Índice

Introdução ao linux

O linux é um kernel de sistema operativo *open source* que serve como base para diferentes sistemas operativos, desde computadores até televisões e máquinas de café. Foi projetado pelo finlandês Linus Torvalds em 1991 para ser um sistema altamente personalizável. Entre as características chave do Linux está o facto de ser *open source*, sendo distribuído sob a licensa *GNU General Public License (GPL)*, o que significa que os usuários são livres de utilizar, modificar e distribuir o software para qualquer finalidade, desde que estas modificações também sejam distribuidas com a mesma licensa. Assim, o linux atraiu uma grande comunidade de desenvolvedores e entusiastas da computação, o que levou à rapida evolução deste sistema.

Comandos básicos de navegação

Certamente que uma das primeiras coisas que vos passa pela cabeça ao pensar em usar um terminal é "porquê perder tempo a aprender comandos se podemos fazer tudo através de uma interface gráfica?". E é verdade, quase tudo o que dá para fazer pelos comandos dá para fazer por uma interface gráfica, e o intuito de aprender a utilizar o terminal é complementar a interface gráfica e não substituí-la. Pensem no terminal como mais uma ferramenta para a vossa caixa de ferramentas. Quanto mais completa a caixa de ferramentas, melhor.

Para começar, estes são os comandos básicos de navegação pelas diversas pastas e arquivos do sistema.

Comando	**Descrição**	**Uso**
pwd	print working directory	pwd
man	manual do comando	man [command]
ls	listar arquivos e pastas	ls [options] [directory]

Comando	Descrição	Uso
cd	change directory	cd [directory]

Caminho relativo vs caminho absoluto

Ao navegar através do terminal um dos conceitos mais importantes é o de caminho relativo e absoluto

Caminho absoluto

Representa a localização exata do ficheiro/diretório, não dependendo do diretório atual.

Exemplo:

`/home/acm/documentos/workshop.txt`

> Caminhos absolutos começam sempre com `/``.

Caminho relativo

Representa a localização do ficheiro/diretório em relação ao diretório atual.

Exemplo:

`../downloads/comandos.txt`

Ao lidar com caminhos há 3 simbolos que são usados frequentemente:

- `~` : Representa o diretório Home do utilizador atual

> `cd ~` ou simplesmente `cd` leva o utilizador para a pasta Home.
>
> `cd ~/documentos` refere-se à pasta documentos dentro da paste Home.

- `.` : Representa o diretório atual

> Se a pasta atual é `/home/acm/documentos` então `.` é um *alias*.

- `..` : Representa o diretório acima do atual

> Se a pasta atual é `/home/acm/documentos` então `..` representa `/home/acm` .

Através destes comandos é possível navegar livremente por todas as pastas do sistema (desde que o utilizador tenha as permissões necessárias).

Criação e gestão de arquivos

Uma das partes mais importantes de utilizar um terminal é a criação e gestão de arquivos, e estes são os comandos mais utilizados para esse efeito. É necessário ter cuidado ao realizar ações destrutivas (apagar arquivos/pastas) pois não é possível desfazer alterações.

Comando	Descrição	Uso
mkdir	criar diretório	mkdir [diretório]
touch	criar arquivo	touch [arquivo]
rm	apagar arquivo/diretório	rm [options] [file]
mv	mover / mudar o nome	mv [source] [destination]
cp	copiar	cp [source] [destination]

Manipulação de arquivos

Editores de texto são ferramentas essenciais para criar, editar e manipular ficheiros. Editores de texto de terminal, como o nome sugere, são feitos para serem utilizados através de uma interface de linha de comandos *(CLI)*. Entre os editores mais conhecidos estão o *nano* e o *vim*. A principal diferença entre os 2 é que enquanto que o *nano* é mais focado em editar texto simples, o *vim* é mais apropriado para escrever código. Devido a esta diferença, é muito mais fácil aprender a utilizar o *nano*.

Nano

Para abrir o *nano* é muito simples, pois vem pré-instalado na maioria dos sistemas Unix (Linux e MacOS). Assim, basta escrever `nano [arquivo]` para abrir.

Após terem o *nano* aberto há alguns conceitos muito importantes que facilitam o seu uso.

- Para navegar pelo texto utiliza-se as setas do teclado
- Para saltar várias linhas de uma vez pode-se utilizar as teclas `Page Up` e `Page Down` ou `Ctrl + ↑` e `Ctrl + ↓`
- Para ir para o início ou fim da linha basta clicar em `Home` e `End`
- Para ir para o início ou fim do arquivo utiliza-se `Ctrl + Home` e `Ctrl + End`
- Para copiar texto utiliza-se `Alt + 6`, para cortar `Ctrl + K` e para colar `Ctrl + U`
- Para guardar utiliza-se `Ctrl + O` e sair `Ctrl + X`

Existem muitos outros comandos que podem ser consultados através do manual (`man nano`)

Vim

`WIP`

Permissões

No ecossistema Linux, a gestão de permissões desempenha um papel fundamental na segurança e no controle de acesso aos arquivos e diretórios. As permissões determinam quem pode ler, escrever e executar cada ficheiro, garantindo assim a integridade do sistema e dos dados.

Ao contrário de sistemas mais orientados para o usuário, o Linux utiliza um sistema de permissões baseado em três entidades principais: proprietário, grupo e outros. Cada arquivo e diretório possui permissões específicas atribuídas a cada uma dessas entidades, que podem ser alteradas para alcançar o nível desejado de segurança e privacidade.

Neste contexto, as permissões são uma parte essencial da administração de sistemas baseados em Linux e da garantia que apenas as pessoas e processos autorizados podem ler, alterar e executar determinados recursos. De seguida, iremos explorar mais detalhadamente como é que as permissões funcionam e como podem ser configuradas.

Para visualizar as permissões de um arquivo ou diretório basta utilizar o seguinte comando: `ls -l [arquivo (opcional)]`.

Exemplo:

```
acm@acm: ls -l workshop.txt
-rw-rw-r-- 1 henrique acm 6618 set 25 13:31 workshop.txt
```

Vamos agora ver uma desconstrução do *output*:

Output	Descrição
-rw-rw-r--	permissões
1	contagem de *hard links*
henrique	dono
acm	grupo
6618	tamanho
set 25	data
13:31	hora
workshop.txt	nome

Daqui, as partes mais importantes são as permissões, o dono e o grupo.

Vamos agora focar-nos nas permissões. À primeira vista pode parecer uma sequência de 10 caracteres sem sentido, mas na verdade é bastante simples.

- O primeiro caracter diz se é um arquivo (`-`) ou um diretório (`d`)
- Os próximos 9 podem ser divididos em 3 partes iguais, a primeira para o dono, a segunda para o grupo e a terceira para os restantes utilizadores.
- Cada uma destas partes é ainda dividida em 3 da seguinte forma:

  > A primeira (`r` *read*) representa a permissão de leitura
  >
  > A segunda (`w` *write*) representa a permissão de escrita

> A terceira (`x` *execute*) representa a permissão de executar
>
> A ausência de cada permissão é representada por `-`

- Assim, `-rw-rw-r--` é interpretado como:

`-`	`rw-`	`rw-`	`r--`
ficheiro	o utilizador/dono pode ler e escrever	o grupo pode ler e escrever	os restantes apenas podem ler

> `rwx` significa permissão total e `---` significa sem permissões

Como alterar as permissões?

Para alterar as permissões utiliza-se o comando `chmod` e utiliza-se da seguinte forma:

```
chmod [permissão] [arquivo]
```

Aqui, a parte da permissão é dividida em 3 partes:

1. Quem?
2. Revogar ou conceder?
3. Que permissão?

Vamos agora analisar cada uma das partes.

- Para a primeira parte utiliza-se o `ugoa` (user/owner, group, others, all).
- Para decidir entre revogar e conceder a permisão usa-se `-` para revogar e `+` para conceder
- Para escolher a permissão utiliza-se `r`, `w`, `x`

Exemplos:

- `chmod g+wx script.sh`

> Dar permissão de escrita e execução ao grupo

- `chmod u-w workshop.txt`

> Revogar a permissão de escrita ao utilizador/dono

- `chmod a+rwx programaincrivel.py`

> Conceder todas as permissões a toda a gente

Apesar de esta forma ser bastante intuitiva, também há outra maneira de alterar as permissões utilizando o `chmod`

Reparem que, para cada uma das 9 permissões, podemos atribuir o valor de `1` ou `0`, podendo assim converter o conjunto de caracteres para um número binário de 9 digitos.

Exemplo:

`rwxrw-r--` pode ser convertido em `111110100`

Se representarmos este número em octal ficamos com 3 digitos, sendo que cada um coincide com cada uma das 3 entidades.

`111110100` em octal fica `764`

Para facilitar a conversão podem utilizar esta tabela:

Binário	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Desta forma, pode-se substituir o conjunto de caracteres por um número de 3 digitos, facilitando a escrita do comando.

Exemplo:

- `chmod 764 script.sh`

> Dar permissão total ao utilizador/dono, permissão de escrita e leitura ao grupo e permissão de leitura aos restantes.

- `chmod 000 workshop.txt`

> Revogar todas as permissões

- `chmod 777 programaincrivel.py`

> Conceder todas as permissões a toda a gente

Filtros e *regular expressions*

No mundo *Linux*, filtros e *regular expressions* são ferramentas poderosas para processamento e edição de texto, bem como extração de dados.

Filtros

Filtros são programas que aceitam texto, processam, e transformam noutro texto. Normalmente são utilizados em cadeia *(pipeline)* para realizar uma sequência de operações.

Vamos agora ver alguns exemplos:

- `head [nº de linhas] [arquivo]`

> É utilizado para mostrar as primeiras n linhas (10 caso não seja especificado) de um arquivo.
>
> Exemplo:
>
> `head -3 workshop.txt`
>
> ```
> Introdução ao linux
> Comandos básicos de navegação
> Criação e gestão de arquivos
> ```

- `tail [nª de linhas] [arquivo]`

> É o contrário do `head`, pois imprime as últimas n linhas.
>
> Exemplo:
>
> `tail -3 workshop.txt`
>
> ```
> Gerenciamento de processos
> Remote shell (SSH)
> Package manager (apt)
> ```

- `sort [opções] [arquivo]`

> Imprime o conteúdo do arquivo de acordo com as opções especificadas (por defeito ordena alfabeticamente).
>
> Exemplo:
>
> `sort nomes.txt`
>
> ```
> Ana
> Bernardo
> Carlos
> Guilheme
> João
> ```

Grep e *regular expressions*

Pipeline

Em Linux, pipelining é o ato de pegar no output de um comando e enviar para outro, criando assim uma cadeia de processamento. Para esse efeito, utiliza-se o caracter `|`.

Exemplo:

```
head -3 nomes.txt | sort -r
```

> Carlos
>
> Bernardo
>
> Ana

Um dos comandos mais utilizados para pipeline é `grep`, pois é utilizado para pesquisar, utilizando para isso *regular expressions (regex)*