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]
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]

Comando	Descrição	Uso
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`)

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

Pipeline e Redirecionamento

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

Redirecionamento

Normalmente, os comandos enviam o seu output diretamente para o terminal, no entanto às vezes é conveniente guardar o resultado num ficheiro. Para isso, podemos utilizar o caracter `>`.

Exemplo:

```
ls > output.txt
cat output.txt
```

> Desktop
>
> Downloads
>
> Documents
>
> Images
>
> Videos

Porém, caso o ficheiro de destino já exista, será sobrescrito, e para evitar isso podemos utilizar o caracter `>>`. Este, em vez de sobrescrever, adiciona o output ao ficheiro já existente.

Exemplo:

```
ls >> output.txt
cat output.txt
```

> gato
>
> cão
>
> Desktop
>
> Downloads
>
> Documents
>
> Images
>
> Videos

Também é possível fazer o contrário, utilizar um arquivo como input do comando, através do caracter `<`.

Exemplo:

```
wc < output.txt
```

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*

Grep

O comando `grep` é muito usado para pesquisar e filtrar data com base em padrões, como por exemplo pesquisar palavras específicas num documento.

Uso:

```
grep [opções] [padrão] [ficheiro]
```

Entre as várias opções temos -i para correspondência sem distinção entre maiúsculas e minúsculas e -r para pesquisa recursiva em diretórios.

O padrão corresponde a uma *regular expression* que iremos ver mais à frente.

Também é possível utilizar o pipelining visto anteriormente como input do comando.

Exemplo:

```
cat log.txt | grep "error"
```

> Este comando vai procurar pelo arquivo log.txt a palavra 'error'.

Regular Expressions

As expressões regulares, frequentemente abreviadas como "regex" ou "regexp," são padrões poderosos e flexíveis usados para corresponder e manipular texto. São uma sequência de caracteres que definem um padrão de pesquisa. As expressões regulares são usadas em várias linguagens de programação, editores de texto e ferramentas para tarefas como pesquisa, validação e extração de informações de texto.

Estes são os componentes de uma *regex*:

- `.` - um caracter
- `?` - o caracter anterior existe 0 ou 1 vezes
- `*` - o caracter anterior existe 0 ou mais vezes
- `+` - o caracter anterior existe 1 ou mais vezes
- `{n}` - o caracter anterior existe exatamente n vezes
- `{n,m}` - o caracter anterior existe pelo menos n vezes e no máximo m vezes
- `[bhp]` - o caracter é um destes (neste caso ou é b ou h ou p)

- `[^bhp]` - o caracter **não** é um destes (neste caso não é b nem h nem p)
- `[g-k]` - o caracter está entre g e k (neste caso pode ser g,h,i,j,k)
- `()` - agrupa o conjunto de caracteres num só
- `|` - operador de disjunção (**ou**)
- `^` - procura no início da linha
- `$` - procura no fim da linha

Para utilizar regular expressions com o grep é necessário utilizar a opção ***-E***

Vamos agora ver exemplos:

- `grep -E '[aeiou]{3,}' texto.txt`

> Retorna as linhas do arquivo que possuem uma sequência de 3 ou mais vogais

- `grep -E '(s.)$' texto.txt`

> Retorna as linhas do arquivo que acabam em 's.'

- `grep -E '^(A|O)' texto.txt`

> Retorna as linhas que começam com A ou O

- `grep -E '.*@.*(.com)' text.txt`

> Retorna as linhas que contêm endereços de email

Gestão de Processos

Listar processos

A gestão de processos em sistemas Linux é uma parte essencial da administração de sistemas operativos. Envolve a supervisão, controlo e monitorização de programas em execução, conhecidos como "processos". Estes processos podem ser aplicações, scripts ou até mesmo tarefas do sistema em segundo plano.

Para ver uma lista de todos os processos em tempo real, bem como o uso da RAM e do CPU utiliza-se o comando `top`. Outro comando bastante utilizado é o `ps aux`.

- `ps aux | grep nautilus`

> neacm 6245 14.1 1.5 3021052 215328 ? Sl 15:42 0:00 /usr/bin/nautilus --gapplication-service
>
> este output pode ser bastante confuso, por isso normalmente usa-se apenas `ps -A | grep nautilus` que dá o seguinte output:
>
> 6245 ? 00:00:10 nautilus

Terminar processos

Pode não acontecer muito frequentemente, mas às vezes os programas congelam e não conseguem terminar sozinhos. Nestes casos é útil saber terminar processos. Vamos ver um caso prático em que o firefox congela.

O primeiro passo é saber o id do processo (PID), e para isso já vimos que podemos utilizar o seguinte comando:

```
ps -A | grep firefox
```

> 5585 ? 00:00:12 firefox

Agora utilizamos o comando `kill` para terminar o processo:

```
kill 5585
```

No entanto, às vezes apenas utilizar `kill [PID]` não é suficiente, pois este apenas "pede" ao processo para terminar. Nestes casos utiliza-se `kill -9 [PID]` pois assim obrigamos o processo a terminar.

Secure Shell (SSH)

O SSH, ou Secure Shell, é um protocolo de rede amplamente utilizado para estabelecer conexões seguras e autenticadas entre computadores de uma rede, geralmente pela internet. Fornece uma maneira segura de aceder e controlar sistemas remotos e é uma ferramenta fundamental para administradores de sistemas, desenvolvedores e qualquer pessoa que precise de acesso remoto a servidores ou dispositivos.

Para utilizar SSH é bastante simples, basta utilizar o comando `ssh username@remote_host`. De seguida, terão de aceitar a *fingerprint* e escrever a password da máquina a que estão a tentar aceder (esta não é a maneira mais segura de fazer, devem utilizar um par de chaves para fazer o acesso, porém não vamos cobrir esse tópico aqui).

Além de permitir controlar um servidor remotamente, o SSH também possui muitas outras funcionalidades, como por exemplo abrir túneis para encaminhar o tráfego de internet, no entanto não iremos falar disso.

Package Manager

Os *package managers* são ferramentas fundamentais em sistemas Linux. Desempenham um papel crucial na instalação, atualização, remoção e gestão de software.A principal função de um *package manager* é simplificar o processo de instalação e manutenção de programas, garantindo que as dependências sejam atendidas e que o software seja mantido atualizado. Normalmente, diferentes distribuições de Linux utilizam diferentes *package managers*, e estes são alguns exemplos:

> `apt` em Debian
>
> `yum/dnf` em Fedora
>
> `pacman` em Arch

Vamos agora falar um pouco acerca do `apt` pois é o que está presente no Ubuntu.

- `sudo apt install [pacote]`

> Instala o pacote e todas as suas dependências

- `sudo apt update`

> Atualiza a lista de pacotes

- `sudo apt upgrade`

> Atualiza os pacotes (utilizado após *apt update*)

- `sudo apt remove [pacote]`

> Desinstala o pacote