

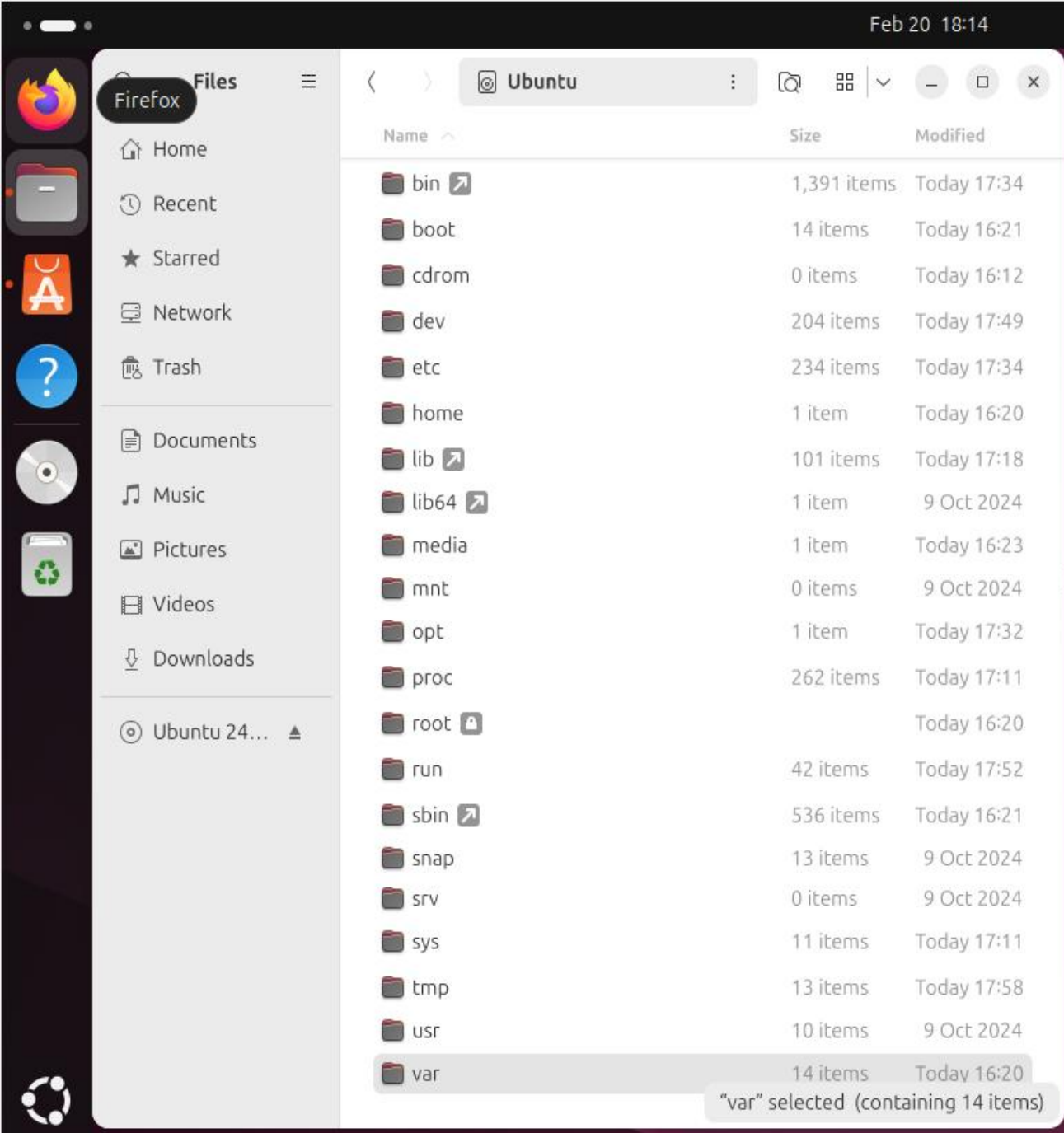
Professor
Massaki

SISTEMAS OPERACIONAIS ABERTOS: Linux

(Parte 1)



Sistema de Arquivos



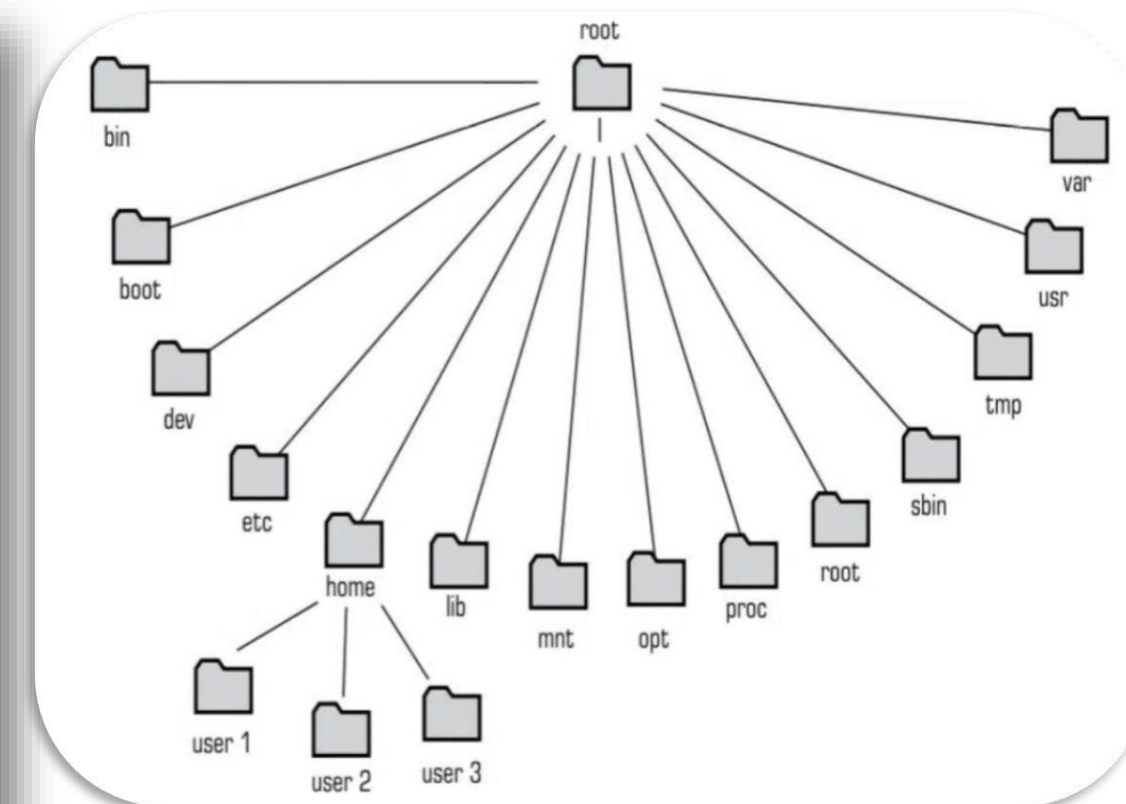
The screenshot shows the Ubuntu file manager interface. The left sidebar lists common locations like Home, Recent, Starred, Network, Trash, Documents, Music, Pictures, Videos, and Downloads. The main pane displays the root directory structure with columns for Name, Size, and Modified. The 'var' directory is selected, showing it contains 14 items.

Name	Size	Modified
bin	1,391 items	Today 17:34
boot	14 items	Today 16:21
cdrom	0 items	Today 16:12
dev	204 items	Today 17:49
etc	234 items	Today 17:34
home	1 item	Today 16:20
lib	101 items	Today 17:18
lib64	1 item	9 Oct 2024
media	1 item	Today 16:23
mnt	0 items	9 Oct 2024
opt	1 item	Today 17:32
proc	262 items	Today 17:11
root		Today 16:20
run	42 items	Today 17:52
sbin	536 items	Today 16:21
snap	13 items	9 Oct 2024
srv	0 items	9 Oct 2024
sys	11 items	Today 17:11
tmp	13 items	Today 17:58
usr	10 items	9 Oct 2024
var	14 items	Today 16:20

"var" selected (containing 14 items)

A identificação dos objetos de um sistema de arquivo no Linux é conhecida como **inode**. Ele carrega as informações de onde o objeto está localizado no disco, informações de segurança, data e hora de criação e última modificação dentre outras. Quando criamos um sistema de arquivos no Linux, cada dispositivo tem um número finito de inodes que será diretamente proporcional ao número de arquivos que este dispositivo poderá acomodar.

Basicamente no Linux tudo é um arquivo, que dependendo do seu tipo, pode se tornar um arquivo comum, um diretório, um link, um socket, um condutor, um descritor de dispositivos, etc.



Comparando com o Windows, a organização dos diretórios no Linux é um pouco mais complexa. O sistema de arquivos no Linux é semelhante a uma árvore de cabeça para baixo. **No topo da hierarquia do Linux existe um diretório raiz nomeado simplesmente como root e identificado como o sinal "/"**. Não confunda diretório raiz (root) com o superusuário root.

Sistema de Arquivos

/

Diretório raiz do sistema de arquivos;

/bin

Contêm os comandos que podem ser utilizados pelos usuários e pelo administrador do sistema. São requeridos no modo monousuário ou de manutenção (single-user mode) e também podem conter comandos que são utilizados indiretamente por alguns scripts. Nele ficam os arquivos executáveis, tais como: cat, chgrp, **chmod**, chown, cp, date, dd, df, dmesg, echo, hostname, kill, ln, more, mount, mv, ps, pwd, rm, rmdir, sed, su, umount e uname;

/boot

Arquivos estáticos necessários à carga do sistema. É onde fica localizada a imagem do Kernel, initramfs e alguns arquivos do Grub. Este diretório contém tudo que é necessário para carregar o sistema, exceto os arquivos de configuração e o gerenciador de boot. O /boot inclui o setor master de carga do sistema (master boot sectors) e arquivos de mapa de setor.

/dev

Abstração do Kernel onde ficam os arquivos para acesso dos dispositivos do sistema, como discos, cd-roms, pendrives, portas seriais, terminais, etc. Os arquivos são descritores que facilitam o acesso aos dispositivos. Este diretório é um pseudo-filesystem, e não existe no disco. Seu conteúdo, por exemplo, tem descritores de dispositivos como /dev/sda, /dev/cdrom, etc.

/etc

Arquivos necessários à configuração do sistema. São necessários para a carga do sistema operacional. Ele possui arquivos importantes tais como: fstab, exports, passwd, shadow, group, hosts, hosts.allow, hosts.deny, inittab, ld.so.conf, mtab, profile, services, etc. Nele também residem os arquivos de configuração das interfaces de rede.

Tipicamente /etc possui dois subdiretórios:

/etc/X11: Arquivos de configuração para a interface gráfica do Linux (X Window System);

/etc/skel: Esqueletos da configuração usuários. No diretório /etc/skel ficam localizados os arquivos de “modelo” para os usuários. O conteúdo deste diretório é replicado para o diretório home dos novos usuários quando são criados no sistema.

/home

Geralmente é neste diretório onde ficam os diretórios home dos usuários. Nestes diretórios localizam-se scripts de carga de perfil e do shell bash de cada usuário.

Sistema de Arquivos

/lib

Arquivos de bibliotecas essenciais ao sistema, utilizadas pelos programas em /bin e módulos do Kernel. É comum existir um diretório /lib[arquitetura]. Nos processadores de 64 bits, existe o diretório /lib64. Nos processadores de 32 bits, deve existir um diretório /lib32.

/mnt

Diretório vazio utilizado como ponto de montagem de dispositivos na máquina. Usado pelo administrador para montar discos.

/media

Diretório vazio utilizado como ponto de montagem de dispositivos na máquina, tais como cdrom, dvd, pendrives, etc.

/proc

Informações do Kernel e dos processos. É um pseudo-filesystem e não existe realmente no disco. Neste diretório ficam as abstrações de descritores de tudo quanto há no Kernel do sistema. É possível não só ler os dados, bem como fazer alterações no comportamento do Kernel alterando o conteúdo de arquivos em /proc. Fazendo uma abstração do Windows, este diretório seria o “registro” do sistema.

/opt

Neste diretório ficam instalados os aplicativos que não são da distribuição do Linux, como por exemplo, banco de dados de terceiros, software de vetores Cad, etc.

Sistema de Arquivos

/root

Diretório home do superusuário root. Dependendo da distribuição pode estar presente ou não;

/run

Este diretório contém informações do sistema desde a última carga. Os arquivos deste diretório são apagados ou zerados no início do processo de boot. Arquivos como aqueles que indicam o PID do processo em execução devem residir neste diretório.

/sbin

Arquivos essenciais ao sistema, como aplicativos e utilitários para a administração da máquina. Normalmente só o superusuário tem acesso a estes arquivos, tais como: fdiskm fsck, ifconfig, init, mkfs, mkswap, **route**, reboot, swapon e swapoff.

/tmp

Diretório de arquivos temporários. Em algumas distribuições este diretório é montado em memória. Recomenda-se que seu conteúdo seja apagado de tempos em tempos ou a cada reboot.

Sistema de Arquivos

/usr

Arquivos pertencentes aos usuários e a segunda maior hierarquia de diretórios no Linux. Seu conteúdo pode ser distribuído via rede para diversos sistemas Linux da mesma distribuição sem problema algum. Ele tem alguns subdiretórios a saber:

/usr/bin: Ferramentas e comandos auxiliares de usuários, bem como interpretadores de programação, como o perl, python, etc;

/usr/include: Cabeçalhos e bibliotecas da linguagem C;

/usr/lib e **/usr/lib64:** bibliotecas de aplicações de usuários;

/usr/libexec: binários que não são executados normalmente por usuários;

/usr/local: hierarquia de diretórios para instalação de aplicativos locais no sistema. Possui vários subdiretórios como bin, etc, include, lib, man, sbin, share e src;

/usr/sbin: contém ferramentas não essenciais, mas exclusivas da administração do sistema.

/usr/share: arquivos de somente leitura de arquitetura independente. São arquivos que podem ser compartilhados entre distribuições de Linux iguais, independente da arquitetura. O diretório man por exemplo é onde residem os manuais dos comandos e fica em /usr/share.

/usr/src: pode conter arquivos de código fonte de programas.

Sistema de Arquivos

/var

Diretório onde são guardadas informações variáveis sobre o sistema, como arquivos de log, arquivos de e-mail etc. Possui subdiretórios importantes, a saber:

/var/cache: mantém informações de cache das aplicações como cálculos, etc;

/var/lib: mantém informações de estado das aplicações;

/var/lock: mantém arquivos de trancamento que retém dispositivos para uso exclusivo de alguma determinada aplicação;

/var/log: mantém os arquivos de log do sistema, tais como messages, lastlog e wtmp.

/var/mail: mantém os diretórios de contas de email dos usuários;

/var/opt: mantém os arquivos variáveis das aplicações;

/var/run: a funcionalidade deste diretório foi movida para o /run.

/var/spool: mantém dados de processos que mantém filas de arquivos, tais impressão e saída de email;

/var/tmp: mantém os arquivos temporários das aplicações que precisem ser preservados entre o reboot do sistema

A Interface linux



É um tanto contraditório nos dias de hoje ainda se usar uma interface denominada Terminal, que parece reviver a década de 1960, onde digitam-se textos de código e tem um fundo preto com aparência simplista e minimalista.

Esta interface é chamada **Shell** e tem sido uma forma comum de interagir com computadores desde que as primeiras telas e teclados foram criados. Motivos pelos quais usá-la:

- Rápida digitação:** Um usuário de *shell* habilidoso pode manipular um sistema em velocidades estonteantes usando apenas um teclado. Digitar comandos é geralmente *muito* mais rápido do que explorar interfaces de usuário com um mouse
- São programáveis:** os usuários geralmente programam enquanto trabalham em um shell, criando scripts para automatizar processos demorados ou repetitivos.
- São portáteis:** um *shell* pode ser usado para fazer interface com quase qualquer tipo de computador, de um mainframe a um Raspberry Pi, de uma maneira muito semelhante.

Mas antes precisaremos entender como se divide a arquitetura Unix, as demais interfaces e alguns dos principais comandos usados na interface shell do linux.

Arquitetura Unix/Linux

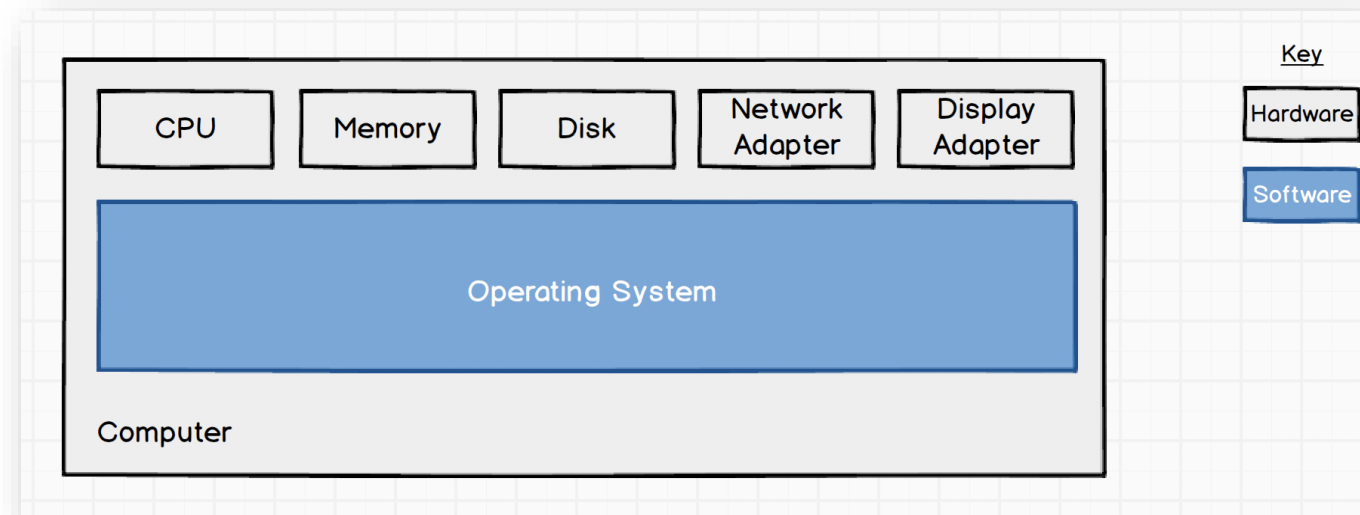
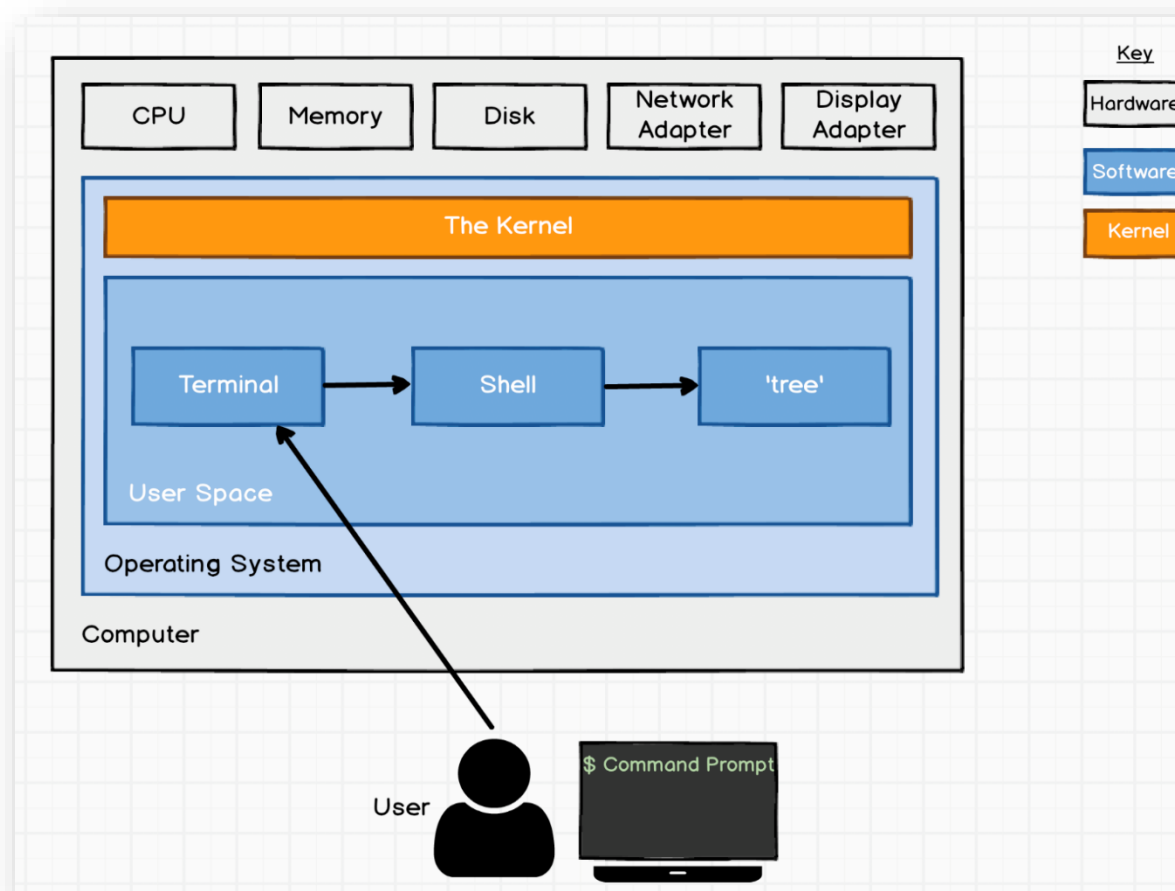


Figura 1 - Visão simplificada de um computador típico

O Sistema Operacional: é o pedaço de software instalado em um computador que pode interagir com o *hardware*. Sem hardware, como uma CPU, memória, um adaptador de rede, uma placa gráfica, unidades de disco e assim por diante, não há muito que você possa fazer com o computador. **O sistema operacional é a interface primária para este hardware.** Nenhum programa normal falará com o hardware diretamente - o sistema operacional abstrai este hardware e fornece uma interface *de software* para ele.

Seu computador é composto por uma CPU, memória e muito provavelmente um adaptador de rede e um adaptador de vídeo. Não menos importante, terá pelo menos um disco rígido. Para PCs domésticos, provavelmente também haverá diversos periféricos (mouse, teclado, impressoras, pen drives, webcam, etc.). Tudo isto será coordenado pelo **Kernel Linux**.



Kernel Linux

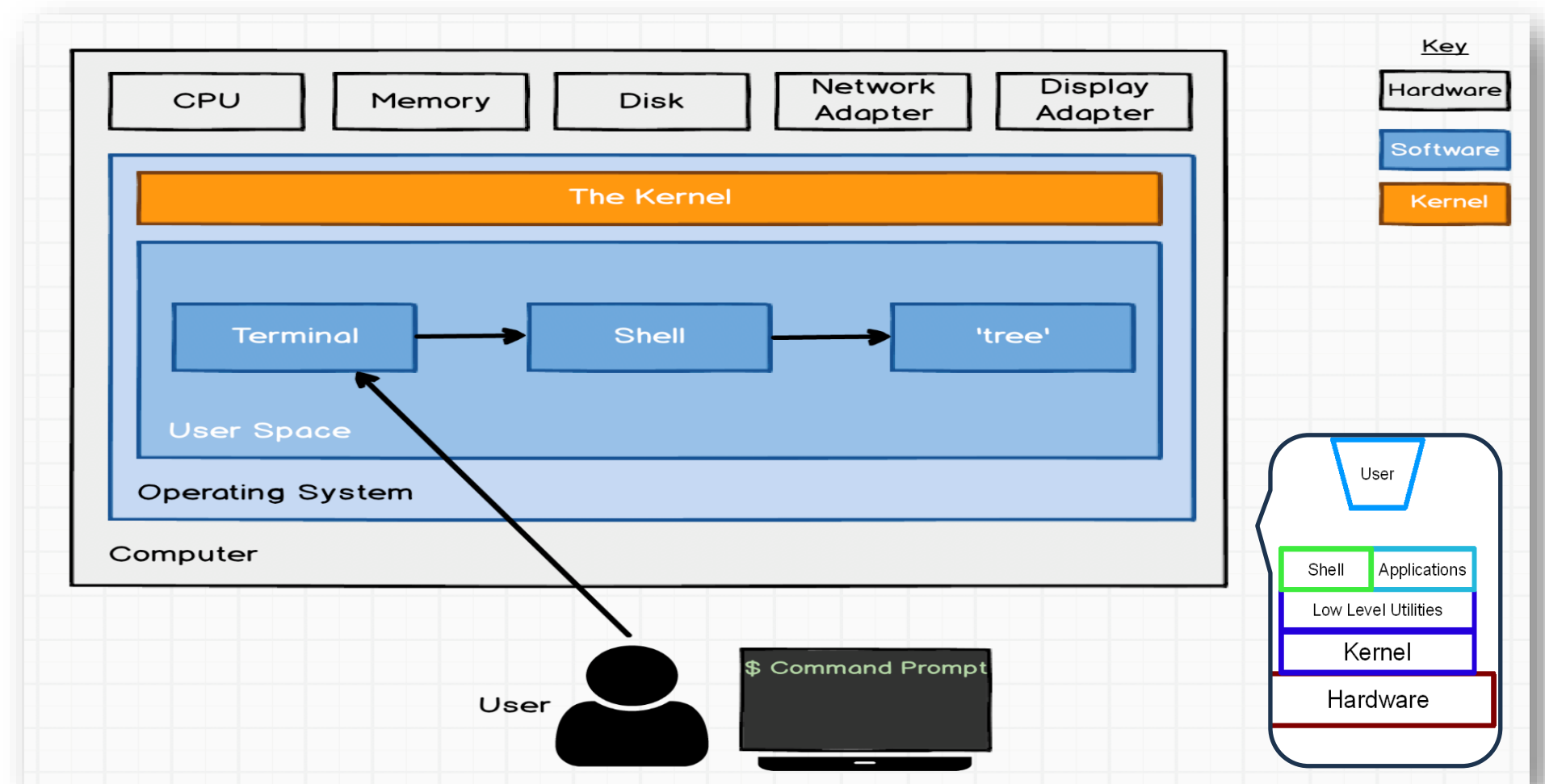
O **Kernel** (ou "núcleo") é o coração de um sistema operacional, tanto no Linux quanto no Unix. Ele é o responsável por gerenciar o hardware do computador (como memória, processador e dispositivos) e garantir que todos os programas do sistema possam interagir com ele de maneira organizada.

📌 **Analogia:** O **Kernel** é como o motor do carro 🚗. Ele funciona o tempo todo e faz o carro andar, mas o motorista (usuário) não precisa entendê-lo para dirigir o carro.

Arquitetura Unix/Linux

A abstração que o sistema operacional fornece é essencial. Os desenvolvedores não precisam saber as especificidades de como **trabalhar com dispositivos** individuais de diferentes fornecedores; o sistema operacional fornece uma interface **padronizada** para tudo isso. Ele também lida com várias tarefas, como garantir que o sistema seja inicializado corretamente.

O sistema operacional é geralmente **dividido em duas partes: o kernel e o espaço do usuário:**



O Kernel (“Núcleo”): é a parte do sistema operacional responsável pelas tarefas mais sensíveis (interagir com dispositivos físicos, gerenciar os recursos disponíveis para usuários e programas, iniciar os vários sistemas necessários, etc.). O software em execução no kernel tem acesso direto aos recursos, então é *extremamente* sensível. O kernel equilibrará os recursos entre os programas no espaço do usuário. Se você já teve que instalar 'drivers', estes são exemplos de partes do software que serão executadas no kernel - elas terão acesso direto a um dispositivo físico que você instalou e o exporão ao resto do software no computador.

Analogamente o **kernel é a parte macia e comestível de uma noz** ou semente, que é cercada por uma casca. Ao lado você pode ver uma noz - o kernel é a parte macia no meio, e a casca a envolve e protege. Esta analogia nos ajuda a entender as partes de um computador e o SO Linux.



Arquitetura Unix/Linux

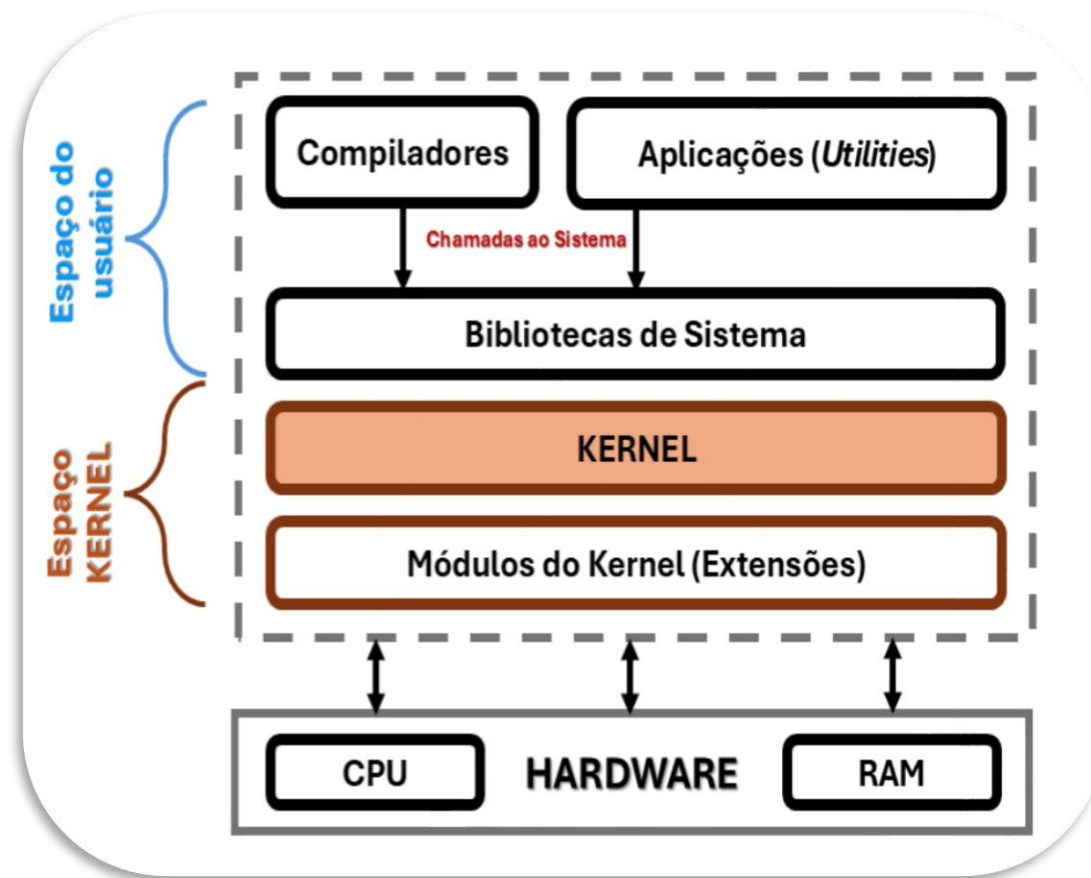


Figura 2- Kernel e espaço de usuário Linux (adaptado de BURGUERA et. al., 2011)

Espaço do usuário: A grande maioria dos programas é executada no "espaço do usuário" (também popularmente denominado "território do usuário"). Quando um programa inicia, o kernel alocará a ele um segmento privado de memória e fornecerá acesso *limitado* aos recursos. O programa recebe acesso a uma biblioteca de funções do sistema operacional, que ele pode usar para acessar recursos como arquivos, dispositivos e assim por diante. Programas no espaço do usuário estão essencialmente em *sandboxes*¹, onde há um limite para quanto dano eles podem causar. Por exemplo, um programa em execução no espaço do usuário pode usar a função *fopen*, que é fornecida em quase todos os sistemas operacionais como parte da Biblioteca Padrão C. Isso permite que um programa tente abrir um arquivo.

¹**Sandbox** é uma **área isolada e controlada** onde programas podem ser executados de maneira restrita e sem afetar o resto do sistema. O conceito de **sandbox** vem da ideia de uma caixa de areia, onde você pode brincar de forma segura, mas sem que a areia saia da caixa e cause danos ao ambiente ao redor.

FUNCIONAMENTO:

Uma **sandbox** cria um ambiente **virtual isolado** dentro do sistema operacional, onde um programa ou processo pode ser executado sem ter acesso direto ao sistema ou aos dados sensíveis do usuário. Em outras palavras, a sandbox impede que o programa acesse outras partes do sistema ou execute ações prejudiciais fora de seu ambiente controlado.

O sistema operacional tomará uma decisão sobre se o programa tem *permissão* para abrir o arquivo (com base em coisas como permissões, onde o arquivo está e assim por diante) e então, se estiver OK com a chamada, dará ao programa acesso ao arquivo. Por baixo dos panos, essa chamada de 'espaço do usuário' se traduz em uma chamada de sistema no kernel.

Agora que os componentes-chave foram introduzidos, *podemos olhar para o terminal, shell entre outros conceitos básicos*. O nome não deve ser nenhuma surpresa, pois é um *wrapper* ou camada externa para o sistema operacional (que contém o nugget sensível do kernel).

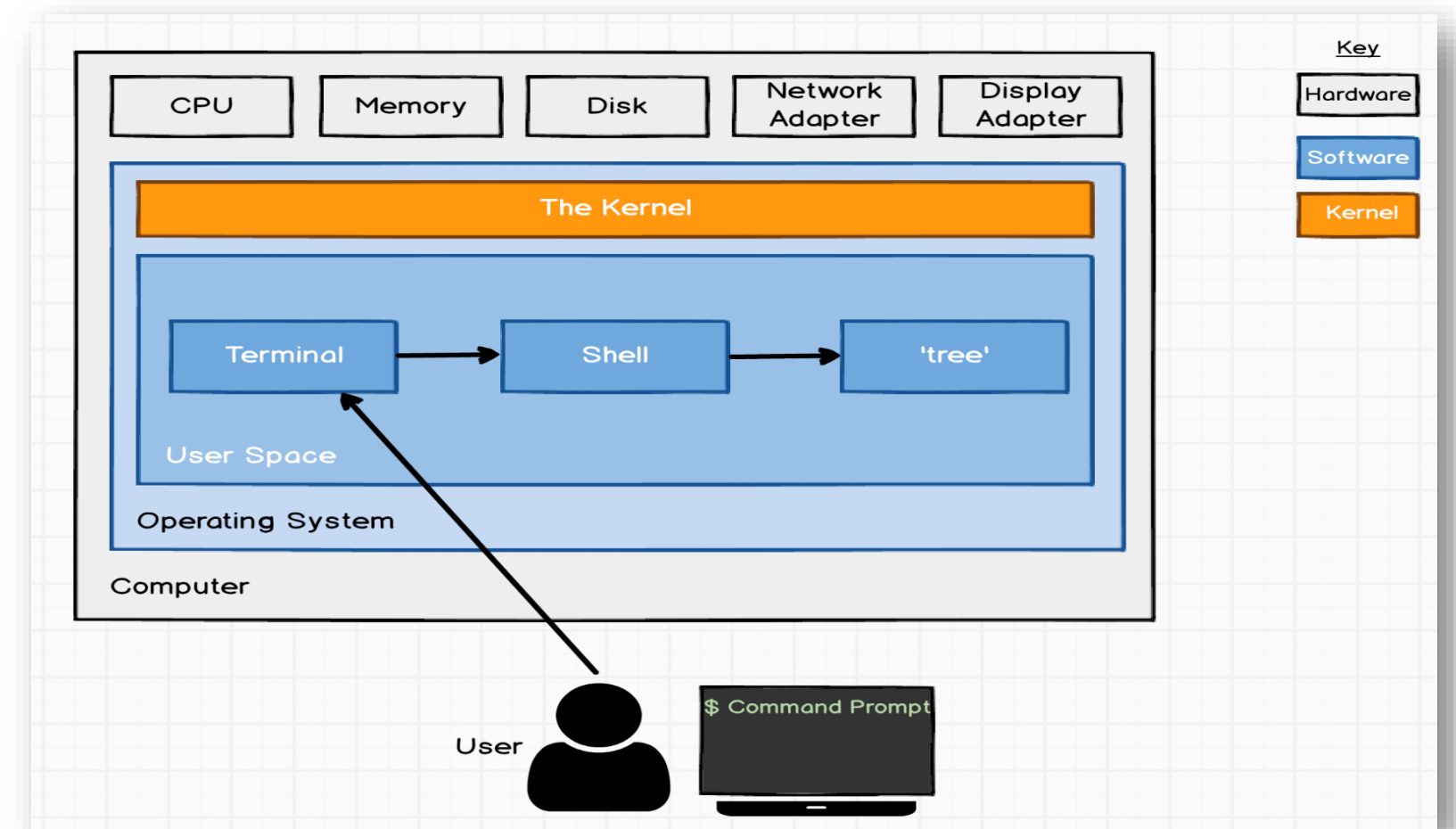
Conforme a figura ao lado, há um *usuário* que está interagindo com um *terminal*, que está executando uma interface *shell*, que está mostrando um *prompt de comando*. Este usuário escreveu um comando que está chamando um programa. Então Vamos dissecar para entender como tudo isso funciona aos poucos:

Não estamos interagindo *diretamente* com o 'shell' neste diagrama. Na verdade, estamos usando um *terminal*. Quando um usuário quer trabalhar com um shell interativamente, usando um teclado para fornecer entrada e um display para ver a saída na tela, o usuário usa um *terminal*.

Um terminal é apenas um programa que lê a entrada do teclado, passa essa entrada para outro programa (normalmente um *shell*) e exibe os resultados na tela. Um programa *shell* por si só não faz isso - ele requer um terminal como interface.

Por que a palavra *terminal*? Isso faz sentido quando você olha como as pessoas interagiam com computadores historicamente. A entrada para um computador pode ser por meio de cartões perfurados, e a saída geralmente é por meio de uma impressora. O *Teletype Terminal*⁸ se tornou uma maneira comum para os usuários interagirem com computadores.

Olá Linux!



O Terminal

No passado os computadores eram máquinas muito grandes, complexas e caras. Era comum ter *muitos* terminais conectados a uma única máquina grande (ou "*mainframe*"), ou alguns terminais que as pessoas compartilhavam. Mas o terminal em si era apenas uma interface humana para o Sistema Operacional. Um exemplo de terminal usado no passado é o conhecido **TERMINAL IBM 3486**.



Mas o **terminal** ainda é muito usado atualmente. **Por exemplo, em um data center**, não vou e encontro a máquina, conecto um teclado e um monitor e faço interface diretamente com a rede de computadores; **onde executo um *programa de terminal* no meu computador para fornecer acesso à(s) máquina(s) remota(s)**. Meu programa de terminal me permite usar meu teclado e monitor para trabalhar com uma máquina remota - **tudo por meio de um *shell seguro* - que é uma conexão de shell seguro em uma rede**.

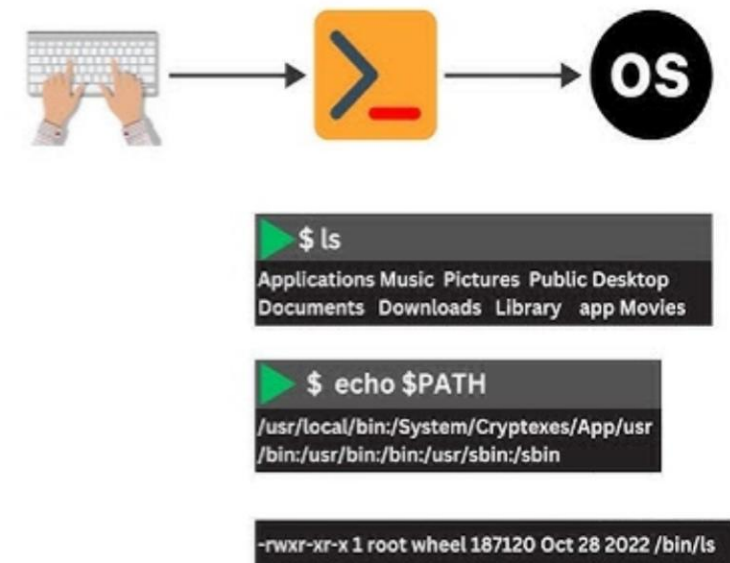
Portanto, terminais são, de muitas maneiras, **bem simples - são interfaces**. Mas como são programas bem simples, não podemos fazer muita coisa com eles. Então, normalmente, a primeira coisa que um programa terminal fará é executar um programa *shell* - um programa que podemos usar para operar o computador. **Não há nada de especial sobre terminais - qualquer um pode escrever um programa para operar como um terminal, e é por isso que você verá muitos terminais diferentes por aí**. Exemplos são o aplicativo padrão 'terminal' para MacOS X, o [gnome-terminal](#) para Linux, e [iTerm2](#) e [Hyper](#).

SHELL (A interface Usuário-Linux)

No mundo Linux, frequentemente encontramos termos como shell, script ou bash. Para entender bem o tópico, precisamos começar com conceitos básicos. O objetivo do shell do sistema Linux é facilitar a comunicação entre o usuário e o kernel do sistema; já que o kernel atua como cérebro no sistema, tendo acesso aos componentes de hardware do nosso computador, responsável, por exemplo, por gerenciar drivers ou memória.

Os comandos servem como a interface de mais alto nível com a qual os usuários interagem por meio do terminal. O shell mais comumente usado no Linux é o Bash (Bourne Again Shell), baseado no Bourne shell (Sh). Se o bash é apenas um shell, agindo como uma camada de abstração entre programas e o kernel, você pode se perguntar: o que é um script? Nós nos apressamos em responder – um script é simplesmente um arquivo que contém comandos para um shell específico. Então, um script Bash é apenas um arquivo contendo uma sequência de comandos para o shell Bash. O uso de scripts permite maior confiabilidade, maior grau de automação no trabalho do administrador e, conseqüentemente, otimização das operações realizadas no sistema Linux.

| What is a Shell ?



Suscintamente, o Shell é o programa que recebe os comandos digitados pelo usuário e os envia para o Kernel processar!

- 📌 **Tipos de Shells:** Bash (mais comum), Zsh, Fish, Dash
- 📌 **Principais Funções:** Interpretar comandos (Ex.: mkdir, rm, ls, etc.).
Permitir a criação de **scripts para automação**.

📌 **Exemplo** (Quando digitamos):

```
$ echo "Olá, Linux!"
```

Olá Linux!

BASH (A interface Usuário-Linux)

Bash? (O Shell mais popular 🖥️)

📌 Definição:

O **Bash** (*Bourne Again Shell*) é o **tipo de Shell mais usado no Linux**. Ele permite executar comandos, criar scripts e automatizar tarefas.

📌 Analogia:

Se o **Shell** é um garçom, o **Bash** é o tipo de treinamento que ele recebeu para entender melhor os pedidos! Ele possui mais recursos e facilita a vida do usuário.

📌 Exemplo de comando no Bash:

```
bash
```

📄 Copiar

✎ Editar

```
echo "Olá, mundo!"
```

👉 Esse comando faz o Bash exibir Olá, mundo! na tela.

📌 **Dica:** Para saber qual Shell você está usando, digite:

```
bash
```

📄 Copiar

✎ Editar

```
echo $SHELL
```

Se aparecer **/bin/bash**, você está usando o Bash!



Low-Level Utilities

(Ferramentas Essenciais 🛠️)

📌 Definição:

São **ferramentas e comandos básicos** que permitem a interação com o sistema.

📌 Função:

- Realizam tarefas como **gerenciamento de arquivos, usuários e redes**.
- Alguns exemplos de **Low-Level Utilities** no Linux:
 - ls → Lista arquivos
 - cd → Navega entre diretórios
 - cp → Copia arquivos
 - rm → Remove arquivos
 - ps → Lista processos

Abordaremos ainda nesta aula sobre alguns dos principais comandos Linux!

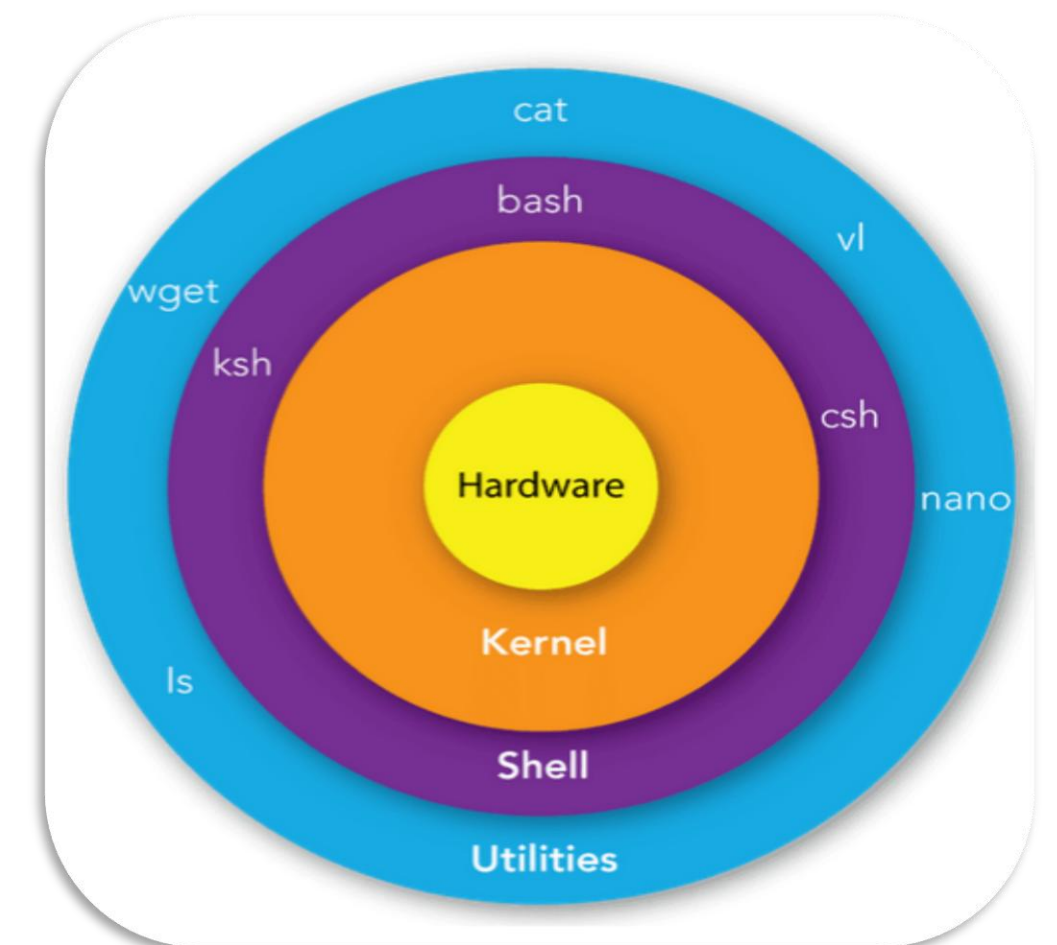


Figura 1 – Shell e comandos no contexto de comunicação com hardware no sistema Linux, fonte: mindmajix.com

Utilities ou Aplicações

(Os Programas que o Usuário Usa 🗂️)

📌 Definição:

As utilities ou Aplicações (**Applications**) são os programas e aplicativos que usamos no dia a dia, como **navegadores, editores de texto, reprodutores de vídeo, etc.**

📌 Exemplos de Aplicativos no Linux:

- **Firefox** (Navegador) 🌐
- **VLC** (Reprodutor de vídeo) 🎬
- **LibreOffice** (Editor de documentos) 📄
- **GIMP** (Editor de imagens) 🎨

📌 Função:

Os aplicativos permitem que o **usuário interaja com o sistema** sem precisar usar o terminal.

📌 Exemplo:

Se você deseja **editar uma foto**, pode abrir o **GIMP** ao invés de usar comandos no terminal.

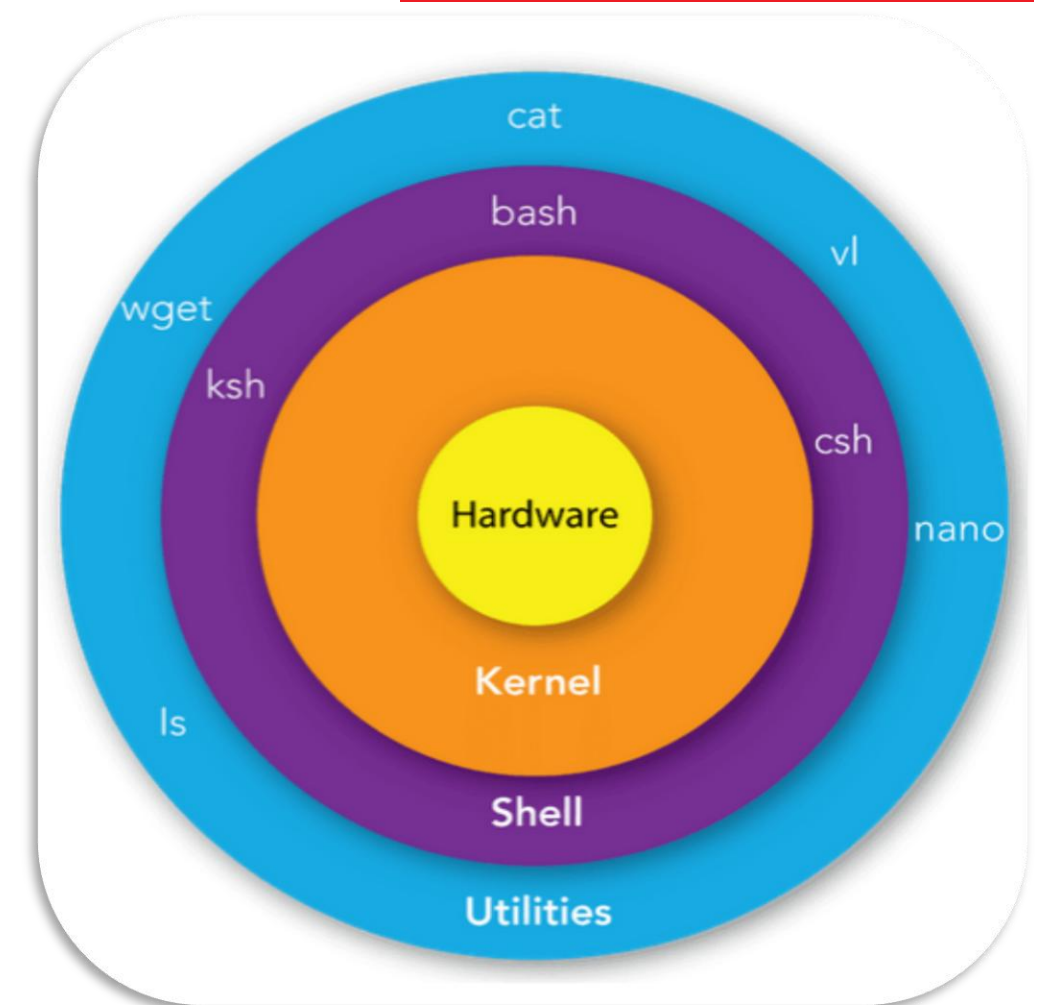


Figura 1 – Shell e comandos no contexto de comunicação com hardware no sistema Linux, fonte: mindmajix.com









RESUMO

O Linux segue uma **arquitetura em camadas**, onde cada nível tem um papel fundamental.

Podemos representá-la assim:

Hierarquia do Linux (de baixo para cima):

-  **[Usuário]:** Quem interage com o sistema
-  **[Aplicações]:** Programas como navegadores, editores, etc.
-  **[Shell]:** Interface entre usuário e sistema
-  **[Low-Level Utilities]:** Ferramentas essenciais do sistema
-  **[Kernel]:** O "cérebro" do sistema operacional
-  **[Hardware]** CPU, memória, disco, etc.

- ✓ **1. O que é o Terminal?** (Interface para o usuário Linux)
- ✓ **2. O que é o Shell?** (Programa que interpreta comandos digitados no terminal)
- ✓ **3. O que é o Bash?** (Shell mais usado no shell Linux)
- ✓ **4. O que é o Root?** (Usuário Administrador com controle total)
- ✓ **5. O que é o Sudo?** (Comando com permissões temporárias de admin)

Arquitetura Unix (A Base do Linux 📌)

📌 Definição:

O Linux segue a **arquitetura Unix**, um modelo de sistema operacional baseado em **simples componentes que trabalham juntos**.

📌 Princípios da Arquitetura Unix:

- ✓ **Tudo é um arquivo** → Pastas, dispositivos, processos, tudo é tratado como um arquivo
- ✓ **Programas pequenos fazem uma coisa bem feita** → Comandos simples que podem ser combinados para tarefas complexas.
- ✓ **Interação via linha de comando (CLI)** → O terminal é uma ferramenta poderosa para manipular o sistema.

📌 Relacionando com o Linux:

- Linux é baseado no Unix e mantém essa **estrutura modular**.
- O **Kernel Linux** gerencia o hardware e os processos.
- O **Bash** permite que o usuário interaja com o sistema.
- Comandos como grep, who, ps, ls e more ajudam a acessar informações.

Root /



Root (o "Dono" do Sistema 👑)

📌 Definição:

O Root é o **usuário administrador** do Linux, que tem **total controle** do sistema.

📌 Analogia:

Se o Linux fosse um **prédio**, o **usuário comum** seria um morador que só pode entrar no seu próprio apartamento, enquanto o **root** seria o zelador, que tem a **chave-mestra** 🔑 para acessar tudo!

📌 Perigo! ⚠️

O root pode **apagar arquivos críticos** do sistema, então **não use essa conta sem necessidade!**

📌 Verificando se você é root:

```
bash
whoami
```

Sudo (Poder de Root Temporário ⚡)

📌 Definição:

O comando sudo (**Super User Do**) permite que um usuário comum execute **comandos administrativos temporariamente**, sem precisar estar logado como root.

📌 Analogia: O **sudo** é como um **passse VIP** 🎫 para acessar certas áreas restritas do prédio (**sistema**), mas apenas para tarefas específicas.

📌 Exemplo:

Se um usuário comum **tentar instalar um programa sem sudo**, ele verá um erro:

```
bash
apt install git
```

Mas **se usar sudo**, ele terá permissão para instalar:

```
bash
sudo apt install git
```

📌 Dica:

O sudo sempre pede a senha do usuário antes de conceder permissões.

Fonte: <https://www.sudo.ws/docs/man/1.8.15/sudo.man/>



Historicamente, o comando Sudo foi concebido e implementado pela primeira vez por Bob Coggeshall e Cliff Spencer por volta de 1980 no Departamento de Ciência da Computação da SUNY/Buffalo. Ele rodava em um VAX-11/750 rodando 4.1BSD. Depois disso, uma outra versão atualizada, creditada a Phil Betchel, Cliff Spencer, Gretchen Phillips, John LoVerso e Don Gworek, foi postada no grupo de notícias Usenet net.sources em dezembro de 1985.

Ao longo dos anos novas versões e mudanças foram surgindo até que em 1994, após manter o sudo informalmente dentro do CU-Boulder por algum tempo, Todd C. Miller fez um lançamento público do "CU sudo" (versão 1.3) com correções de bugs e suporte para mais sistemas operacionais. O "CU" foi adicionado para diferenciá-lo da versão "oficial" do "The Root Group". Depois, em 1999, o prefixo "CU" foi retirado do nome, já que não havia lançamento formal do sudo pelo "The Root Group" desde 1991 (os autores originais agora trabalham em outro lugar). A partir da versão 1.6, o Sudo não continha mais nenhum código original do "Root Group" e está disponível sob uma licença estilo ISC.

Em 2005, Todd reescreveu o parser sudoers para melhor suportar os recursos que foram adicionados nos últimos dez anos. Este novo parser remove algumas limitações do anterior, remove restrições de ordenação e adiciona suporte para incluir múltiplos arquivos sudoers. Em 2010, a Quest Software começou a patrocinar o desenvolvimento do Sudo ao contratar Todd para trabalhar no Sudo como parte de seu trabalho de tempo integral. Isso permitiu a adição de registro de E/S, a API do plugin, o servidor de registro, testes adicionais de regressão e fuzz, suporte para pacotes binários e lançamentos mais regulares. O patrocínio da Quest ao Sudo terminou em fevereiro de 2024.

O Sudo, em sua forma atual é mantido por: Todd C. Miller Todd.Miller@sudo.ws

Fonte:

<https://www.sudo.ws/about/history/>

Principais comandos Linux

A seguir, algumas coisas básicas que podemos fazer com o sudo (**Super User Do**):

Alterar data

Por exemplo, para alterar a data do sistema para 01 de janeiro de 2025 precisaremos ir até o terminal Linux e digitar:

sudo date +%Y%m%d -s "20250101"

```
aluno@aluno-Virtual-Machine:~$ sudo date +%Y%m%d -s "20250101"
```

Alterar hora

Por exemplo, para alterar o horário do sistema para "08:00:00" precisaremos ir até o terminal Linux e digitar:

sudo date +%T -s "08:00:00"

```
aluno@aluno-Virtual-Machine:~$ sudo date +%T -s "08:00:00"
```

Principais comandos Linux

A seguir, algumas coisas básicas que podemos fazer com o sudo (**Super User Do**):

Alterar data

Por exemplo, para alterar a data e hora do sistema para 8h do dia 01 de janeiro de 2025 precisaremos ir até o terminal Linux e digitar: **sudo date -s "2025-01-01 08:00:00"**

```
aluno@aluno-Virtual-Machine:~$ sudo date -s "2025-01-01 08:00:00"
```

A necessidade de que DATA e HORA do sistema operacional estejam atualizadas pode impactar, por exemplo, em problemas com:

❖ **Certificados SSL/TLS e segurança**

- Conexões HTTPS dependem de certificados SSL com datas de validade.
- Se o relógio do sistema estiver alterado, o sistema pode considerar um certificado como expirado, impedindo acessos seguros.

❖ **Sincronização em redes e servidores**

- Em redes distribuídas, servidores precisam estar sincronizados para evitar conflitos de dados.
- Bancos de dados, servidores web e sistemas em cluster dependem de timestamps corretos.
- Protocolos como Kerberos, que exigem autenticação baseada no tempo, podem falhar se houver diferença de horário entre máquinas.



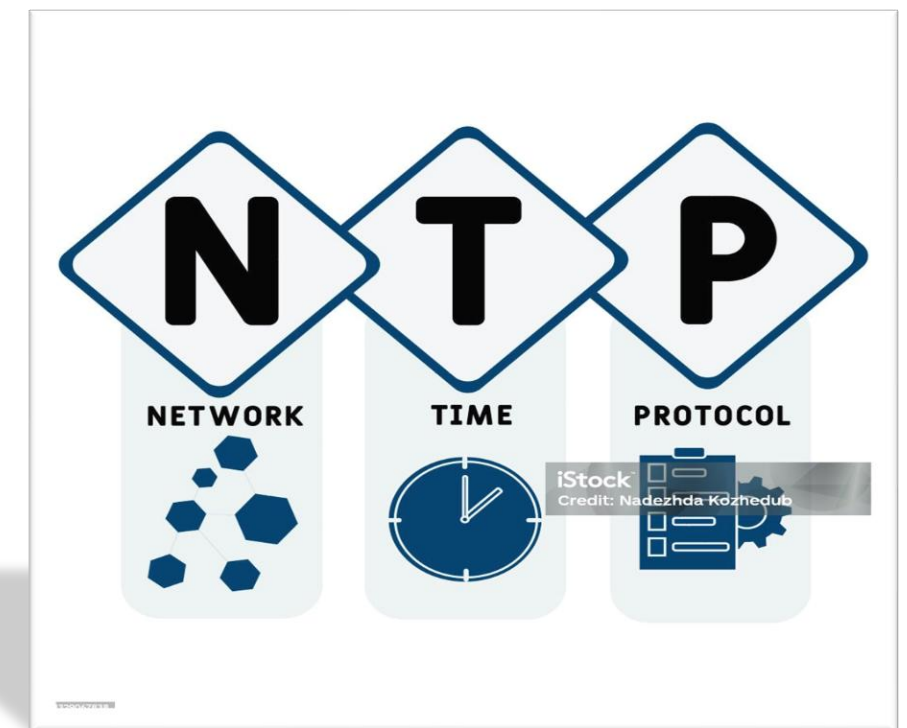
#Fica a Dica

Principais comandos Linux

A seguir, algumas coisas básicas que podemos fazer com o sudo (**Super User Do**):

Timedatectl set-ntp on

O comando `sudo timedatectl set-ntp on` ativa a sincronização automática da data e hora do sistema com servidores NTP (Network Time Protocol). Isso significa que o Linux ajustará seu relógio periodicamente com base em fontes de tempo confiáveis na internet, garantindo precisão e evitando discrepâncias que possam afetar logs, transações, autenticação e outros serviços dependentes do horário correto. Esse comando é especialmente útil para servidores e sistemas distribuídos, onde a sincronização precisa entre máquinas é essencial para o funcionamento adequado.



```
aluno@aluno-Virtual-Machine:~$ sudo timedatectl set-ntp on
```

Para verificar o status do serviço de sincronização:

Se precisar desativar a sincronização para testar alterações manuais:

```
timedatectl status
```

```
sudo timedatectl set-ntp off
```


Principais comandos Linux

A seguir, algumas coisas básicas que podemos fazer com o sudo (**Super User Do**):

Fusos horários

Para listar os fusos horários disponíveis no SO precisaremos ir até o terminal Linux e digitar:

timedatectl list-timezones

```
aluno@aluno-Virtual-Machine:~$ sudo date +%Y%m%d -s "20250101"
```

Definindo fuso horário

Para definirmos o fuso horário para America/São_Paulo devemos ir até o terminal digitar:

sudo timedatectl set-timezone America/Sao_Paulo

```
aluno@aluno-Virtual-Machine:~$ sudo timedatectl set-timezone America/Sao_Paulo
```


Arquitetura Unix/Linux

Principais comandos e ferramentas

Application Program

Shell: é um programa de linha de comando que permite ao usuário interagir com o sistema operacional, executando comandos e scripts. Ele interpreta e executa os comandos digitados no terminal.

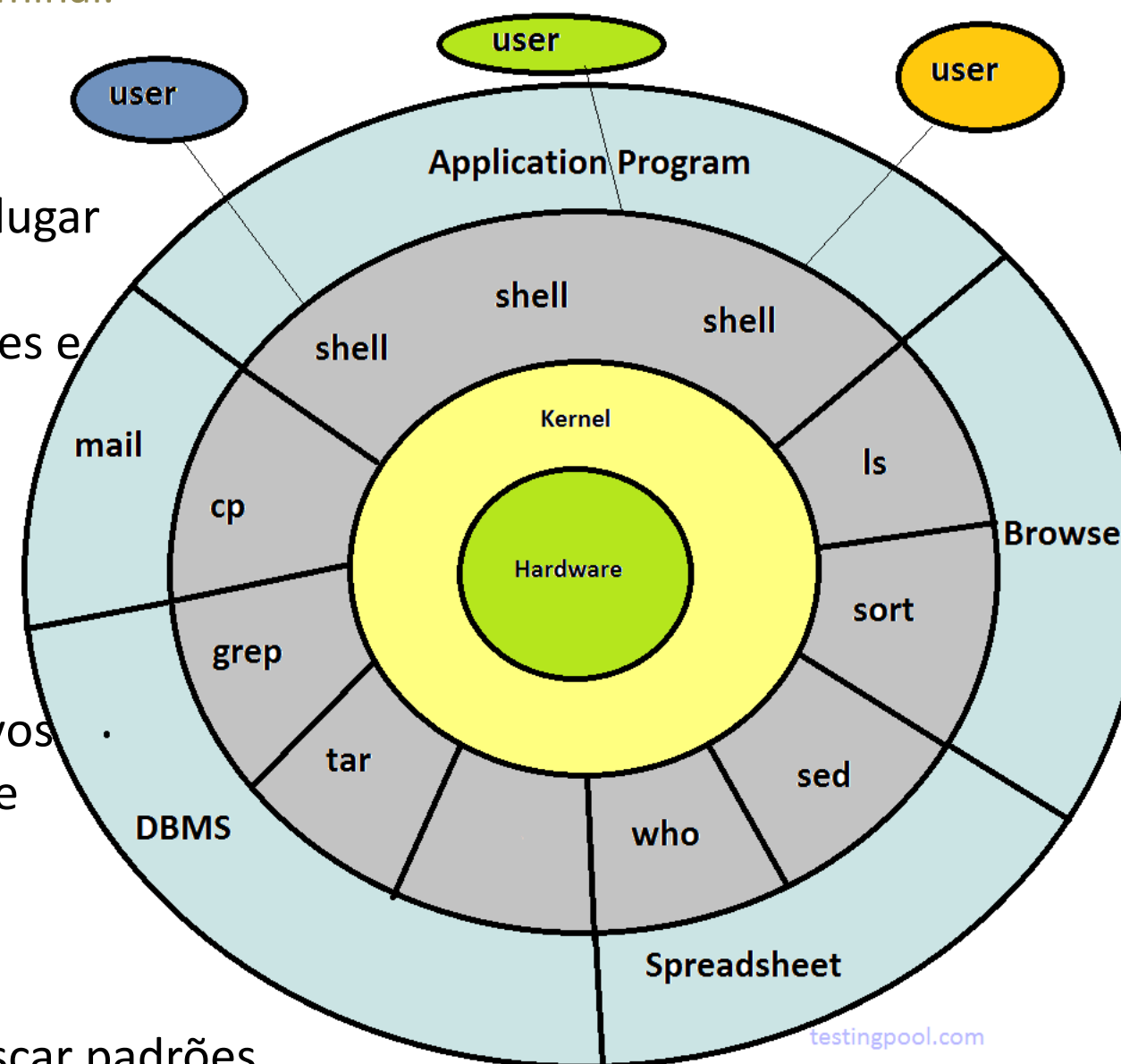
MANIPULAÇÃO DE DADOS

cp: O **cp** é um comando usado para copiar arquivos ou diretórios de um lugar para outro no sistema de arquivos, mantendo ou modificando permissões e propriedades do arquivo conforme necessário.

GESTÃO DE DADOS

tar: O **tar** é um comando utilizado para criar, extrair e manipular arquivos de arquivo compactados, geralmente usados para empacotar múltiplos arquivos em um único arquivo.

grep: é um comando usado para buscar padrões específicos de texto dentro de arquivos, retornando as linhas que correspondem ao padrão buscado.



COMANDOS DE BUSCA

ls: é um comando usado para listar o conteúdo de diretórios no Linux. Ele mostra os arquivos e pastas presentes no diretório atual ou em um diretório especificado.

sort: é um comando que organiza linhas de texto em arquivos ou na saída de outros comandos em ordem crescente ou decrescente.

EDITOR DE TEXTO

sed (Stream Editor): é uma ferramenta de manipulação de texto no Linux, usada para editar arquivos de forma automática e eficiente, como substituir ou excluir texto em fluxos de dados.

who: é um comando que exibe informações sobre os usuários logados no sistema, como o nome do usuário, o terminal e o horário de login.



SOP Abertos: LINUX

Principais Comandos

Abordaremos os principais comandos mais importantes do Linux, do básico ao avançado. Também forneceremos algumas dicas sobre como praticar e aprender comandos do Linux. Estas dicas são para iniciantes e profissionais experientes.

CMD	DESCRIÇÃO	OPÇÃO	EXEMPLOS
whoami	fornece uma resposta rápida, exibindo o nome de usuário associado ao usuário que emitiu o comando.		•whoami
id	Exime nome do usuário e grupo		id
who	lista de nomes de usuários, IDs de terminais, horários de login e endereços IP de origem, se aplicável		•who



SOP Abertos: LINUX

Principais Comandos

1. Comandos de operações de arquivo e diretório

Operações de arquivo e diretório são fundamentais para trabalhar com o sistema operacional Linux. Aqui estão alguns comandos de operações de Arquivo e Diretório comumente usados:

CMD	DESCRIÇÃO	OPÇÃO	EXEMPLOS
ls	Listar arquivos e diretórios.	<ul style="list-style-type: none">•-l : Listagem de formato longo.•-a : Incluir arquivos ocultos•-h : Tamanhos de arquivo legíveis por humanos.	<ul style="list-style-type: none">•ls -l exibe arquivos e diretórios com informações detalhadas.•ls -a mostra todos os arquivos e diretórios, incluindo•ls -lh exibe os tamanhos dos arquivos em um formato legível por humanos.
cd	Alterar diretório.		<ul style="list-style-type: none">•cd /caminho/para/diretório altera o diretório atual para o caminho especificado.
pwd	Imprima o diretório de trabalho atual.		<ul style="list-style-type: none">•pwd exibe o diretório de trabalho atual.
mkdir	Crie um novo diretório.		<ul style="list-style-type: none">•mkdir my_directory cria um novo diretório chamado “my_directory”.
rm rmdir	Remover arquivos e diretórios.	<ul style="list-style-type: none">•-r : Remove diretórios recursivamente.•-f : Força a remoção sem confirmação.	<ul style="list-style-type: none">•rm file.txt exclui o arquivo chamado “file.txt”.•rm -r my_directory exclui o diretório “my_directory” e seu conteúdo.•rm -f file.txt exclui à força o arquivo “file.txt” sem confirmação.
clear	Limpa o que foi exibido		<ul style="list-style-type: none">•clear

Principais comandos Linux

whoami

O comando 'whoami' é um utilitário simples, mas poderoso, projetado para revelar o nome de usuário atual associado à sessão de usuário ativa. Quando executado, ele fornece uma resposta rápida, exibindo o nome de usuário associado ao usuário que emitiu o comando.

O comando **whoami** é usado tanto no sistema operacional Unix quanto no sistema operacional Windows.

- É basicamente a concatenação das strings “**who**”, “**am**”, “**i**” como **whoami** .
- Ele exibe o nome de usuário do usuário atual quando este comando é invocado.
- É semelhante a executar o comando id com as opções **-un** .

Imagine que você está trabalhando no seu computador e esquece quem está logado. No Linux, há um truque especial chamado "whoami" que é como perguntar "Ei, computador, quem sou eu agora?" Este artigo explica como esse comando simples funciona e ajuda você a lembrar quem está no comando! Não se preocupe, não será cheio de palavras técnicas confusas. Vamos mantê-lo divertido e fácil de entender, assim como falar com um amigo.

```
aluno@aluno-Virtual-Machine:~$ whoami  
'aluno'
```

Principais comandos Linux

Who

O comando `who` é uma maneira simples e eficaz de exibir informações sobre usuários atualmente logados. Ao digitar `who` no terminal, você receberá uma lista de nomes de usuários, IDs de terminais, horários de login e endereços IP de origem, se aplicável.

O comando `who` é usado para descobrir as seguintes informações:

1. Hora da última inicialização do sistema
2. Nível de execução atual do sistema
3. Lista de usuários logados e muito mais.

```
aluno@aluno-Virtual-Machine:~$ who
```


Principais comandos Linux

Id

Este comando é útil para descobrir as seguintes informações listadas abaixo:

- Nome de usuário e ID de usuário real.
- Descubra o UID específico do usuário.
- Exibe o UID e todos os grupos associados a um usuário.
- Liste todos os grupos aos quais um usuário pertence.
- Exibe o contexto de segurança do usuário atual.

```
aluno@aluno-Virtual-Machine:~$ id
```

Principais comandos Linux

ls

O comando 'ls' no Linux é uma ferramenta poderosa usada para exibir nomes de usuários e grupos junto com seus IDs numéricos (ID do usuário – UID ou ID do grupo – GID) do usuário atual ou de qualquer usuário especificado no sistema. Este comando é particularmente útil para administradores de sistema e usuários que precisam verificar identidades de usuários e permissões associadas.

1. Opções Básicas

ls → Lista arquivos e diretórios no diretório atual.

ls -l → Exibe detalhes como permissões, proprietário, grupo, tamanho e data de modificação.

ls -a → Mostra tudo, incluindo os ocultos (arquivos que começam com .).

ls -h → Exibe tamanhos de arquivos em formato legível (KB, MB, GB).

2. Opções de Ordenação e Formatação

ls -t → Ordena os arquivos por data de modificação (mais recente primeiro).

ls -r → Inverte a ordem da listagem.

ls -S → Ordena os arquivos por tamanho (do maior para o menor).

ls -X → Ordena os arquivos por extensão.

ls -1 → Lista os arquivos em uma única coluna.

3. Opções Avançadas

ls -lh → Combina -l e -h para exibir detalhes e tamanhos legíveis.

ls -la → Combina -l e -a, mostrando todos os arquivos com detalhes.

ls -d */ → Lista apenas diretórios.

ls -i → Mostra o número do inode de cada arquivo.

ls -R → Lista arquivos e subdiretórios recursivamente.

Principais comandos Linux

cd

O comando 'cd' permite que os usuários alterem seu diretório de trabalho atual dentro do sistema de arquivos. A sintaxe básica do comando 'cd' é a seguinte:

```
aluno@aluno-Virtual-Machine:~$ cd [directory]
```

Aqui, substitua **[directory]** pelo caminho do diretório de destino para o qual você deseja navegar. Se nenhum diretório for especificado, 'cd' redirecionará para seu diretório inicial por padrão. Vamos explorar a funcionalidade do comando por meio de exemplos.



Principais comandos Linux

mkdir

O comando 'mkdir' é como uma varinha mágica para criar pastas super facilmente. 'mkdir' significa "criar diretório" e ajuda você a organizar as coisas do seu computador criando pastas com apenas um comando. Não importa se você está criando uma pasta ou várias delas em sequência, 'mkdir' está lá para ajudar você a manter as coisas limpas e organizadas no seu computador. Neste guia, falaremos sobre como usar 'mkdir', quais palavras digitar e alguns truques legais para deixar suas pastas do jeito que você quer no Linux.

Este comando pode criar vários diretórios de uma vez, bem como definir as permissões para os diretórios. É importante notar que o usuário que executa este comando deve ter permissão suficiente para criar um diretório no diretório pai, ou ele/ela pode receber um erro de 'permissão negada'.

```
aluno@aluno-Virtual-Machine:~$ mkdir [opções...] [nome_do_diretório]
```

OPÇÕES	DESCRIÇÃO
-help	Exibe opções de ajuda sobre determinado comando
-version	Exibe o número da versão e informações adicionais sobre a licença para mkdir
-v or verbose	Habilita o modo verbose, exibindo uma mensagem para cada diretório criado. Quando usado com o argumento [directories], ele mostra os nomes dos diretórios que estão sendo criados.
-p	Um sinalizador que permite a criação de diretórios pais conforme necessário.
-m	Define modos de arquivo ou permissões para os diretórios criados. A sintaxe segue a do comando chmod.

SOP Abertos: LINUX

Principais comandos Linux

rm

rm significa **remover** here. rm comando é usado para remover objetos como arquivos, diretórios, links simbólicos e assim por diante do sistema de arquivos como o UNIX. Para ser mais preciso, rm remove referências a objetos do sistema de arquivos, onde esses objetos podem ter várias referências (por exemplo, um arquivo com dois nomes diferentes). **Por padrão, ele não remove diretórios.** Este comando normalmente funciona silenciosamente e você deve ter muito cuidado ao executar o comando **rm**, porque depois de excluir os arquivos, você não poderá recuperar o conteúdo dos arquivos e diretórios. **Sintaxe:**

```
aluno@aluno-Virtual-Machine:~$ rm [OPTION]... FILE...
```

Já quando preciso excluir um diretório digito:

```
aluno@aluno-Virtual-Machine:~$ rm -r [DIRETORIO]
```


Sistema de Arquivos

Arquivos e Diretórios Escondidos

Para esconder os arquivos e diretórios no Linux, basta renomear o seu nome, iniciando-o com um ponto “.”.

Se renomearmos o diretório musicas, iniciando com um “.”, ele não irá aparecer mais na lista de arquivos e diretórios:

```
$ mv musicas .musicas
```

O comando “mv” serve para mover, ou renomear arquivos e diretórios.
Ao listar a pasta, o diretório “musicas” não irá aparecer:

```
$ ls  
Arquivo
```

O ARQUIVO `.bashrc`

O arquivo `.bashrc` é um arquivo de **script** que é executado quando um usuário faz login. O arquivo em si contém uma série de configurações para a sessão de terminal. Isso inclui configurar ou habilitar: coloração, conclusão, histórico de shell, aliases de comando e muito mais. Ele é um **arquivo oculto**¹ e um simples comando `ls` não mostrará o arquivo. Para visualizar arquivos ocultos, você pode executar o comando abaixo:

```
$ ls -a
```

ou

```
$ ls -al
```

¹**Arquivos ocultos** no Linux são os arquivos que não são listados quando o usuário executa o comando `ls`. O nome de um arquivo oculto começa com um ponto(.) No Linux, não apenas arquivos, mas diretórios também podem ser ocultos. Arquivos são ocultos no Linux por muitos motivos. Um deles é garantir que os usuários não modifiquem acidentalmente o conteúdo desses arquivos. Outro pode ser evitar a exclusão acidental desses arquivos. Em sistemas compartilhados, arquivos podem ser ocultos por questões de privacidade.

O ARQUIVO `.bashrc`

O arquivo `.bashrc` tem muitos comentários que o tornam inteligível. Para visualizar o arquivo `.bashrc` basta digitar:

```
$ cat .bashrc
```

ou

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

`bashrc` pode ser usado para definir funções que reduzem esforços redundantes. Essas funções podem ser uma coleção de comandos básicos. Essas funções podem até usar argumentos do terminal. Vamos definir uma função que informa a data de uma maneira mais descritiva. Primeiro você precisará inserir o arquivo `.bashrc` no modo de edição.

```
$ vi .bashrc
```

O **Nano** é um editor de texto simples e intuitivo para sistemas Linux, amplamente utilizado por sua facilidade de uso, especialmente por iniciantes. Ele é baseado no **Pico**, o editor do cliente de e-mail Pine, e oferece uma interface minimalista que permite edição rápida diretamente no terminal. Com suporte a atalhos de teclado fáceis de memorizar, como **Ctrl + O para salvar** e **Ctrl + X para sair**, o Nano é uma alternativa leve a editores mais complexos como **Vim** e **Emacs**. Apesar de sua simplicidade, possui recursos úteis como realce de sintaxe, busca e substituição, além de numeração de linhas, tornando-se uma ferramenta prática para edição de arquivos de configuração e scripts.

Para começar a edição, pressione qualquer letra do teclado. No final do arquivo, adicione o código ou PATH necessário. Para refletir as alterações no bash, saia e inicie o terminal novamente. Ou use o comando:

```
$ source .bashrc
```