

# Introdução a Linux

## O Shell do Linux

→ o shell é um programa que recebe comandos do teclado do utilizador e os passa ao sistema operativo para execução. Ao executar comandos no shell podemos realizar tarefas como:

- navegar entre diretórios;
- gerir ficheiros e pastas;
- instalar ou remover programas;
- executar scripts e programas.

- Existem vários programas de shell disponíveis para Linux. Para este curso vamos utilizar o Bash. O ficheiro /etc/shells contém a lista de shells instaladas num sistema.
- O terminal é um programa que abre numa janela e permite aos utilizadores interagir com o shell.

## Comando Man

→ utilizado para abrir as páginas de manual de comandos e programas no Linux. Estas páginas contêm informações detalhadas sobre como utilizar o comando, opções disponíveis e exemplos.

Ex:

\$ man ls → abre a página de manual para o comando ls.

\$ man cp → abre a página II II II cp.

/palavra : Procura a palavra "palavra" no manual.

m : Vai para a próxima ocorrência da pesquisa anterior.

N : Vai para a ocorrência anterior da pesquisa.

g : Vai para o inicio da página de manual.

G : Vai para o fim da página de manual

q : Vai sair da página do manual.

## Operações básicas de diretório:

(p/ ver onde estou no Sistema de arquivos)

**Pwd** (print working directory): exibe o diretório atual em que me encontro.

**ls** (list): lista o conteúdo do diretório atual, incluindo arquivos e pastas.

Comando **ls**:

**ls -a**: lista todos os arquivos, incluindo os ocultos (aqueles que começam por um ponto).

**ls -l**: fornece uma listagem detalhada, incluindo permissões de arquivo, número de links, proprietário, grupo, tamanho e data de modificação.

ou **ls -a -l** ou **ls -al**

**mkdир -p**: allows the creation of a sequence of folders and subfolders with a single command line.

**mkdir**: para criar um novo diretório

**rmdir**: para remover um diretório vazio

**cd**: utilizado para mudar para um diretório específico.

**cd ~**: para retornar ao diretório **home**

**cd ..**: para retornar para o diretório pai

**cd ../../**: para voltar dois níveis acima

**Caminho absoluto**: começa com / e fornece localização completa a partir da raiz dos arquivos

**Caminho relativo**: começa do diretório atual e refere-se à localização do destino a partir daí.

**touch my\_file**: cria um arquivo vazio chamado my\_file. (cria um ficheiro)

**echo "Hello World" > hello.txt**: cria o arquivo hello.txt com o texto "Hello World".

**echo "second line" >> hello.txt**: anexa "second line" ao final do arquivo hello.txt.

↳ "pipe"

## Operações básicas com arquivos

cat hello.txt : exibe o conteúdo do arquivo hello.txt

wc hello.txt : mostra o n.º de linhas, palavras e caracteres em hello.txt

wc -l hello.txt : conta e exibe apenas o n.º de linhas

wc -w hello.txt : conta e exibe apenas o n.º de palavras

Path

relativo → quando estás a dar path p/ um ficheiro ou pasta a partir de onde estás no momento

Path absoluto →

" " " " " " a partir da root e descrever todo o caminho desde a root até onde queremos chegar.

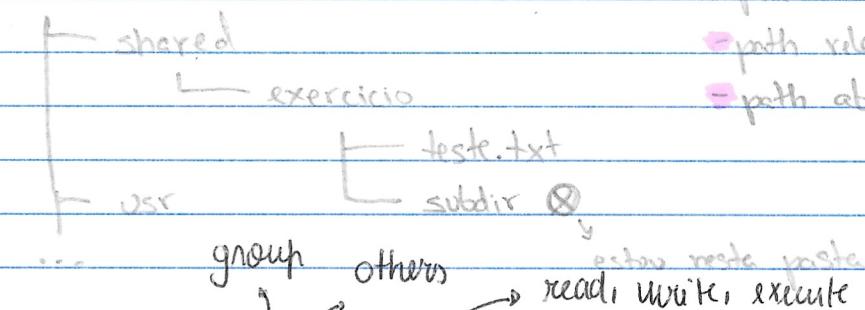
home / Inu / shared

Imprimir teste.txt:

- path relativo: cat .. / teste.txt

- path absoluto: cat ~/shared/exercicio/teste.txt

3a)



chmod go-rwx Exercicio3

para mudar permissões

(estou a remover todas as permissões do group e de others)

ctrl d : Volta ao inicio (abandonar)

## File and directory operations

cp : copia um ficheiro ou pasta.

↳ -r para copiar recursivamente também os subdiretórios;

mv é em vez de copiar, apenas move e apaga a "source". Também usado para renomear um ficheiro.

rm : remove ficheiros permanentemente.

## Input / output redirection

stdin (0) : standard input ( keyboard );

stdout (1) : standard output ( terminal );

stderr (2) : standard error ( terminal );

Ex:

```
$ wc -l < file-list.txt > number.txt
```

↳ conta o n.º de linhas no arquivo file-list.txt e grava esse número no arquivo number.txt

```
$ echo -e "third \nfirst \nsecond" | sort > sorted-lines.txt
```

↳ lê as palavras third, first e second, ordena-as em ordem alfabética e grava o resultado em sorted-lines.txt

## Paging through long text files / outputs

more : só permite avançar para a frente;

less : permite avançar e voltar páginas atrás (+/ctrl)

## Display a partial content of a file / output

head : para ver as primeiras m linhas de um ficheiro

```
$ head -m 15 Readme
```

tail : para ver as últimas m linhas de um ficheiro

```
$ tail -m 15 Readme
```

## File / Directory ownership and permissions

User	Group	Other
rwx	rwx	rwx

para mudar  
permisões

Write : create/delte new entries;

read : list (ls) the contents;

execute: enter (cd) the directory;

chmod go-rwx Executar

ls -l → para ver quem tem  
permisões

`whoami` : retorna o nome do usuário atual da sessão

`groups` : retorna o grupo ao qual o usuário pertence

`chown user:group` : altera o usuário e/ou grupo de um arquivo ou diretório

Ex: `chown jorge:admins arquivo.txt`



faz com que usuário `jorge` seja o proprietário  
do arquivo `.txt`, e o grupo `admins`  
seja o grupo ao qual o arquivo pertence

-`r` para recursividade



`chown -r jorge:admins /pasta`



muda o proprietário e o grupo de todos os  
arquivos e subdiretórios dentro de `/pasta` para  
`jorge` e `admins`



→ Apenas o superusuário (`root`) tem permissão para alterar  
o proprietário de arquivos que pertencem a outros usuários.

Searching for pattern

`grep` : busca um padrão específico dentro de arquivos.

Ex: `$ grep "test" .. /exercicio/test1.txt`



procura a palavra "test" no arquivo `test1.txt`.  
O comando vai imprimir "This is a test file"

`grep -i` : ignora a diferença entre maiúsculas e minúsculas

`grep -v` : exibe as linhas que NÃO contêm o termo "test".

`find` : localiza arquivos e diretórios com base em atributos específicos,  
como nome, tamanho, tipo, etc..

Ex: `$ find /home/luis -perm 644`



→ este comando encontra arquivos no diretório `/home/luis`  
com a permissão 644 (permite leitura e escrita para o proprietário, e

Ex: \$ find /home/user -name "\*.txt" -size +1M

↳ encontra arquivos no diretório /home/luser com o nome que termina com .txt e com tamanho maior que 1Mb.

Ex: \$ find /tmp -name "\*.log" -exec rm {} \;

↳ encontra todos os arquivos no diretório /tmp que terminam com .log e executa o comando rm para removê-los.

**File** : utilizado para determinar o tipo de um arquivo e exibir informações sobre o seu conteúdo.

Ex : \$ file myfile  
myfile : ASCII text  
format

Ex: \$ file test.tar.gz  
test.tar.gz: gzip compressed data, from Unix

CRLF line terminator

Em sistemas Windows, os editores de texto adicionam um caractere Carriage Return (CR) seguido por Line Feed (LF) ao final de cada linha de texto. Isso é conhecido como CRLF (caractéres de terminação de linha), enquanto no Unix/Linux, apenas o Line Feed (LF) é usado para terminar as linhas de texto.

Lime Feed (LF) é usado para terminar as linhas de texto.

O comando dos 2 unix converte arquivos de texto de formato  
CR LF para LF (unix/linux)

## Compression and file packaging

Zip : Unia pacotes comprimidos de arquivos e diretórios.

Ex: zip arquivo zip arquivo1 arquivo2

nome do arquivo  
computado que será  
gerado arquivos ou diretórios que  
serão computados

```
$ zip -r archive.zip ./*
```

incluir todos os arquivos e  
dicionários no diretório atual

adding: .\task1.txt (deflated 60%)  
adding V: .\mydir / (deflated 0%)

O arquivo test1.txt foi compactado com uma taxa de compressão de 6

O diretório myclns foi adicionado, mas não contém arquivos que precisam ser comprimidos, então a tarefa de compressão é só.

**Umzip**: utilizado para extraír o conteúdo de arquivos .zip e também para listar o conteúdo de arquivos compactados.

Ex : \$ unzip archive.zip

Ex: \$ **unzip -l** archive.zip

↓ verifica o que há dentro do arquivo zip sem o descomprimir.

## Archiving

**Tar**: utilizado para empacotar vários arquivos e diretórios em um único arquivo de arquivo de tar (ou tarball), muitas vezes com a opção de compressão (gzip ou bzip2). Útil para arquivar e compartilhar grandes quantidades de dados de forma compacta.

Ex: tar [options] dest-file src1 src2

arquivo de destino

o que se deseja adicionar ao

arquivo destino

### Options:

- C: cria um novo arquivo .tar
- N: extrai o conteúdo de um arquivo .tar
- f: especifica o nome do arquivo destino
- v: exibe os arquivos sendo processados
- z: compacta com gzip
- j: compacta com bzip2
- a: adiciona ao arquivo .tar existente
- t: lista o conteúdo de um arquivo .tar

## The process abstraction

Processo → instância de um programa em execução. Meio principal pelo qual o sistema operacional oferece funcionalidades de multiprogramação e multitarefa.

### Importância:

Multiprogramação / multitarefa: os processos permitem que o sistema operacional mantenha a ilusão de que múltiplos programas estão a ser executados simultaneamente. Na realidade, o SO (sistema operativo) aloca uma fatia de tempo do CPU (central processing unit) para cada processo, numa sequência rápida, dando a sensação de execução paralela.

### Estrutura de um processo:

#### 1. Instruções de programa ("program text")

↳ representa o código do programa que será executado.

#### 2. Estado do CPU:

↳ inclui o conteúdo do programa (PC) que armazena o endereço da próxima instrução a ser executada.

↳ Registradores guardam <sup>já feito</sup> resultados temporários e resultados intermediários

↳ Endereços e outras informações que refletem o estado atual da execução

### 3. Estado da memória

→ cada processo tem o seu próprio espaço de endereçamento de memória, isolado dos demais processos, o que ajuda na segurança e estabilidade do sistema;

### 4. Outros recursos:

→ recursos adicionais que o processo pode estar a usar, como arquivos abertos, dispositivos de entrada e saída, conexões de rede...

## Process management

Nos SO's, sempre que uma aplicação é executada (seja a partir da linha de comandos ou da interface gráfica), é criado um novo processo. Esse processo é onde o ficheiro binário da aplicação é carregado, e, em seguida, executado.

Um sistema de multitarefas é aquela que permite que múltiplos processos operem aparentemente em simultâneo sem interferirem uns com os outros.

→ Isto acontece porque os processos podem alterar dinamicamente os seus estados

### Process state

Um processo passa por várias etapas durante o seu ciclo de vida, desde a sua criação até à sua finalização.

→ Cada etapa pode ser designada como um estado de processo.

→ Os processos mudam entre diferentes estados através de sinais e traps codificados nos próprios processos.

### Process management

- `ps aux` option lists all processes in the system

- `ps aux` option lists all processes, their status, and resource usage

`ps (process status)`: mostra o estado dos processos no sistema.

Por default, o comando `ps` lista apenas os processos que pertencem ao utilizador atual e que estão associados ao terminal atual.

Cada processo no sistema possui um identificador único (PID).

→ O processo inicial é normalmente carregado a partir de /sbin/init ou /usr/lib/systemd e é atribuído com o PID 1. Este processo inicial é

## Process hierarchy

Cada processo faz parte de uma hierarquia, onde processos pai podem gerar processos filho.

→ Os processos filho herdam variáveis de ambiente, permissões e outras características do processo pai, garantindo que a estrutura e as permissões se mantêm consistentes entre os processos relacionados.

**PsTree** : mostra as relações de parentesco entre os processos em execução.

Permite compreender facilmente quais processos foram iniciados por outros, facilitando a análise e gestão dos processos no sistema.

## Signals

Os sinais são interrupções de software enviadas a um processo para indicar que ocorreu um evento importante.

→ apenas o proprietário de um processo ou o utilizador root têm permissão para enviar sinais a um processo.

Comportamento dos processos ao receberem sinais: quando um processo recebe um sinal, ele pausa temporariamente o que estava a fazer, executa uma função chamada signal handler.

Assim que a execução do handler termina (seja, a função devolve controlo), o processo retoma a atividade anterior.

No entanto, se o sinal recebido tiver como objetivo terminar o processo, este não retornará a execução e será encerrado.

**Kill** : utilizado para enviar um sinal a um processo.

Existem diversos sinais, cada um com um propósito específico.

**Opción -l** permite listar todos os sinais disponíveis.

→ Para enviar um sinal a um processo é necessário fornecer o PID do processo como argumento. ↗

→ Por padronização, o sinal enviado é o SIGTERM, que pede ao processo para terminar de forma ordenada. Este sinal permite que o processo encerre as suas operações corretamente, executando o seu próprio handler para tratar a terminação.

→ O **SIGKILL**, por outro lado, força a terminação imediata do processo e é gerido diretamente pelo Kernel. Como tal, o processo não consegue capturar ou lidar com este sinal, sendo imediatamente encerrado.

Há outros atalhos que são interpretados como pedidos para enviar sinais ao processo.

**CTRL + C**

↳ envia um **SIGINT** (interrupt) ao processo em execução. indica ao processo que deve interromper a sua execução, permitindo-lhe encerrar de forma controlada.

**CTRL + Z**

↳ envia um **SIGTSTP** (terminal stop) ao processo, colocando-o em modo suspensão. O processo pode ser restaurado posteriormente, através do comando **fg** para trazê-lo de volta para o primeiro plano ou **bg** para o colocar em segundo plano.

**CTRL + I**

↳ envia um **SIGABRT** (abort) ao processo. Este sinal ationa, por defeito, a chamada do sistema **abort()**, que termina o processo abruptamente e gera um core dump (registo do estado da memória), útil para diagnóstico de falhas.

Execução de comandos em segundo plano

↳ Permite que o comando seja executado em paralelo com o programa chamador, sem que este suspenda a sua execução.

→ Para executar um comando em segundo plano, basta adicionar o caractere **&** no final da linha de comando

\$ sleep 5000 & → O comando sleep foi iniciado em 2º plano e irá dormir durante 5000 segundos antes de terminar  
[1] 156903

↳ PID do processo que está a ser executado  
id de trabalho do comando em segundo plano

Se o processo "sleep" estiver a ser executado estaria no estado 5 "interrupível", ou seja, "adormecido".

→ após executar o comando em segundo plano, pode-se verificar os processos em execução com o comando `ps`.



lista os processos em execução, incluindo os que estão em 2º plano

`jobs`

o número do trabalho é uma forma simples e prática do shell gerir os processos, sem que tenha de lidar diretamente com os PID's.

Um job é uma forma específica do shell para gerir processos em segundo plano. Cada comando ou processo executado em segundo plano é considerado um job.

`jobs` : permite listar os jobs ativos numa sessão, mostrando as informações sobre os processos que estão a ser executados em segundo plano.

Ex: `$ jobs`

[1]+ Running sleep 5000 &

É útil para saber que processos estão ativos ou suspenso na minha sessão!

o comando `sleep 5000 &` foi executado em segundo plano e o shell mostra que está a correr como um job com o número [1].

`jobs -l` : listar jobs com mais detalhe

`fg & foreground` : traz um processo que está em segundo plano (background) para o primeiro plano (foreground), permitindo que o utilizador interaja diretamente com ele.

`[fg 1]` traz o trabalho identificado como [1] (1º trabalho listado) para o primeiro plano

`CTRL + Z` : pausa o processo atual em 1º plano e coloca-o no estado `Stopped` (`SIGTSTP`)

após pausar o processo, ele pode ser enviado para segundo plano usando `bg`, ou retomado no 1º plano com `fg`

► "pausa" → não está em execução nem em 1º plano nem em 2º plano.

As variáveis de ambiente armazenam informações definidas pelo utilizador ou pelo sistema e podem ser usadas dentro do shell.

Para criar uma variável de ambiente, basta atribuir um valor a um nome, sem espaços ao redor do símbolo = :

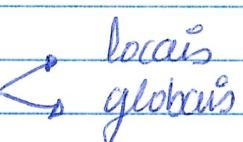
\$ myfirstVar = "SCOMRED"  
    |                                |  
    nome                            valor

Para usar o valor armazenado numa variável, colocamos um \$ antes do nome da variável

\$ echo \$ myfirstVar  
→ a saída será SCOMRED

PATH : utilizada para localizar os comandos executáveis.

O \$PATH é uma lista de diretórios separados por dois pontos (:). Quando executo um comando, o shell procura por ele nos diretórios especificados no \$PATH, da esquerda para a direita, e executa o primeiro executável correspondente encontrado.

As variáveis de ambiente podem ser  locais globais

Locais : são conhecidas apenas pelo shell que as criou;

Se iniciarmos um novo processo ou programa a partir deste shell, as variáveis locais não serão visíveis para o processo filho.

Export : para tornar uma variável de ambiente disponível para subprocessos (ou seja, global).

→ quando exportarmos uma variável ela fica visível para todos os processos filhos iniciados a partir do shell atual.

Ex: \$ export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk /...

→ aqui JAVA\_HOME é definido como o caminho para o JAVA e exportado, o que o torna acessível a todos os processos do shell atual.

→ As variáveis de ambiente definidas diretamente no terminal são temporárias. Se o terminal for fechado, essas variáveis já não estarão mais definidas.

Definir uma variável de ambiente permanentemente:

→ adicionar-la ao arquivo ~/.bashrc. Esse arquivo é executado todas as vezes que uma nova sessão de shell é iniciada, carregando as variáveis definidas nele.

1. Abrir o arquivo .bashrc no diretório home do utilizador  
nano ~/.bashrc

2. Adicionar a variável ao final do arquivo

export JAVA\_HOME=/usr/lib/jvm/java-11-open...

3. Para aplicar imediatamente as mudanças no terminal atual  
source ~/.bashrc

ou → adicionar a variável diretamente no terminal usando  
echo e >>, que adiciona o comando ao final do arquivo  
~/.bashrc

\$ echo "export JAVA\_HOME=/usr/lib/jvm/..." >> ~/.bashrc

## Text editors

Nano : pequeno, simples e amigável

Vim : poderoso, que pode ser usado para editar todos os tipos de texto  
emacs : parte do GNU Project

# Basic Bash scripting

Um script de shell permite automatizar e simplificar tarefas no Linux.

\$ nano myscript.sh

#!/bin/bash

→ shebang que indica o interpretador Bash

echo "Hello World!"

→ comando que imprime uma mensagem

\$ chmod u+x myscript.sh

→ concede permissão de execução ao script

\$ ./myscript.sh

→ executa o script

Hello world!

## Controlo de fluxo

if [ condição ]; then

elif [ outra condição ]; then

else

fi

Var = 2

if [ \$VAR -eq 1 ]; then

echo "A variável igual a 1"

elif [ \$VAR -gt 1 ]; then

echo "A variável maior que 1"

else

echo "A variável menor que 1"

fi

## Loops (while, until, for)

number = 0

while [ \$number -lt 10 ]; do

echo "Number = \$number"

number=\$((number + 1))

done

While enquanto

for true

Until enquanto

for false

`#!/bin/bash` → indica que será interpretado pelo Bash

`if [ -f ~/.bashrc ]; then` → verifica se o arquivo .bashrc existe no diretório home do utilizador  
  `echo "you have a .bashrc. Things are fine"`  
`else`  
  `echo "yikes! you have no .bashrc!"`  
`exit` → encerra o script  
`fi`

`count = 0` → irá começar a contagem de palavras  
`for word in $(cat ~/.bashrc); do` → lê cada palavra no .bashrc usando  
  `count = $((count + 1))` → para cada palavra aumenta count em 1  
  `echo "Word $ count ($word)"`  
  contém `$(echo -n $word | wc -c)` characters

done

exibe a palavra e o seu nº caracteres  
-n remove a linha do echo, para que  
wc -c conte apenas os caracteres da  
palavra

A Bash Script is an executable text file  
that starts with the `#!/bin/bash` line. The  
Linux Kernel will then recognize this file as  
an interpreted program to be executed by  
`/bin/bash` (the interpreter).

mano : to create a file, with a certain name.  
→ text editor

`$ chmod u+x myscript.sh`  
↳ give it execute permissions

`$ ./myscript.sh`  
↳ test the script