiasts often prefer the UNIX command-line shell. You might wonder what makes this text-based interface so popular. Is it merely nostalgia at play?

UNIX vs. Windows

UNIX and Windows are two major operating systems, each with its own unique traits. UNIX, known for its open-source nature, operates under the General Public License. It's highly flexible, portable, and well-suited for network management. In contrast, Windows, developed by Microsoft, is a proprietary system designed with a graphical user interface (GUI) for user-friendly interaction.

UNIX is predominantly text-based, which might pose a slight learning curve for newcomers, while Windows provides a graphical interface for a straightforward user experience. UNIX shines in multi-processing, enabling simultaneous executions with separate address spaces, whereas Windows favors multi-threading, creating multiple threads within a single process. UNIX prioritizes security through explicit file execution permissions, while Windows, due to simpler permissions, may be more susceptible to malware.

CLI vs. GUI: Unveiling the Differences

Performance and Precision: The CLI operates through text-based commands, directing all processing power to the task at hand, resulting in high efficiency and precision. Conversely, GUI relies on visual elements like icons and windows, which consume system resources and may reduce performance. While CLI offers granular control and functionality, GUI provides a visually intuitive interface but sacrifices some precision due to its graphical nature.

Learning Curve and Ease of Use: The CLI comes with a steeper learning curve, requiring the memorization of commands with potentially complex syntax. Different shells may use varying commands. On the flip side, GUI is known for its user-friendliness and minimal learning curve. It presents functions through dialog boxes, icons, and graphical elements, making it highly intuitive and suitable for beginners.

Automation and Resource Requirements: The CLI excels in automation, crucial for managing multiple systems and configurations. It enables the creation of reusable scripts containing various commands. In contrast, GUI lacks scripting and automation capabilities, requiring users to repeat actions through dialog boxes. CLI demands fewer system resources as it operates within a terminal window, whereas GUI consumes more memory and processing power due to its graphical components.

One last detail to note: The UNIX command-line interpreter is known as a "Shell." Several UNIX shells coexist today. This tutorial focuses on the "Bourne-Again shell" (bash), which is the default interactive shell on most Linux systems. However, the commands introduced here are understood by all UNIX shells.

UNIX CLI Part 1/4: Safe Commands

Welcome to the world of UNIX Command Line Interface (CLI). In this section, we'll start with the basics of CLI commands. Think of the CLI as your command center—a place where you can type commands and receive feedback.

Exploring: The `ls` Command (Harmless)

Our first command is `ls`, which stands for "list directory contents." Think of it as your trusty flashlight in the UNIX world. The best part? It's absolutely harmless; you can't break anything with it.

```
> ls
Desktop
Downloads
Documents
(...)
```

In this example, `ls` was executed in a location with three folders: `Desktop`, `Downloads`, and `Documents`.

You can point your flashlight wherever you want by specifying a target:

```
> ls Documents
BOOKS
PROJECTS
TMP
(...)

**Note that commands and arguments are separated by the "space" character.**

> ls Documents/BOOKS
FortranModernisationWorkshop.pdf
Organizational_Analysis_Book_2016.pdf

> ls Documents/BOOKS/FortranModernisationWorkshop.pdf
Documents/BOOKS/FortranModernisationWorkshop.pdf
```

You can tweak the command's behavior to gather more information. For example:

```
> ls -lrt Documents
drwxr-xr-x  51 dauptain  staff   1632 Sep  6 15:19 BOOKS
drwxr-xr-x  12 dauptain  staff    384 Sep 18 14:28 PROJECTS
drwxr-xr-x  22 dauptain  staff    704 Sep 21 09:16 TMP
```

- `l` lists files in a long format, showing details like dates, ownership, and permissions.
- `t` sorts by descending time modified (most recent first).
- `r` reverses the sorting order (putting the most recent at the bottom).

(I personally use `-lrt` a lot!)

Autocompletion (Harmless)

Typing every letter for each command can be tiresome. Most terminals offer **autocompletion**. Just hit the Tab key (often noted as `\t`) to see if your input is enough to complete the full name:

```
> ls D\t
Desktop
Downloads
Documents
> ls Docu\t
> ls Documents/\t
BOOKS
PROJECTS
TMP
(...)

> ls Documents/B\t
> ls Documents/BOOKS/F\t
> ls Documents/BOOKS/FortranModernisationWorkshop.pdf
Documents/BOOKS/FortranModernisationWorkshop.pdf
```

Wildcards (Harmless)

To speed up your searches, consider using wildcards. Use `*` for any group of characters and `?` for any single character:

```
> ls *.pdf  # Lists all files ending with .pdf
> ls */*.pdf  # Lists all first-level files ending with .pdf
> ls */*/*.pdf  # Lists all second-level files ending with .pdf

FortranModernisationWorkshop.pdf
Organizational_Analysis_Book_2016.pdf

> ls Documents/BOOKS/Fortran*  # You get the idea!
Documents/BOOKS/FortranModernisationWorkshop.pdf

> ls Documents/BOOKS/F?rtranM?dernisationW?rksh?p.pdf
Documents/BOOKS/FortranModernisationWorkshop.pdf
```

What we've learned about the `ls` command applies to many others.

Dive Deeper: The `man` Command (Harmless)

The `man` command is your gateway to understanding UNIX commands and their options. Try it on `ls`:

```
> man
What manual page do you want?
> man ls
```

The `man` command displays on-line manual pages, providing comprehensive information about the command's usage and options. You can even try `> man man` to learn how to use `man` itself!

Navigating: The `cd` Command (Harmless)

Now, let's explore directory navigation with the `cd` (change directory) command.
Before we begin, remember these conventions:

1. `.` refers to the current folder.
2. `..` refers to the parent folder (one step back).

We'll also use `pwd` (present working directory) to keep track of our location:

```
> pwd
/Users/dauptain/
> ls
Desktop
Downloads
Documents
> cd Documents
> pwd
/Users/dauptain/Documents
> ls
BOOKS
PROJECTS
TMP
> cd BOOKS
> pwd
/Users/dauptain/Documents/BOOKS
> ls
FortranModernisationWorkshop.pdf
Organizational_Analysis_Book_2016.pdf
> cd ..
> pwd
/Users/dauptain/Documents
> cd ..
> pwd
/Users/dauptain
```

Once you've mastered navigation with full names, try using **autocompletion** (Tab key) to speed things up. And if you ever feel lost, `> cd` will return you to your "home directory," where you started.

1. **When you're lost or far from home**: The raw command `> cd` will send you back to your "home directory," your starting point in the terminal.
2. The mysterious `> cd -` will take you to the *previous* folder you visited, a helpful shortcut when you're switching between two locations frequently.

Peek at Text: `cat`, `head`, and `tail` Commands (Harmless)

These commands help you examine file contents as text. The `cat` command prints any content, whether ASCII or binary. For example, most UNIX distributions include an ASCII file with a list of words, which you can access at `/usr/share/dict/words` or `/usr/dict/words`.

```
> cat /usr/share/dict/words
A
a
aa
aal
aalii
aam
Aani
aardvark
aardwolf
Aaron
(...)
```

It's the simplest way to view an entire file. You can also use `head` or `tail` to read the beginning or end. By default, they display ten lines, but you can customize this with a few extra characters.

```
> head /usr/share/dict/words
A
a
aa
aal
aalii
aam
Aani
aardvark
aardwolf
Aaron
> head -n 4 /usr/share/dict/words
A
a
aa
aal
> tail /usr/share/dict/words
zymotoxic
zymurgy
Zyrenian
Zyrian
Zyryan
zythem
Zythia
zythum
Zyzomys
Zyzzogeton
> tail -n 4 /usr/share/dict/words
Zythia
zythum
Zyzomys
Zyzzogeton
```

Try mixing this commands. Experiment **completion** and **wildcards** until you feel you are speeding up. Use `man` to discover new options and try some of them.

So, in this part, we've introduced you to commands that are not only safe but also your first steps into the world of UNIX CLI.

UNIX CLI Part 2/4: File Management Commands

In this section, we'll dive into essential file management commands.

Touch a File: `touch`

The `touch` command might seem simple, but it serves a crucial purpose. It updates a file's access and modification times without changing the file itself. It's probably the smallest modification you can make without breaking things.

Interestingly, if the file doesn't exist, `touch` will create an empty file for you.

```
> ls -lrt tmp*
-rw-r--r--  1 dauptain  staff  0 Sep 25 07:16 tmp.txt
> touch tmp.txt
> ls -lrt tmp*
-rw-r--r--  1 dauptain  staff  0 Sep 25 07:20 tmp.txt
> touch tmp2.txt
> ls -lrt tmp*
-rw-r--r--  1 dauptain  staff  0 Sep 25 07:20 tmp.txt
-rw-r--r--  1 dauptain  staff  0 Sep 25 07:21 tmp2.txt
```

Create a Folder: `mkdir`

Creating a folder is a breeze with the `mkdir` command.

```
> ls      # When there's nothing to report, nothing happens
> mkdir TEST
> ls
TEST
> mkdir TEST
mkdir: TEST: File exists
> cd TEST
> ls
```

Copy a File/Folder: `cp`

The `cp` command does exactly what you'd expect : it creates a duplicate of a file. Some users add the `-i` flag for safety, which prompts for confirmation before overwriting a file.

```
> ls tmp*
tmp.txt         tmp2.txt
> cp tmp.txt tmp_copy.txt
> ls tmp*
tmp.txt         tmp2.txt        tmp_copy.txt
> cp tmp2.txt tmp_copy.txt
> ls tmp*
tmp.txt         tmp2.txt        tmp_copy.txt
> cp -i tmp.txt tmp_copy.txt
overwrite tmp_copy.txt? (y/n [n]) y
tmp.txt         tmp2.txt        tmp_copy.txt
```

You can use `cp` with more complex folder and file targets by separating items with a `/` (slash):

```
> mkdir TMP  # Create a dummy void folder
> mkdir TMP2 # Create another dummy folder
> touch TMP/test1.txt # Create a dummy file
> cp TMP/test1.txt TMP2/test2.txt
> ls TMP2
test2.txt
```

If you need to copy an entire folder, you must explicitly use the recursive `-r` flag:

```
> mkdir TMP  # Create a dummy void folder
> cp TMP TMP3
cp: TMP is a directory (not copied).
> cp -r TMP TMP3
> ls TMP3
test1.txt
```

Visualize the Full Directory Tree: `tree`

The `tree` command, available on most UNIX distributions, provides a crude representation of the directory tree beneath your current location:

```
> tree
.
├── TMP
│   ├── test1.txt
│   └── test2.txt
├── TMP2
│   └── test2.txt
└── TMP3
    ├── test1.txt
    └── test2.txt
```

Important note: Avoid running `tree` at the root directory of your system, as it may produce a massive amount of output. Only use it where it's relevant.

If `tree` isn't available, you can solve the problem the UNIX way:

1. Google your question, e.g., "UNIX tree command not found."
2. Find a relevant discussion on a reputable platform like Stack Exchange: Tree command not found.
3. Copy and paste the provided macro command, aliased as `tree2` to avoid conflicts with the `tree` command.

In the right-hand side of the alias, two UNIX commands are given. First, `find` is used to locate folders and files.

```
> alias tree2='find . -print | sed -e "s;[^/]*/;|____;g;s;____|; |;g"'
> tree2
.
|____TMP2
| |____test2.txt
|____TMP3
| |____test1.txt
|____TMP
| |____test1.txt
```

Move or Rename Files/Folders: `mv`

The `mv` command serves a dual purpose: it moves files and folders. Importantly, if the source and destination folders are the same, it effectively renames the item.

```
> mkdir TMP  # Create a dummy void folder
> mkdir TMP2 # Create another dummy folder
> touch TMP/test1.txt # Create a dummy file
> tree
.
├── TMP
│   └── test1.txt
└── TMP

3 directories, 1 file
> mv TMP/test1.txt TMP2/test_02.txt
> tree
.
├── TMP
└── TMP2
    └── test_02.txt

3 directories, 1 file
> mv TMP2/test_02.txt TMP2/test2.txt
> tree
.
├── TMP
└── TMP2
    └── test2.txt

3 directories, 1 file
```

Remove Files: `rm`

Finally, we come to the most powerful and potentially dangerous command: `rm`. In its simplest form, it goes like this:

```
> tree
.
├── TMP
└── TMP2
    └── test2.txt

3 directories, 1 file
> rm TMP2/test2.txt
> tree
.
├── TMP
└── TMP2

3 directories, 0 files
```

By default, `rm` doesn't remove folders, only files. You'll need to use the `-r` (recursive) flag to delete an entire folder and all of its contents.

```
> rm -r TMP2
>
```

The specific case of "rm -rf *".

In the realm of the UNIX command line, there exists a seemingly innocuous yet incredibly powerful command - "rm -rf *" To the uninitiated, it may appear deceptively simple, but its potential for destruction is monumental.

Dissecting the Command:

- "rm" stands for remove, a directive to delete files or directories.
- "-rf" comprises two flags: "-r" for recursive and "-f" for force. Together, they signify removing everything beneath the specified point, annihilating any barriers or safeguards that would normally prevent such actions.

The wildcard "*", which in this context means "everything here," amplifies the command's reach. It aims to obliterate every file and directory within the current location without hesitation or confirmation. The consequences of a misstep with this command are profound.

The Perils of Precision:

In graphical user interfaces, unintentional mass deletions are exceedingly rare. Each cleaning operation is shielded by layers of point-and-click actions, necessitating deliberate effort to reach the folder or file in question.

However, the command-line interface operates with surgical precision. Swift jumps through the directory hierarchy are facilitated by simple commands, but therein lies the danger. A single typo while navigating can transform a routine task into a catastrophic blunder.

An Illustrative Mishap:

Consider this scenario:

```
>tree
.
├── IMPORTANT
└── CRITICAL
└── DUMMY
└── STUPID
└── TEMPORARY
> cd C\t  # Employing autocompletion to reach CRITICAL
CRITICAL> rm -rf *  # Cleansing the contents of CRITICAL
CRITICAL> cd ..
> cd S\t  # Utilizing autocompletion to access STUPID
CRITICAL> rm -rf *  # Purging the contents of STUPID
CRITICAL> cd ..
> cd Y\t  # Aiming for TEMPORARY but making a typo (T->Y)
-bash: cd: Y: No such file or directory
> rm -rf *  # Eradicating TEMPORARY, CRITICAL, IMPORTANT
```

In this example, a simple typographical error led to the unintentional removal of more than anticipated.

The "Killer Script" Conundrum:

These accidents are not isolated incidents. A common scenario is the "killer script." A user embeds a command akin to "rm -rf *" within a script designed to streamline a complex task. While the script may execute flawlessly under most circumstances, unanticipated failures in navigation can result in significant, and often hidden, damage.

A Word of Caution:

In essence, the command "rm -rf *" is a double-edged sword, offering unparalleled power but demanding meticulous precision. Proceed with caution, wield it judiciously, and always double-check your intentions. One small slip can trigger a cascade of irreversible consequences.

UNIX CLI Part 3/4: File Edition & Your First UNIX Script

In this part of our UNIX Command Line Interface (CLI) tutorial, we will delve into file editing and introduce you to your first UNIX script.

Text editor

When operating on local machine, you can easily use GUI interface for editing your text. As an example you should be able to use either `gedit` or `vscode` to launch your first script.

```
> code hello.sh
```

or

```
> gedit hello.sh
```

Your terminal will switch to a textual window where you can try a simple "Hello, World!" script. In the UNIX Shell, text is printed using the `echo` command.

```
echo "Hello, World!"
```

Once you've saved the file (`ctrl +s`), check for its presence and then ask the terminal to run this script with the `source` command:

```
> source hello.sh
Hello, World!
```

UNIX CLI part 4/4 : Test your basics

In this final section, we'll put your newfound UNIX command-line skills to the test. We'll use a script called `dungeon.sh` to create a random file structure resembling a dungeon with monsters and loot. Each time you run the script, it will generate a unique dungeon for you to explore.

The `dungeon.sh` script is available in the directory of your teacher

```
/home/newton/ienm2021/chabotv/COURS_CS
```

Now, let's embark on your UNIX adventure! Here is your quest:

1. Create a folder `Cours_CS` and a subfolder `Test_unix` and move inside it
2. Copy the file `dungeon.sh` from your teacher repository (`/home/newton/ienm2021/chabotv/COURS_CS`)
3. Try to execute the script
4. If needed, change the access right to the file (via `chmod` command) so the file can be executed.
5. Count how many `goblins` files there are in the dungeon
6. Find all the `prince` and `princess` occurrences in the files
7. Delete the dungeon - without wiping out the script `dungeon.sh` !
8. restart with a new dungeon.

Introspective commands

Now, let's finish with some introspective commands to understand your UNIX environment:

- **What's happening here?**: `w` displays who is logged in and what they are doing:

```
>w
 6:50  up 4 days, 20:15, 7 users, load averages: 1.49 1.73 1.74
USER     TTY      FROM              LOGIN@  IDLE WHAT
dauptain console  -                Wed10   4days -
dauptain s000     -                Wed10   22:23 -bash
dauptain s001     -                Wed10   22:23 /usr/bin/less -is
dauptain s002     -                Wed10   22:23 -bash
dauptain s003     -                Wed10   22:23 -bash
dauptain s004     -                Wed10   22:23 -bash
dauptain s007     -                Sun08       - w
```

- What is this machine? `uname -a` : Print operating system name with all (`a`) usual options. This will give you the operating system version, the machine network name, the type of processors. These infos are often asked if you file a bug.

```
Darwin eldarion-macbookair-coop.home 21.6.0 Darwin Kernel Version 21.6.0: Thu Jul  6 22:19:54 PDT 2023; root:xnu-8020.240.18.702.13~1/RELEASE_ARM64_T8101 arm64
```

- Who am I ? `whoami` : display effective user id. It is an old command, you should prefer `id -p` which also tells you what permission groups you are allowed to access (but its name is less fun)

```
>whoami
dauptain
id -p
uid     dauptain
groups  staff everyone localaccounts  admin access_remote_ae
```

- How big it is ? `du -sh` : display disk usage statistics (`du`), with a single entry for each specified file (`s`) using “Human-readable” output. (`h`)

```
>du -sh
129M    .
>du -sh .
129M    .
>du -sh FortranModernisationWorkshop.pdf Organizational_Analysis_Book_2016.pdf
5.9M    FortranModernisationWorkshop.pdf
123M    Organizational_Analysis_Book_2016.pdf
>du -sh *
108K     Decision Making.docx
228K     Decision Making.pdf
5.9M    FortranModernisationWorkshop.pdf
123M    Organizational_Analysis_Book_2016.pdf
4.0K    toto.txt
```

- what were my last commands? The `history` command gives you the chronological series of commands you already typed.

```
>history
(...)
  513  cd ../
  514  rm -rf TMP/
  515  ls
  516  ls *.py
```

On most terminals, the arrow keys up and down will recall the previous commands on your command line. For example Arrow-Up,Enter will repeat your last command in just two keys.

Takeaways

Congratulations! You've completed this comprehensive UNIX CLI tutorial, and you're now equipped with valuable skills to navigate and utilize a UNIX terminal effectively. Here are the key takeaways:

- Commands like `ls`, `cd`, `pwd`, `cat/head/tail` serve as your foundational tools for navigation and exploration in the UNIX environment.
- Wildcards (`*`, `?`) and autocompletion (tab key) are two important accelerators when typing commands
- The history of your terminal, recalled by `history` Arrow-up/down, is a more global accelerator. It is also a very useful log.
- File management is done with commands such as `cp`, `mv`, `mkdir`, and `rm`. However, exercise caution when wielding the power of `rm` to prevent accidental data loss.
- Recurrent operations can be saved in aliases (`alias`) for short sequences, and in files, names shell-scripts, for long sequences.

With these newfound skills and knowledge, you're well on your way to mastering the UNIX command-line interface, opening up a world of possibilities for efficient file management, text editing, and system navigation. Continue to explore, experiment, and grow your proficiency in the UNIX environment. Happy coding!