

O que é Linux?

O termo "Linux" é frequentemente usado de forma ambígua, o que pode gerar confusão. Para usuários casuais, Linux é sinônimo de um **sistema operacional completo**, como o Windows ou o MacOS, mas para especialistas, ele se refere especificamente ao **kernel**.

Quando falamos de Linux no sentido amplo, estamos nos referindo a distribuições como **Ubuntu, Fedora, Debian ou Kali Linux**. Essas distribuições são compostas pelo kernel Linux, além de uma série de softwares adicionais, como bibliotecas (**libc**), ambientes gráficos (**GNOME, KDE**), editores de texto (**nano, vim**), compiladores e utilitários. Essas distribuições são projetadas para serem usadas diretamente por usuários finais. Por exemplo, ao instalar o Ubuntu, você obtém um sistema operacional funcional que inclui o kernel Linux como base, mas também ferramentas como o gerenciador de pacotes **apt** e interfaces gráficas.

O **kernel Linux**, por outro lado, é o componente central do sistema operacional, responsável por gerenciar os recursos de hardware e software. Ele atua como uma camada intermediária entre o hardware (como CPU, memória RAM, discos rígidos e dispositivos periféricos) e as aplicações que rodam no sistema. Suas funções principais incluem:

- **Gerenciamento de processos:** Permite a execução de múltiplos programas simultaneamente (multitarefa) e gerencia o agendamento de tarefas.
- **Gerenciamento de memória:** Aloca e libera memória para os processos, garantindo eficiência e segurança.
- **Drivers de dispositivos:** Facilita a comunicação com hardware, como teclados, impressoras e placas de rede.
- **Chamadas de sistema (*System calls*):** Fornece uma interface para que programas interajam com o hardware de forma segura, sem acesso direto.

O kernel Linux foi inicialmente desenvolvido por **Linus Torvalds** em 1991, como um projeto de código aberto, e é distribuído sob a licença **GNU General Public License versão 2 (GPLv2)**. Ele é altamente personalizável e está presente em uma vasta gama de dispositivos, desde servidores e desktops até smartphones (como no Android) e sistemas embarcados.

<https://github.com/torvalds/linux>

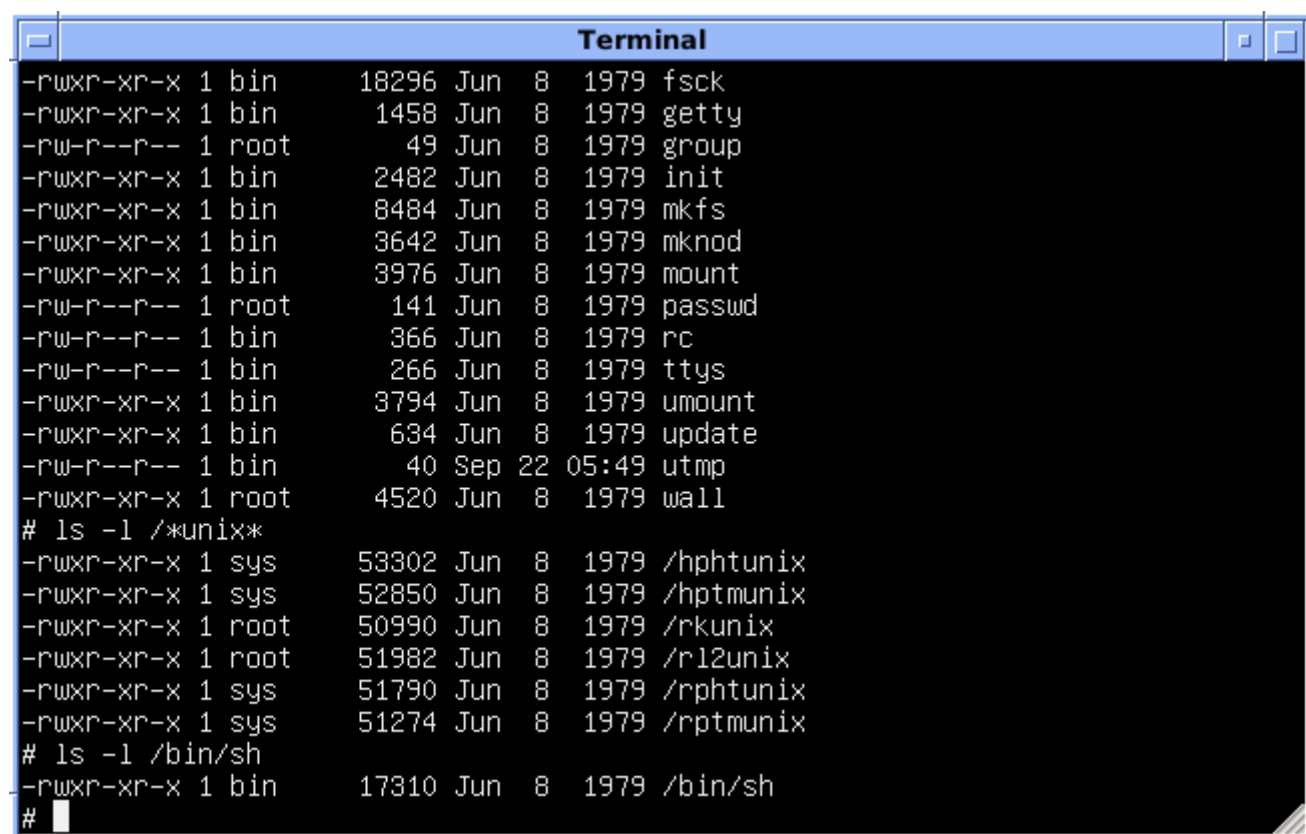


Shell

Um shell é um programa que atua como uma interface entre o usuário e o kernel do sistema operacional Linux. Ele permite que os usuários interajam com o sistema por meio de comandos digitados em um terminal, interpretando esses comandos e enviando instruções ao kernel para execução. O shell é essencial para tarefas como navegação no sistema de arquivos, gerenciamento de processos, execução de programas e automação de tarefas por meio de scripts.

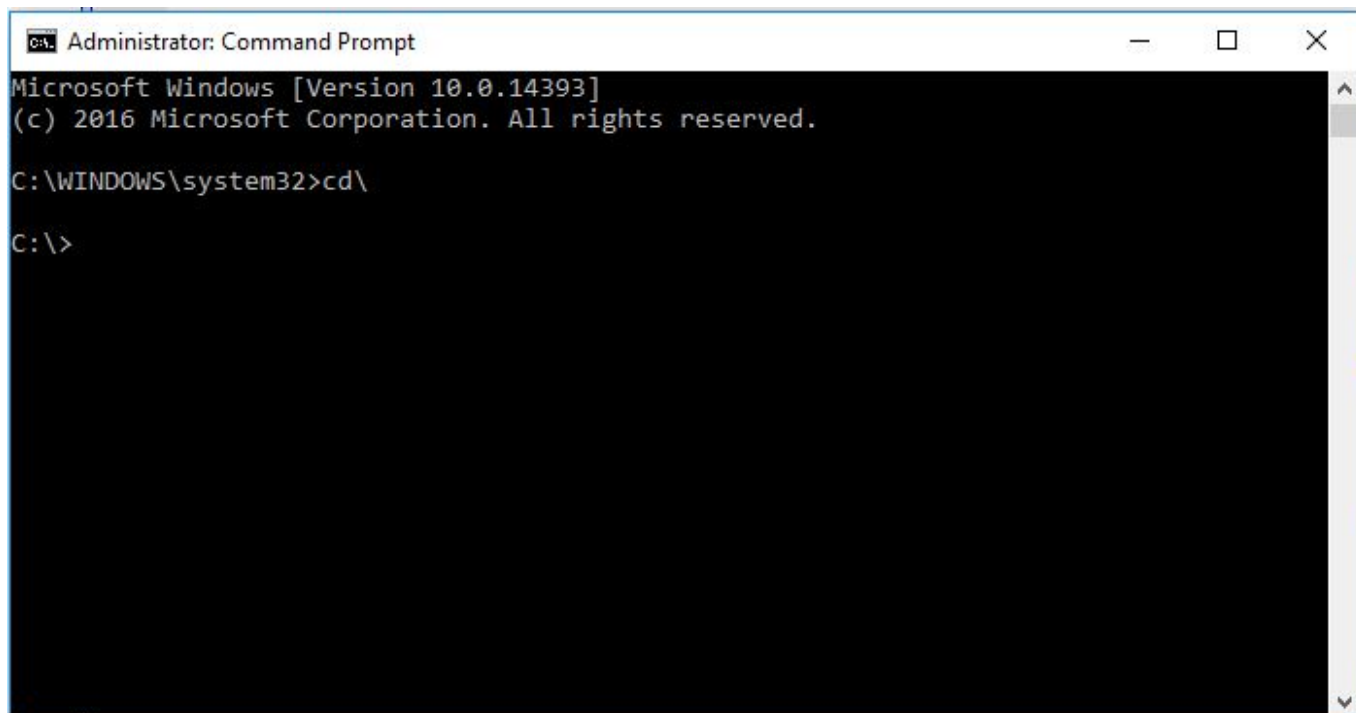
Existem vários tipos de shells disponíveis no Linux, cada um com características, sintaxes e propósitos específicos:

- **sh (Bourne Shell):** Desenvolvido na década de 1970 por Stephen Bourne na Bell Labs, é o shell original do Unix. É minimalista, focado em compatibilidade e frequentemente usado em scripts que precisam rodar em diferentes sistemas Unix-like. Sua simplicidade o torna ideal para ambientes onde recursos são limitados.
- **bash (Bourne Again Shell):** Criado no final dos anos 1980 por Brian Fox para o projeto GNU, é uma evolução do **sh**. Tornou-se o shell padrão em muitas distribuições Linux, como Ubuntu e Fedora, oferecendo recursos adicionais, como histórico de comandos, edição de linha de comando e suporte robusto a scripting. É amplamente usado tanto para interação quanto para automação.



```
Terminal
-rwxr-xr-x 1 bin      18296 Jun  8  1979 fsck
-rwxr-xr-x 1 bin      1458 Jun  8  1979 getty
-rw-r--r-- 1 root        49 Jun  8  1979 group
-rwxr-xr-x 1 bin     2482 Jun  8  1979 init
-rwxr-xr-x 1 bin     8484 Jun  8  1979 mkfs
-rwxr-xr-x 1 bin     3642 Jun  8  1979 mknod
-rwxr-xr-x 1 bin     3976 Jun  8  1979 mount
-rw-r--r-- 1 root     141 Jun  8  1979 passwd
-rw-r--r-- 1 bin      366 Jun  8  1979 rc
-rw-r--r-- 1 bin      266 Jun  8  1979 ttys
-rwxr-xr-x 1 bin     3794 Jun  8  1979 umount
-rwxr-xr-x 1 bin      634 Jun  8  1979 update
-rw-r--r-- 1 bin       40 Sep 22 05:49 utmp
-rwxr-xr-x 1 root    4520 Jun  8  1979 wall
# ls -l /*unix*
-rwxr-xr-x 1 sys     53302 Jun  8  1979 /hphtunix
-rwxr-xr-x 1 sys     52850 Jun  8  1979 /hptmunix
-rwxr-xr-x 1 root    50990 Jun  8  1979 /rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 /rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 /rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 /rptmunix
# ls -l /bin/sh
-rwxr-xr-x 1 bin     17310 Jun  8  1979 /bin/sh
#
```

- **cmd (Command Prompt):** Evoluiu do **command.com** do MS-DOS e é o shell padrão do Windows desde o NT. Lançado inicialmente em 1987, é baseado em texto, com limitações em scripting e automação, sendo mais adequado para tarefas básicas, como navegação de diretórios e execução de comandos simples.

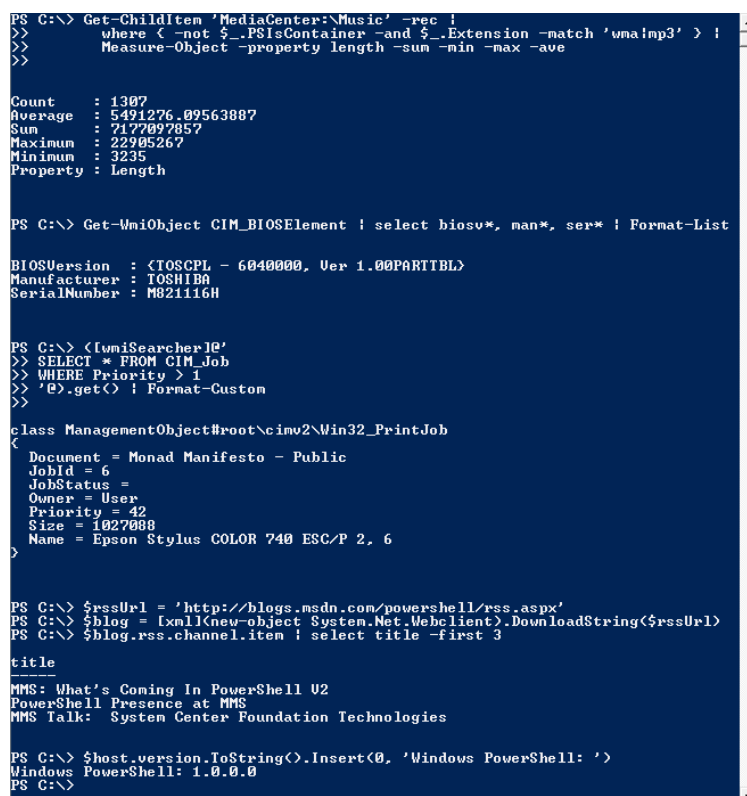


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd\

C:\>
```

- **PowerShell:** Introduzido pela Microsoft em 2006, é um shell avançado projetado para superar as limitações do `cmd`. É orientado a objetos, integrado ao .NET Framework e oferece funcionalidades como cmdlets (comandos específicos) e suporte a scripting sofisticado, sendo ideal para administração de sistemas Windows. Desde 2016, com o PowerShell Core, também está disponível para Linux e macOS.



```
PS C:\> Get-Childitem 'MediaCenter:\Music' -rec |
>> where < -not $_.PSIsContainer -and $_.Extension -match '.mp3' > |
>> Measure-Object -property length -sum -min -max -ave

Count       : 1307
Average     : 5491276.09563887
Sum         : 7177097857
Maximum     : 22905267
Minimum     : 3235
Property    : Length

PS C:\> Get-WmiObject CIM_BIOSElement | select biosv*, man*, ser* | Format-List

BIOSVersion : <TOSCP1 - 6040000, Ver 1.00PARTIBL>
Manufacturer : TOSHIBA
SerialNumber : M821116H

PS C:\> <[wmiSearcher]@'
>> SELECT * FROM CIM_Job
>> WHERE Priority > 1
>> '().get()' | Format-Custom
>>

class ManagementObject#root\cimv2\Win32_PrintJob
{
    Document = Monad Manifesto - Public
    JobId = 6
    JobStatus =
    Owner = User
    Priority = 42
    Size = 1027008
    Name = Epson Stylus COLOR 740 ESC/P 2. 6
}

PS C:\> $rssUrl = 'http://blogs.msdn.com/powershell/rss.aspx'
PS C:\> $blog = [xml](new-object System.Net.WebClient).DownloadString($rssUrl)
PS C:\> $blog.rss.channel.item | select title -first 3

title
----
MMS: What's Coming In PowerShell V2
PowerShell Presence at MMS
MMS Talk: System Center Foundation Technologies

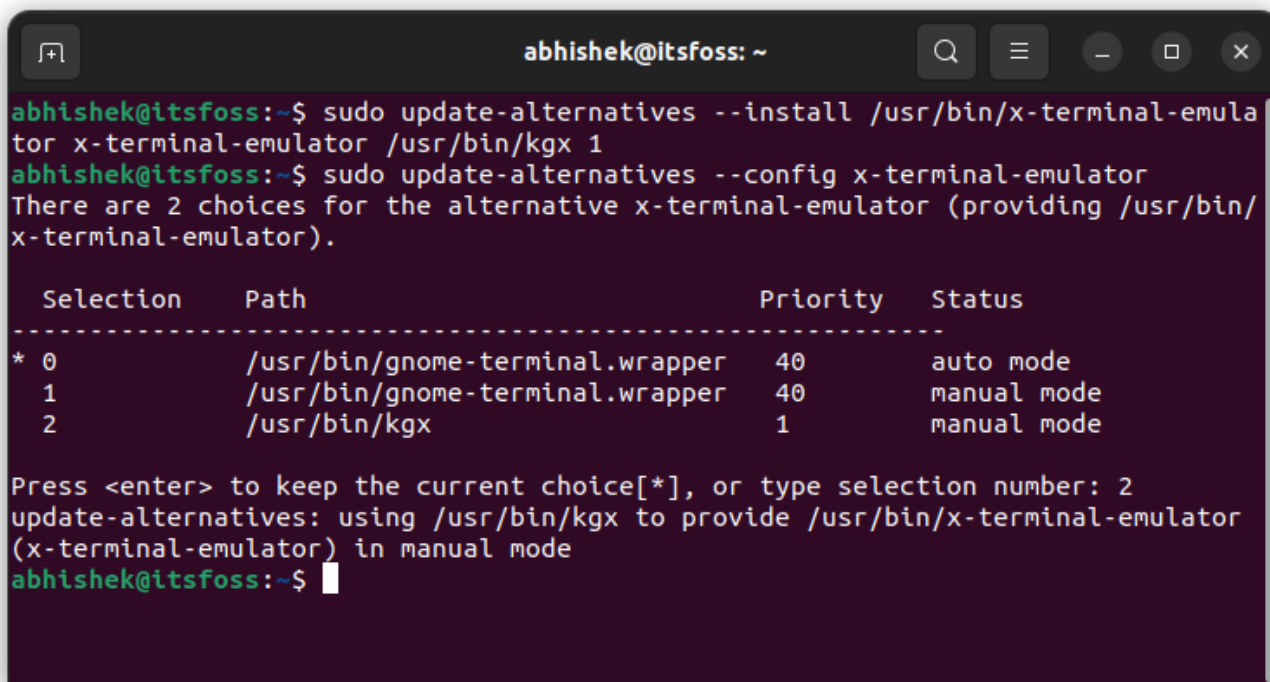
PS C:\> $host.version.ToString().Insert(0, 'Windows PowerShell: ')
Windows PowerShell: 1.0.0.0
PS C:\>
```

Hoje existem muitos outros shells, como o `zsh`, `fish` e `nushell`, cada um com suas próprias características e funcionalidades. A escolha do shell depende das preferências do usuário e das necessidades específicas de cada tarefa.

Terminais

Um terminal é um programa que fornece uma interface de linha de comando (**CLI**) para o usuário interagir com o sistema operacional. Ele permite que os usuários executem comandos, visualizem saídas e interajam com o shell. O terminal pode ser acessado diretamente em um ambiente gráfico ou por meio de uma conexão remota, como SSH.

Tecnicamente, o terminal que a maioria dos usuários vê em um sistema gráfico é, na verdade, um **emulador de terminal**, um programa que simula o ambiente de linha de comando dentro de uma janela gráfica. Exemplos comuns incluem **GNOME Terminal** (usado no Ubuntu), **Konsole** (no KDE), ou **xterm**. Esses emuladores fornecem uma interface amigável para acessar o shell, que é o programa que interpreta os comandos.



```
abhishek@itsfoss: ~  
abhishek@itsfoss:~$ sudo update-alternatives --install /usr/bin/x-terminal-emulator x-terminal-emulator /usr/bin/kgx 1  
abhishek@itsfoss:~$ sudo update-alternatives --config x-terminal-emulator  
There are 2 choices for the alternative x-terminal-emulator (providing /usr/bin/x-terminal-emulator).  
  
  Selection    Path                                          Priority  Status  
-----  
* 0            /usr/bin/gnome-terminal.wrapper            40      auto mode  
  1            /usr/bin/gnome-terminal.wrapper            40      manual mode  
  2            /usr/bin/kgx                               1       manual mode  
  
Press <enter> to keep the current choice[*], or type selection number: 2  
update-alternatives: using /usr/bin/kgx to provide /usr/bin/x-terminal-emulator (x-terminal-emulator) in manual mode  
abhishek@itsfoss:~$
```

Historicamente, o termo "terminal" referia-se a dispositivos físicos conectados a computadores mainframe, como teclados e monitores usados para entrada e saída de dados.



Principais comandos

Os comandos são a forma principal de interação com o shell. Eles permitem que os usuários executem tarefas, manipulem arquivos e gerenciem o sistema. Os comandos listados abaixo são alguns dos mais utilizados:

ls (list)

Lista os arquivos e diretórios no diretório atual ou especificado, essencial para visualizar o conteúdo de pastas. `ls` lista o conteúdo do diretório atual; `ls /home/user` lista o conteúdo de `/home/user`.

Flags importantes:

- `-l` (list): Exibe informações detalhadas, como permissões, dono, tamanho e data de modificação.
- `-a` (all): Mostra todos os arquivos, incluindo ocultos (começando com `.`). Exemplo: `ls -a` revela arquivos como `.bashrc`.
- `-R` (recursive): Lista recursivamente, incluindo subdiretórios. Exemplo: `ls -R /var/log` lista logs e subpastas.
- `-t` (time): Ordena por data de modificação, útil para ver os arquivos mais recentes. Exemplo: `ls -lt`.

cd (change directory)

Muda o diretório atual, fundamental para navegação no sistema de arquivos. `cd /home/user/documents` muda para o diretório especificado. Utilize `cd ..` para voltar ao diretório pai e `cd ~` para ir ao diretório home do usuário atual.

pwd (print working directory)

Exibe o caminho completo do diretório atual, útil para confirmar a localização no sistema de arquivos. `pwd` retorna algo como `/home/user/documents`.

mkdir (make directory)

Cria um novo diretório. `mkdir new_folder` cria um diretório chamado `new_folder` no diretório atual.

Flags importantes:

- `-p` (parent): Cria diretórios pai se necessário, sem erro se já existirem. Exemplo: `mkdir -p /caminho/novo/diretorio`.
- `-m` (mode): Define permissões iniciais, como `mkdir -m 755 nova-pasta` para permissões de leitura e execução para outros.

rm (remove)

Remove arquivos ou diretórios. `rm file.txt` remove o arquivo `file.txt`.

Flags importantes:

- `-r` (recursive): Remove recursivamente, necessário para diretórios. Exemplo: `rm -r pasta-com-subpastas`.
- `-f` (force): Força a remoção sem perguntar, útil para scripts. Exemplo: `rm -f arquivo.txt`.

- `-i` (interactive): Modo interativo, pergunta antes de cada remoção. Exemplo: `rm -i arquivo.txt` para confirmação.

cp (copy)

Copia arquivos ou diretórios. `cp source.txt destination.txt` copia `source.txt` para `destination.txt`.

Flags importantes:

- `-r` (recursive): Copia diretórios e conteúdo recursivamente. Exemplo: `cp -r pasta /backup`.
- `-i` (interactive): Pergunta antes de sobrescrever, evitando erros. Exemplo: `cp -i arquivo.txt /backup`.
- `-p` (preserve): Preserva atributos como data de modificação e permissões. Exemplo: `cp -p arquivo.txt /backup`.

mv (move)

Move ou renomeia arquivos e diretórios. `mv oldname.txt newname.txt` renomeia `oldname.txt` para `newname.txt`.

Flags importantes:

- `-i` (interactive): Pergunta antes de sobrescrever. Exemplo: `mv -i arquivo.txt /novo/caminho`.
- `-f` (force): Força a movimentação sem perguntar. Exemplo: `mv -f arquivo.txt /novo/caminho`.

touch

Cria um novo arquivo vazio ou atualiza a data de modificação de um arquivo existente. `touch newfile.txt` cria `newfile.txt` se não existir ou atualiza a data se já existir.

cat (concatenate)

Exibe o conteúdo de arquivos ou concatena múltiplos arquivos, útil para visualizar logs ou textos. `cat file.txt` exibe o conteúdo de `file.txt`. Para concatenar, use `cat file1.txt file2.txt > combined.txt`, criando `combined.txt`.

Flags importantes:

- `-n` (number): Numera as linhas do arquivo exibido. Exemplo: `cat -n file.txt`.
- `-b` (number non-blank): Numera apenas linhas não vazias. Exemplo: `cat -b file.txt`.

grep (global regular expression print)

Busca por padrões em arquivos, útil para filtrar informações. `grep "pattern" file.txt` procura por "pattern" em `file.txt`.

Flags importantes:

- `-i` (ignore case): Ignora diferenças entre maiúsculas e minúsculas. Exemplo: `grep -i "pattern" file.txt`.
- `-r` (recursive): Busca recursivamente em diretórios. Exemplo: `grep -r "pattern" /path/to/dir`.

- **-v** (invert match): Exibe linhas que não correspondem ao padrão. Exemplo: `grep -v "pattern" file.txt`.
- **-l** (files with matches): Lista arquivos que contêm o padrão. Exemplo: `grep -l "pattern" *.txt`.
- **-c** (count): Conta o número de ocorrências do padrão. Exemplo: `grep -c "pattern" file.txt`.

find

Busca arquivos e diretórios com base em critérios específicos, como nome, tipo ou data de modificação. `find /path/to/search -name "filename"` procura por `filename` no diretório especificado.

Flags importantes:

- **-type**: Especifica o tipo de arquivo a ser buscado. Exemplo: `find /path -type f` busca apenas arquivos. Pode ser `f` (arquivo), `d` (diretório), `l` (link simbólico), etc.
- **-name**: Busca por nome exato. Exemplo: `find /path -name "*.txt"` busca arquivos `.txt`.
- **-iname**: Busca por nome, ignorando maiúsculas/minúsculas. Exemplo: `find /path -iname "*.txt"`.

O comando `find` é muito poderoso e contém muitas outras opções. Para mais detalhes, consulte a documentação do comando.

chmod (change mode)

Altera as permissões de arquivos e diretórios. `chmod 755 file.txt` define permissões específicas para `file.txt`. Mais no capítulo de [Usuários, grupos e permissões](#)

Flags importantes:

- **-R** (recursive): Aplica mudanças recursivamente a diretórios e subdiretórios. Exemplo: `chmod -R 755 /path/to/dir`.

chown (change owner)

Altera o proprietário e/ou grupo de arquivos e diretórios. `chown user:group file.txt` muda o proprietário para `user` e o grupo para `group`. Mais no capítulo de [Usuários, grupos e permissões](#)

Flags importantes:

- **-R** (recursive): Aplica mudanças recursivamente a diretórios e subdiretórios. Exemplo: `chown -R user:group /path/to/dir`.
- **--reference**: Altera o proprietário e grupo de um arquivo para corresponder a outro arquivo. Exemplo: `chown --reference=ref_file.txt target_file.txt`.
- **--from**: Altera o proprietário e grupo de um arquivo apenas se corresponder a um proprietário ou grupo específico. Exemplo: `chown --from=olduser:oldgroup newuser:newgroup file.txt`.

Usuários, grupos e permissões

No Linux, a gestão de usuários, grupos e permissões é a base para garantir a segurança e organização do sistema. Esses conceitos permitem controlar quem pode acessar ou modificar arquivos e recursos, protegendo dados sensíveis e evitando acessos não autorizados.

Usuários

```
marko@test-main:~$ getent passwd {0..100}
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailng List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
marko@test-main:~$
```

Um usuário no Linux é uma entidade que pode fazer login e interagir com o sistema. Cada usuário tem um nome único e um ID de usuário (UID), armazenado no arquivo `/etc/passwd`. Existem três tipos principais de usuários:

- **Usuários regulares:** Contas criadas para pessoas, geralmente com um diretório home (`/home/kali`) para armazenar arquivos pessoais.
- **Usuários do sistema:** Criados para processos ou serviços, como `www-data` para servidores web, com UIDs baixos (geralmente abaixo de 1000).
- **Root (superusuário):** O administrador com acesso total ao sistema, identificado pelo UID 0. Deve ser usado com cuidado, geralmente via `sudo`.

Gerenciamento de Usuários:

- **Criar usuário:** `sudo useradd -m joao` cria o usuário "joao" com um diretório home (`-m`).
- **Excluir usuário:** `sudo userdel -r joao` remove o usuário e seu diretório home (`-r`).
- **Modificar usuário:** `sudo usermod -aG grupo joao` adiciona "joao" ao grupo especificado (`-aG` (add group) para não remover de outros grupos).
- **Verificar usuários:** `cat /etc/passwd` lista todos os usuários e suas propriedades.

Grupos


```
dnyce@LinuxShellTips:~$ getent group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,dnyce
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
```

Um grupo é uma coleção de usuários que compartilham permissões de acesso a arquivos ou recursos. Cada grupo tem um nome e um ID de grupo (GID), armazenado em `/etc/group`. Todo usuário pertence a um grupo primário (definido em `/etc/passwd`) e pode pertencer a grupos secundários.

Gerenciamento de Grupos:

- **Criar grupo:** `sudo groupadd equipe` cria o grupo "equipe".
- **Excluir grupo:** `sudo groupdel equipe` remove o grupo.
- **Adicionar usuário a grupo:** `sudo usermod -aG equipe joao` adiciona "joao" ao grupo "equipe".
- **Remover usuário de grupo:** `sudo gpasswd -d joao equipe` remove "joao" do grupo.
- **Verificar grupos:** `groups joao` lista os grupos do usuário "joao".

Permissões

```
devnet@lostlap ~ $ ls -l
total 32
drwxr-xr-x  4 devnet devnet 4096 2009-09-28 05:13 Desktop
drwxr-xr-x  6 devnet devnet 4096 2009-09-25 07:23 Documents
drwxr-xr-x 49 devnet devnet 4096 2009-09-25 07:23 Music
drwxr-xr-x  2 devnet devnet 4096 2009-09-25 07:11 Network
drwxr-xr-x  2 devnet devnet 4096 2009-09-25 07:04 Pictures
drwxr-xr-x  2 devnet devnet 4096 2009-09-25 07:11 Public
drwxr-xr-x  2 devnet devnet 4096 2009-09-25 07:11 Templates
drwxr-xr-x  2 devnet devnet 4096 2009-09-25 07:11 Videos
```

Diagram illustrating the components of the `ls -l` output:

- File Type:** The first character (e.g., `d` for directory, `-` for file).
- User Permissions:** The next three characters (e.g., `rwx` for read, write, execute).
- Group Permissions:** The next three characters (e.g., `rxr` for read, execute).
- Other Permissions:** The final character (e.g., `x` for execute).
- # of Hard Links:** The number following the permissions (e.g., `4`).
- User / Owner:** The user name (e.g., `devnet`).
- Group:** The group name (e.g., `devnet`).
- Size:** The file size in bytes (e.g., `4096`).
- Date:** The last modification date and time (e.g., `2009-09-28 05:13`).
- File or Directory Name:** The name of the file or directory (e.g., `Desktop`).

Legend for permissions:

- `d` - directory
- `r` - readable
- `w` - writeable
- `x` - executable

As permissões determinam o que usuários e grupos podem fazer com arquivos e diretórios. Elas são divididas em três categorias:

- **Dono (user):** O usuário que criou ou possui o arquivo.
- **Grupo (group):** Usuários no grupo associado ao arquivo.
- **Outros (others):** Todos os outros usuários do sistema.

Tipos de Permissões:

- **Leitura (r):** Permite visualizar o conteúdo de um arquivo ou listar um diretório.
- **Escrita (w):** Permite modificar ou excluir um arquivo ou diretório.
- **Execução (x):** Permite executar um arquivo (como um script) ou entrar em um diretório.

Visualização de Permissões:

O comando `ls -l` exibe permissões no formato `-rwxr-xr-x`, onde:

- O primeiro caractere indica o tipo de arquivo (- para arquivo regular, `d` para diretório).
- Os próximos nove caracteres mostram permissões para dono (`rwx`), grupo (`r-x`) e outros (`r-x`).
- Exemplo: `-rwxr-xr-x` significa que o dono tem leitura, escrita e execução, enquanto grupo e outros têm apenas leitura e execução.

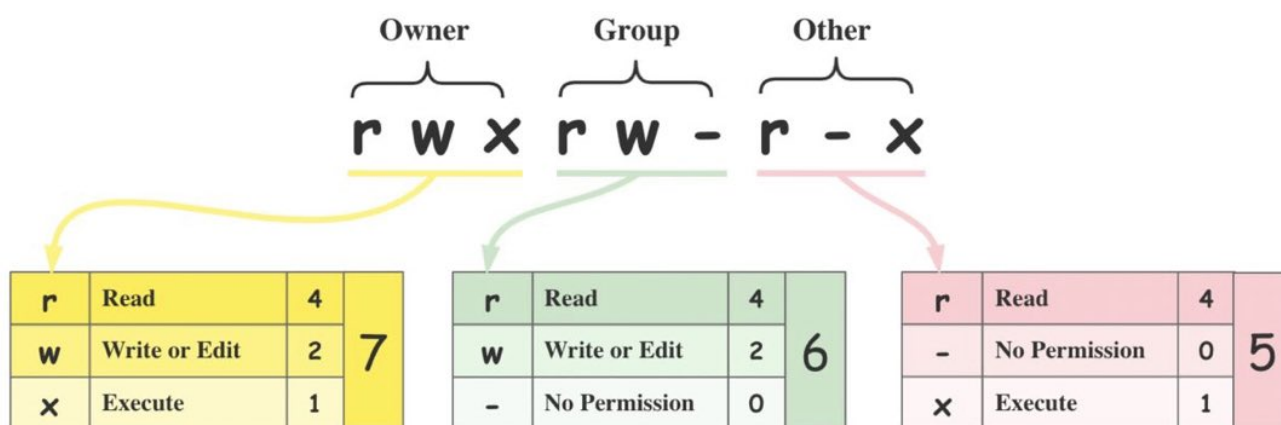
Notação Numérica:

Permissões são representadas por números:

- Leitura (`r`) = 4
- Escrita (`w`) = 2
- Execução (`x`) = 1

Soma-se os valores para cada categoria. Exemplo: `755` significa `rwx` (7=4+2+1) para o dono e `r-x` (5=4+1) para grupo e outros.

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwX	Read + Write + Execute



Gerenciamento de Permissões:

- **Mudar permissões:** `chmod 755 arquivo.sh` (numérico) ou `chmod u+x arquivo.sh` (simbólico, adiciona execução ao dono).
- **Mudar dono:** `sudo chown joao arquivo.sh` altera o dono para "joao".
- **Mudar grupo:** `sudo chgrp equipe arquivo.sh` altera o grupo para "equipe".
- **Recursividade:** Use `-R` para aplicar a diretórios e subdiretórios, como `chmod -R 755 pasta`.

Permissões Especiais

- **SetUID (SUID):** Permite executar um arquivo com as permissões do dono. Configurado com `chmod u+s arquivo`. Exemplo: O comando `passwd` usa SUID para permitir que usuários alterem suas senhas. Configurado com `chmod u+s arquivo`.
- **SetGID (SGID):** Para diretórios, novos arquivos herdam o grupo do diretório. Configurado com `chmod g+s pasta`.
- **Sticky Bit:** Impede que usuários excluam arquivos em um diretório que não possuem permissão. Configurado com `chmod o+t pasta`. Exemplo: `/tmp` usa Sticky Bit.

Gerenciamento de pacotes

```
homehost@homehost:~$ sudo apt-get update
Hit:1 http://br.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://br.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://br.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:7 https://dln.mariadb.com/repo/mariadb-server/10.6/repo/ubuntu jammy InRelease [7.767 B]
Get:8 https://dln.mariadb.com/repo/maxscale/latest/apt jammy InRelease [9.344 B]
Get:4 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8.993 B]
Hit:5 https://downloads.mariadb.com/Tools/ubuntu jammy InRelease
Hit:9 https://repo.zabbix.com/zabbix-agent2-plugins/1/ubuntu jammy InRelease
Hit:10 https://repo.zabbix.com/zabbix/6.0/ubuntu jammy InRelease
Hit:11 https://download.docker.com/linux/ubuntu jammy InRelease
Get:12 https://packages.grafana.com/enterprise/deb stable InRelease [5.984 B]
Hit:13 https://dl.google.com/linux/chrome/deb stable InRelease
Get:14 https://packages.grafana.com/enterprise/deb beta InRelease [5.976 B]
Get:15 http://br.archive.ubuntu.com/ubuntu jammy-updates/main amd64 DEP-11 Metadata [101 kB]
Get:16 http://br.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 DEP-11 Metadata [289 kB]
Get:17 http://br.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 DEP-11 Metadata [940 B]
Get:18 http://br.archive.ubuntu.com/ubuntu jammy-backports/main amd64 DEP-11 Metadata [101 kB]
```

O **apt** (*Advanced Package Tool*) é o gerenciador de pacotes padrão para distribuições Linux baseadas em Debian, como Ubuntu, Debian e Kali Linux, que gerencia pacotes de software no formato **.deb** (Debian). Ele interage com repositórios online, que são servidores que armazenam pacotes e suas versões, listados no arquivo `/etc/apt/sources.list` ou em `/etc/apt/sources.list.d/`. O **apt** automatiza tarefas como:

- Baixar pacotes de repositórios.
- Resolver dependências (outros pacotes necessários para o software funcionar).
- Instalar, atualizar ou remover software.
- Manter o sistema atualizado com as versões mais recentes dos pacotes.

Como o apt Funciona?

O **apt** opera em duas camadas:

- **Backend:** Usa ferramentas de baixo nível como **dpkg**, que lida diretamente com a instalação de arquivos **.deb**. O **apt** é uma interface mais amigável que coordena essas ações.
- **Repositórios:** O **apt** consulta o arquivo `/etc/apt/sources.list` para saber onde buscar pacotes. Um repositório típico pode ser:

```
deb http://deb.debian.org/debian bullseye main
```

Isso indica que o **apt** buscará pacotes da distribuição Debian Bullseye, na seção "main".

O **apt** mantém um cache local com a lista de pacotes disponíveis, atualizado pelo comando **apt update**. Isso garante que o sistema saiba quais são as versões mais recentes antes de instalar ou atualizar algo.

Comandos usuais

apt update

Atualiza o cache local com a lista de pacotes disponíveis nos repositórios, garantindo que o sistema conheça as versões mais recentes.

```
sudo apt update
```

apt upgrade

Atualiza todos os pacotes instalados para suas versões mais recentes, respeitando as dependências.

```
sudo apt upgrade
```

apt install

Instala um novo pacote. O **apt** resolve automaticamente as dependências necessárias.

```
sudo apt install nome-do-pacote
```

apt remove

Remove um pacote instalado, mas mantém os arquivos de configuração. Útil para desinstalar software sem perder configurações.

```
sudo apt remove nome-do-pacote
```

apt purge

Remove um pacote e seus arquivos de configuração, liberando espaço no sistema.

```
sudo apt purge nome-do-pacote
```

apt autoremove

Remove pacotes que foram instalados como dependências, mas não são mais necessários. Isso ajuda a manter o sistema limpo e livre de pacotes desnecessários.

```
sudo apt autoremove
```

apt search

Busca por pacotes disponíveis nos repositórios. Útil para encontrar software específico.

```
apt search nome-do-pacote
```

apt show

Exibe informações detalhadas sobre um pacote específico, como versão, descrição e dependências.

```
apt show nome-do-pacote
```

apt list

Lista todos os pacotes instalados ou disponíveis nos repositórios. Útil para verificar o que está instalado ou disponível para instalação.

```
apt list --installed # Lista pacotes instalados
```

Shell Scripting

```
> newscript.sh x
1  ▶  #!/bin/bash
2
3  function greeting() {
4      hello="Hello, $name"
5      echo "$hello"
6  }
7
8  echo "Enter name"
9  read name
10
11 val=$(greeting)
12 echo "Return value of the function is $val"
```

Shell scripting é a prática de criar scripts, que são arquivos de texto contendo uma sequência de comandos que o shell executa automaticamente. Esses scripts são usados para automatizar tarefas, gerenciar sistemas e

realizar operações complexas de forma eficiente.

- **Automação:** Permite executar tarefas repetitivas, como backups ou verificação de processos, sem intervenção manual.
- **Customização:** Usuários podem criar scripts personalizados para atender a necessidades específicas, como para análise de logs.
- **Administração de Sistemas:** Scripts são amplamente usados para gerenciar usuários, instalar software e configurar servidores.
- **Eficiência:** Reduz o tempo necessário para realizar tarefas complexas, especialmente em ambientes de servidores ou segurança.

Como Criar e Executar um Script

1. Crie o arquivo:
 - Use um editor como nano ou vim: `nano script.sh`.
 - Comece com a linha `#!/bin/bash` (*shebang*), que indica que o Bash executará o script.
2. Escreva comandos: Adicione comandos como se estivesse no terminal.
3. Torne executável: Use `chmod +x script.sh` para dar permissão de execução.
4. Execute: Rode com `./script.sh` ou `bash script.sh`.

Conceitos Fundamentais

Pipes (|)

Pipes conectam a saída de um comando à entrada de outro, criando fluxos de dados. São essenciais para combinar comandos e processar informações.

A saída de um comando (à esquerda do `|`) é enviada como entrada para o próximo comando. Exemplo: `ls | grep "txt"` lista arquivos e filtra apenas os que contêm "txt" no nome.

```
ps aux | grep "nginx" | awk '{print $2}' > pids.txt
```

Este comando lista processos, filtra os do Nginx, extrai seus IDs (coluna 2 com `awk`) e salva em `pids.txt`.

Redirecionamento

Redirecionamento controla a entrada e saída de comandos.

- **Saída (>):** Salva a saída em um arquivo, sobrescrevendo. Exemplo: `ls > lista.txt`.
- **Saída anexada (>>):** Adiciona à saída sem sobrescrever. Exemplo: `echo "Novo log" >> log.txt`.
- **Entrada (<):** Usa o conteúdo de um arquivo como entrada. Exemplo: `grep "erro" < log.txt`.

Variáveis

Variáveis armazenam dados, como strings ou números, para uso posterior no script.

`nome="valor"` (sem espaços). Exemplo: `diretorio="/home/user"`. Acesse com `$nome` ou `${nome}`.
Exemplo: `echo $diretorio`.


```
arquivo="log.txt"
echo "Verificando o arquivo $arquivo"
cat "$arquivo"
```

Condicionais (if)

Condicionais permitem executar comandos com base em condições, como verificar se um arquivo existe.

Sintaxe:

```
if [ condição ]; then
    comando
else
    outro_comando
fi
```

Exemplo:

```
if [ -f "arquivo.txt" ]; then
    echo "Arquivo existe!"
else
    echo "Arquivo não encontrado."
fi
```

Condições comuns:

- `-f arquivo`: Verifica se o arquivo existe.
- `-d diretorio`: Verifica se o diretório existe.
- `$var1 -eq $var2`: Verifica igualdade numérica.

Loops

Loops permitem repetir comandos. Os mais comuns são `for` e `while`.

For

```
for item in lista; do
    comando
done
```

Lista todos os arquivos `.txt`:

```
for arquivo in *.txt; do
    echo "Arquivo: $arquivo"
```

```
done
```

While

```
while [ condição ]; do  
    comando  
done
```

Monitora um processo até ele terminar:

```
while ps aux | grep "nginx" > /dev/null; do  
    echo "Nginx ainda está rodando..."  
    sleep 1  
done
```

Praticando

<https://overthewire.org/wargames/bandit/>

<https://tryhackme.com/module/linux-fundamentals>

<https://academy.hackthebox.com/course/preview/linux-fundamentals>

Links úteis

Livro *Effective Shell*: <https://effective-shell.com/>

TLDR pages. Simplifica as páginas do manual com exemplos práticos: <https://tldr.sh/>

Sistema de permissão: <https://www.redhat.com/en/blog/suid-sgid-sticky-bit>

Bash Scripting CheatSheet: <https://devhints.io/bash>