

Terminal de Linux

Tópicos de Informática para Automação

Mário Antunes

September 22, 2025

Universidade de Aveiro



Mais do que uma Caixa Preta

O **Terminal** é a sua ligação direta, baseada em texto, ao sistema operativo.

- **Porquê usá-lo?**

- **Poder e Velocidade:** Execute tarefas complexas instantaneamente.
- **Automação:** Crie *scripts* para tarefas repetitivas.
- **Eficiência:** Utiliza o mínimo de recursos do sistema.
- **Padrão da Indústria:** Essencial para programadores e administradores de sistemas.

Analogia: Uma GUI é o menu de um restaurante. A CLI é falar diretamente com o *chef*.

A Shell & o Bash

A ***shell*** é o programa que interpreta os seus comandos. O terminal é a janela; a *shell* é o cérebro lá dentro.

- Existem muitas *shells*, cada uma com características diferentes:
 - sh (Bourne Shell): A *shell* original, clássica.
 - zsh (Z Shell): Uma *shell* moderna e popular com vasta personalização.
 - fish (Friendly Interactive Shell): Focada em ser fácil de usar.
 - bash (**Bourne Again SHell**): A *shell* mais comum em Linux. É o padrão *de facto* que vamos aprender hoje.

O Sistema de Ficheiros do Linux (Parte 1: Diretórios Principais)

O sistema de ficheiros é uma árvore que começa na **raiz** (/).

- /: O **diretório raiz**. Tudo começa aqui.
- /home: Os seus ficheiros pessoais estão aqui (ex: /home/student).
- /bin: **Binários** essenciais do utilizador (programas como ls).
- /etc: Ficheiros de **configuração** de todo o sistema.
- /var: Dados **variáveis**, como logs do sistema (/var/log).
- /tmp: Para ficheiros **temporários**.

O Sistema de Ficheiros do Linux (Parte 2: Software & Administração)

Mais locais importantes que irá encontrar.

- `/opt`: Software **opcional**. Usado por programas de terceiros que instala manualmente (ex: Google Chrome).
- `/usr/local`: Um local para *software* que compila ou instala para todos os utilizadores e que não faz parte da distribuição padrão do SO. Frequentemente, encontrará `/usr/local/bin`.
- `/root`: O diretório pessoal do **superutilizador** (utilizador *root*). Não confunda com o diretório raiz `!/`

Ficheiros & Diretórios Ocultos

No seu diretório pessoal (~), muitos ficheiros de configuração estão “ocultos”, começando com um ponto (.). Eles controlam como os seus programas e a *shell* se comportam.

- **Exemplos:**

- `~/.bashrc`: *Script* de configuração da *shell* Bash. Este é um ficheiro crucial.
- `~/.config`: Um diretório comum para configurações de aplicações.
- `~/.themes` ou `~/.local/share/themes`: Para temas do *desktop*.
- `~/.gitconfig`: A sua configuração do Git.

Navegação Básica: pwd e cd

Dois comandos fundamentais para se mover no sistema.

- **pwd:** Print Working Directory. Mostra a sua localização atual.

```
$ pwd  
/home/student
```

- **cd:** Change Directory. Move-o para um caminho absoluto ou relativo.

```
$ cd /var/log      # Mover para um caminho absoluto  
$ cd Documents    # Mover para um subdiretório
```

Atalhos Especiais de Navegação com cd

O cd tem vários atalhos úteis para uma navegação mais rápida.

- Subir um nível:

```
$ cd ..
```

- Ir diretamente para o seu diretório pessoal a partir de qualquer lugar:

```
$ cd ~
```

(Ou apenas cd sem argumentos)

- Voltar ao último diretório onde esteve:

```
$ cd -
```

Listar Conteúdo de Diretórios: ls

O comando `ls` **lista** o conteúdo de um diretório. São os seus olhos no terminal.

- Use **flags** para alterar o seu comportamento. A mais comum é `-l` para um formato de lista longa.

```
$ ls -l
-rw-r--r-- 1 student student 4096 Sep 19 2025 o_meu_doc.txt
drwxr-xr-x 2 student student 4096 Sep 17 2025 Scripts
```

Isto mostra permissões, proprietário, tamanho e data de modificação.

Ver Tudo com ls -a

Como podemos ver aqueles ficheiros de configuração ocultos?

- A *flag* `-a` diz ao `ls` para mostrar **todos** (*all*) os ficheiros.

```
$ ls -a
. .. .bashrc .profile Documents Downloads
```

- Pode combinar *flags*. `ls -la` é um comando muito comum para obter uma lista longa de **todos** os ficheiros.

Criar Diretórios: mkdir

Use o comando `mkdir` para **make a directory** (criar um diretório).

- **Criar um único diretório:**

```
$ mkdir o_meu_projeto
```

- **Criar uma estrutura aninhada:** A *flag* `-p` (**parents**) cria todo o caminho de diretórios, mesmo que os diretórios pais ainda não existam.

```
$ mkdir -p Documentos/Trabalho/2025/Relatorios
```

Criar & Editar Ficheiros: touch & nano

Depois de ter os diretórios, precisa de ficheiros para colocar neles.

- **touch:** A forma mais rápida de criar um ficheiro novo e vazio.

```
$ touch as_minhas_notas.txt
```

- **nano:** Um editor de texto simples e amigável para o terminal.

```
$ nano as_minhas_notas.txt
```

- Escreva o seu texto diretamente na janela.
- Pressione **Ctrl+X** para sair.
- Pressione **S** (Sim) para confirmar que deseja guardar e, de seguida, **Enter**.

Obter Informação do Sistema

O terminal é excelente para verificar rapidamente o estado do sistema.

- `whoami`: Mostra o seu nome de utilizador atual.
- `date`: Mostra a data e hora atuais.
- `uname -a`: Mostra informação do *kernel* e do sistema.
- `top`: Mostra os processos em execução em tempo real (como o Gestor de Tarefas). Pressione `q` para sair.

Utilizadores: Padrão vs. Superutilizador

O Linux é um sistema multiutilizador.

- **Utilizador Padrão (student)**: A sua conta do dia a dia com privilégios limitados.
- **Superutilizador (root)**: O administrador. Tem poder completo sobre o sistema.

Para executar um único comando com privilégios de *root*, use **sudo (Superuser do)**.

```
# Isto precisa de direitos de administrador, por isso usamos sudo  
$ sudo apt update
```

Gerir Utilizadores

Como administrador, pode gerir contas de utilizador a partir da linha de comandos.

- `sudo useradd novo_utilizador`: Cria um novo utilizador.
- `sudo passwd novo_utilizador`: Define a *password* para o novo utilizador.
- `sudo userdel novo_utilizador`: Apaga um utilizador.

Compreender as Permissões de Ficheiros

O comando `ls -l` mostra as permissões como uma cadeia de 10 caracteres, como `-rwxr-xr--`.

- **Lê-se em grupos:** Tipo | Proprietário | Grupo | Outros
- r: Permissão para **ler** (*read*) o ficheiro.
- w: Permissão para **escrever** (*write*) (modificar) o ficheiro.
- x: Permissão para **executar** (*execute*) o ficheiro (correr como um programa).

Gerir Permissões com chmod

Use o comando chmod (**change mode**) para alterar permissões.

- Pode adicionar (+) ou remover (-) permissões para o **utilizador (user)**, **grupo** ou **outros (others)**.

Exemplo: Tornar um *script* executável para si mesmo.

```
# Dar ao utilizador (u) a permissão de execução (x)
$ chmod u+x o_meu_script.sh
```

O que é um Gestor de Pacotes?



Um gestor de pacotes (*package manager*) é uma ferramenta que automatiza o processo de instalar, atualizar e remover *software*.

- Gere **dependências** automaticamente, para que não tenha de instalar as bibliotecas necessárias manualmente.
- Mantém uma base de dados do *software* instalado, facilitando a gestão.
- Para sistemas baseados em Debian e Ubuntu, o principal gestor de pacotes é o **APT** (Advanced Package Tool).

Analogia: Pense no apt como uma App Store para o seu terminal.

Atualizar Listas de Pacotes (apt update)

Antes de instalar ou procurar o que quer que seja, deve sincronizar a sua lista de pacotes local com os repositórios de *software* centrais.

- Este comando **não** atualiza o seu *software*. Apenas descarrega a lista mais recente do que está disponível.
- Esta é uma operação privilegiada, por isso requer sudo.

```
# Descarrega a informação mais recente dos pacotes  
$ sudo apt update
```

Procurar Pacotes (apt search)

Se não tiver a certeza do nome exato de um programa, pode procurá-lo.

- Este comando pesquisa nos nomes e descrições de todos os pacotes disponíveis.
- Não precisa de sudo para procurar.

Exemplo: Procurar um programa que mostre processos do sistema, como o htop.

```
$ apt search htop
```

Instalar Pacotes (apt install)

Assim que souber o nome do pacote, pode instalá-lo.

- O apt irá descarregar e instalar automaticamente o programa e quaisquer dependências de que ele precise para funcionar.
- Isto requer sudo.

Exemplo: Instalar o utilitário htop, um visualizador de processos interativo.

```
$ sudo apt install htop
```

Após a instalação, pode executar o programa simplesmente escrevendo htop.

Remover Pacotes (apt remove / apt purge)

Remover *software* é tão fácil como instalá-lo. Tem duas opções principais:

1. `apt remove`: Desinstala o programa, mas deixa os seus ficheiros de configuração (útil se planejar reinstalá-lo mais tarde).
2. `apt purge`: Desinstala o programa **e** apaga todos os seus ficheiros de configuração.

Exemplos:

```
# Remover o htop, mas manter os seus ficheiros de configuração
$ sudo apt remove htop
```

```
# Remover o htop e todos os seus ficheiros de configuração
$ sudo apt purge htop
```



O **cron** é um *daemon* do sistema (um processo em *background*) que executa tarefas agendadas. Estas tarefas agendadas são conhecidas como ***cron jobs***.

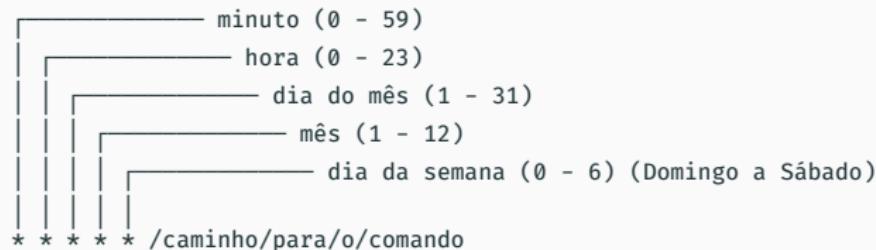
- É a ferramenta padrão para automatizar tarefas repetitivas num horário.
- Gere a sua lista pessoal de *cron jobs* usando o comando crontab.

Utilizações Comuns:

- Executar um *script* de *backup* todas as noites.
- Realizar manutenção do sistema, como um **ZFS scrub** semanal ou um **SSD trim** diário.
- Limpar ficheiros temporários.

Compreender a Sintaxe do crontab

Um *cron job* consiste em duas partes: o **horário** e o **comando**. O horário é definido por cinco campos, muitas vezes representados por asteriscos (*).



Um asterisco * significa “todos”. Por exemplo, um asterisco no campo “hora” significa “a todas as horas”.

Para uma forma fácil de gerar a cadeia de tempo correta, consulte: crontab.guru

Gerir o seu crontab

Pode editar, ver e remover os seus *cron jobs* com o comando `crontab` e uma *flag*.

- `crontab -e`: **Editar** o seu ficheiro `crontab`. Da primeira vez que executar isto, ser-lhe-á pedido para escolher um editor de texto (como o `nano`).
- `crontab -l`: **Listar** os seus *cron jobs* atualmente agendados.
- `crontab -r`: **Remover** todo o seu ficheiro `crontab` (use com cuidado!).

Exemplos de crontab

Aqui estão alguns exemplos práticos que pode adicionar usando crontab -e.

Exemplo 1: Executar um *script de backup* todos os dias às 3:30 da manhã.

```
# Minuto Hora Dia(M) Mês Dia(S) Comando  
30      3      * * * /home/student/scripts/backup.sh
```

Exemplo 2: Executar um comando de manutenção do sistema todos os Domingos às 4:00 da manhã. Este exemplo é para um comando de sistema como um *scrub* a uma *pool* de armazenamento ZFS.

```
# Minuto Hora Dia(M) Mês Dia(S) Comando  
0       4      * * 0      /usr/sbin/zpool scrub my-storage-pool
```

Exemplo 3: Verificar o espaço em disco a cada 15 minutos e registrar o *output*. O >> anexa o *output* a um ficheiro de log, e 2>&1 garante que os erros também são registados.

Redirecionamento: Guardar *Output* com >

Não quer ver o *output* no ecrã? Guarde-o num ficheiro com >.

Aviso: Isto **sobrescreve** o ficheiro se ele já existir.

Exemplo: Guardar uma lista do conteúdo do seu diretório pessoal num ficheiro.

```
$ ls -l ~ > os_meus_ficheiros.txt
```

Redirecionamento: Anexar *Output* com >>

Para **adicionar** *output* ao final de um ficheiro sem apagar o seu conteúdo, use **>>**.

- Isto é ótimo para criar ficheiros de *log*.

Exemplo: Adicionar uma entrada com data e hora a um ficheiro de *log*.

```
$ echo "Sistema reiniciado às $(date)" >> system.log
```

O Poder do Pipe |

O **pipe** é um dos conceitos mais poderosos da *shell*. Ele envia o *output* de um comando para ser o *input* do seguinte.

Pense nisto como canalização: Comando A -> | ->
Comando B

Exemplo: Encontrar todos os ficheiros .log num diretório.

```
# O output de 'ls' é "canalizado" para o 'grep' para ser filtrado.  
$ ls /var/log | grep .log
```

O Seu Ambiente: Variáveis

A *shell* usa variáveis para armazenar informação. Por convenção, estão em MAIÚSCULAS.

- \$HOME: O seu diretório pessoal.
- \$USER: O seu nome de utilizador.
- \$PATH: Uma lista de diretórios onde a *shell* procura programas.

Exemplo: Ver o conteúdo da variável \$PATH.

```
$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Personalizar a Sua Shell: `.bashrc`

O ficheiro `~/.bashrc` é um *script* que é executado sempre que abre um novo terminal. Este é o lugar para personalizar a sua *shell*.

Pode editá-lo com um editor de texto:

```
$ nano ~/.bashrc
```

Lembre-se: As alterações não serão aplicadas até que abra um novo terminal ou execute `source ~/.bashrc`.

Exemplo de Personalização: Aliases

Um **alias** é um atalho ou uma alcunha para um comando mais longo. Poupa-lhe muito tempo a escrever!

- Adicione esta linha ao seu ficheiro `~/.bashrc`:

```
alias ll='ls -alF'
```

- Agora, quando escrever `ll` num novo terminal, o *bash* irá executar `ls -alF` por si.

Introdução ao Bash Scripting

Um **script** é simplesmente um ficheiro de texto que contém uma sequência de comandos.

1. A primeira linha **deve** ser `#!/bin/bash`. Isto é chamado de “*shebang*”.
2. Adicione os seus comandos.
3. Use `#` para comentários para explicar o seu código.
4. Torne o ficheiro executável com `chmod +x`.

Exemplo de Script 1: Olá Mundo

Este *script* usa uma variável e o comando echo. É o “Olá, Mundo!” do *scripting*.

Ficheiro: ola.sh

```
#!/bin/bash
# Um script simples de olá mundo

NOME="Estudante"
echo "Olá, $NOME!"
```

Para executá-lo:

```
$ chmod +x ola.sh
$ ./ola.sh
```

Exemplo de Script 2: Usar if

Este *script* usa uma instrução **if** para verificar se um ficheiro existe antes de tentar usá-lo.

Ficheiro: verifica_ficheiro.sh

```
#!/bin/bash
# Verifica a existência do ficheiro de log do sistema.

FICHEIRO="/var/log/syslog"

if [ -f "$FICHEIRO" ]; then
    echo "$FICHEIRO existe."
    # Agora poderíamos fazer algo com o ficheiro, ex:
    # tail -n 5 "$FICHEIRO"
else
    echo "Aviso: $FICHEIRO não encontrado."
fi
```

Exemplo de Script 3: Ciclo Sobre Ficheiros

Um ciclo `for` permite-lhe realizar uma ação numa lista de itens, como ficheiros.

Ficheiro: adiciona_prefixo.sh

```
#!/bin/bash
# Adiciona o prefixo "backup_" a todos os ficheiros .txt.

for ficheiro in *.txt
do
    # Verificar se é um ficheiro antes de o mover
    if [ -f "$ficheiro" ]; then
        mv -- "$ficheiro" "backup_${ficheiro}"
        echo "→ backup_${ficheiro}"
    fi
done

echo "Renomeação em lote concluída."
```

Exemplo de Script 4: Script Complexo

Este *script* combina argumentos, *if*, variáveis e um programa (*tar*) para criar uma ferramenta útil.

Ficheiro: backup.sh

```
#!/bin/bash
# Faz o backup dos itens especificados para um arquivo .tar.gz.

# Sair se não forem fornecidos argumentos.
if [ "$#" -eq 0 ]; then
    echo "Utilização: $0 <ficheiro1> <dir1> ... "
    exit 1
fi

DEST="$HOME/backups"
TIME=$(date +%Y-%m-%d_%H%M%S)
ARQUIVO="$DEST/$TIME-backup.tar.gz"

mkdir -p "$DEST" # Criar dir de backup se necessário
echo "A criar arquivo ..."

# "$@" contém todos os argumentos da linha de comandos.
tar -czf "$ARQUIVO" "$@"

echo "Backup concluído: $ARQUIVO"
```

Agora já viu os conceitos centrais da linha de comandos do Linux:

- **Navegar** no sistema de ficheiros.
- **Gerir** ficheiros, permissões e utilizadores.
- **Combinar** comandos com *pipes* e redirecionamento.
- **Automatizar** tarefas com *shell scripts*.

Agora, vamos aplicar este conhecimento na parte prática da aula.



Guarde estas páginas nos seus favoritos. São referências incrivelmente úteis.

- **Linux Terminal Cheat Sheet:**

- <https://www.geeksforgeeks.org/linux-unix/linux-commands-cheat-sheet/>

- **Bash Cheat Sheet:**

- <https://github.com/RehanSaeed/Bash-Cheat-Sheet>

- **Bash Scripting Cheat Sheet:**

- <https://developers.redhat.com/cheat-sheets/bash-shell-cheat-sheet>